

Otimização não linear

Irrestrita

Prof. Dr. Clodoaldo A M Lima

Escola de Artes Ciências e Humanidades / Sistema de Informação

26 de agosto de 2025

Agenda

- 1 Definição do problema
- 2 Ótimo local, global
- 3 Condições de otimalidade para problemas irrestritos
- 4 Gradiente descendente
 - Busca Linear
- 5 Método de Newton
- 6 Método de Levenberg Marquardt - LM
- 7 Método de Davidson-Fletcher-Powell - DFD
 - Método de Newton com Positivação da Hessiana

Tipos de problema de otimização

Problema de otimização irrestrita

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$

s. t. $x \in X.$

onde $x = [x_1, \dots, x_n] \in \mathbb{R}^n$, $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$, e X é um conjunto fechado (usualmente $X = \mathbb{R}^n$)

Problema de otimização restrita

$$(P_2) \quad \underset{x}{\text{minimize}} \quad f(x)$$

subject to $\begin{aligned} g_i(x) &\leq 0, \quad i = 1, \dots, m. \\ h_i(x) &= 0, \quad i = 1, \dots, l. \\ x &\in X, \end{aligned}$

onde $g_1(x), \dots, g_m(x)$, $h_1(x), \dots, h_l(x) : \mathbb{R}^n \rightarrow \mathbb{R}$

Seja $g(x) = (g_1(x), \dots, g_m(x)) : \mathbb{R}^n \rightarrow \mathbb{R}^m$, $h(x) = (h_1(x), \dots, h_l(x)) : \mathbb{R}^n \rightarrow \mathbb{R}^l$

Preliminares

Então (P_2) pode ser escrito como

$$(P_2) \quad \underset{x}{\text{minimize}} \quad f(x)$$

subject to

$$g(x) \leq 0,$$
$$h(x) = 0.$$
$$x \in X,$$

Nós dissemos que x é uma solução factível de (P_2) , se $g(x) \leq 0$, $h(x) = 0$, e $x \in X$

Ótimo local, global

A bola centrada em \bar{x} com raio ϵ é o conjunto:

$$B(\bar{x}, \epsilon) := \{x \mid \|x - \bar{x}\| \leq \epsilon\}$$

Considere o seguinte problema de otimização sobre o conjunto \mathcal{F}

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$
$$\text{s. t.} \quad x \in \mathcal{F}.$$

Nós temos a seguinte definição de mínimo/máximo, local/global, estrito/não estrito.

Definição 1

$x \in \mathcal{F}$ é um **mínimo local** de P_1 se há existe $\epsilon > 0$ tal que $f(x) \leq f(y)$ para $\forall y \in B(x, \epsilon) \cap \mathcal{F}$

Ótimo local, global

Definição 2

$x \in \mathcal{F}$ é um mínimo global de P_1 se $f(x) \leq f(y)$ para $\forall y \in \mathcal{F}$

Definição 3

$x \in \mathcal{F}$ é um mínimo local estrito de P_1 se há existe $\epsilon > 0$ tal que $f(x) < f(y)$ para $\forall y \in B(x, \epsilon) \cap \mathcal{F}, y \neq x$

Definição 4

$x \in \mathcal{F}$ é um mínimo global estrito de P_1 se $f(x) < f(y)$ para $\forall y \in \mathcal{F}, y \neq x$

Definição 5

$x \in \mathcal{F}$ é um máximo local de P_1 se há existe $\epsilon > 0$ tal que $f(x) \geq f(y)$ para $\forall y \in B(x, \epsilon) \cap \mathcal{F}, y \neq x$

Definição 6

$x \in \mathcal{F}$ é um máximo global de P_1 se $f(x) \geq f(y)$ para $\forall y \in \mathcal{F}, y \neq x$

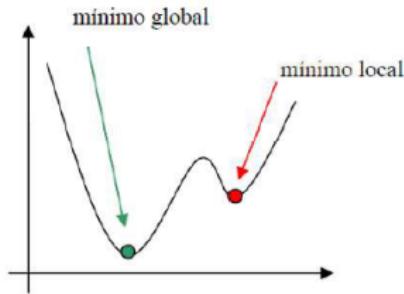
Ótimo local, global

Definição 7

$x \in \mathcal{F}$ é um **máximo local** estrito de P_1 se há existe $\epsilon > 0$ tal que $f(x) > f(y)$ para $\forall y \in B(x, \epsilon) \cap \mathcal{F}, y \neq x$

Definição 8

$x \in \mathcal{F}$ é um **máximo global** estrito de P_1 se há existe $\epsilon > 0$ tal que $f(x) > f(y)$ para $\forall y \in \mathcal{F}, y \neq x$



Gradiente e Hessiana

Seja $f(x) : X \rightarrow \mathbb{R}$, onde $X \subset \mathbb{R}^n$ é fechado.

$f(x)$ é diferenciável em $\bar{x} \in X$ se lá existe um vetor $\nabla f(\bar{x})$ (o gradiente de $f(x)$ em \bar{x}) tal que para cada $x \in X$

$$f(x) = f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) + \|x - \bar{x}\| \alpha(\bar{x}, x - \bar{x}),$$

e $\lim_{y \rightarrow 0} \alpha(\bar{x}, y) = 0$.

$f(x)$ é diferenciável $\forall \bar{x} \in X$. O vetor gradiente é o vetor de derivadas parciais:

$$\nabla f(\bar{x}) = \left[\frac{\partial f(\bar{x})}{\partial x_1}, \dots, \frac{\partial f(\bar{x})}{\partial x_n} \right]^T$$

Exemplo 1

Seja $f(x) = 3x_1^2 x_2^3 + x_2^2 x_3^3$. Então

$$\nabla f(x) = \left[6x_1 x_2^3, 9x_1^2 x_2^2 + 2x_2 x_3^3, 3x_2^2 x_3^2 \right]^T$$

Gradiente e Hessiana

A derivada direcional de $f(x)$ em \bar{x} na direção d é:

$$\lim_{\lambda \rightarrow 0} \frac{f(\bar{x} + \lambda) - f(\bar{x})}{\lambda} = \nabla f(\bar{x})^T d$$

A função $f(x)$ é duas diferenciável em $\bar{x} \in X$ se existe um vetor $\nabla f(\bar{x})$ e uma matriz simétrica $n \times n$ $H(\bar{x})$ (a Hessiana de $f(x)$ em \bar{x}) tal que para cada $x \in X$

$$f(x) = f(\bar{x}) + \nabla f(\bar{x})^T (x - \bar{x}) + \frac{1}{2} (x - \bar{x})^T H(\bar{x}) (x - \bar{x}) + \|(x - \bar{x})\|^2 \alpha(\bar{x}, x - \bar{x}),$$

e $\lim_{y \rightarrow 0} \alpha(\bar{x}, y) = 0$.

$f(x)$ é duas vezes diferenciável sobre X se $f(x)$ é duas vezes diferenciável para todo $\bar{x} \in X$. A Hessiana é a matriz de derivada segunda parcial.

$$H(\bar{x})_{ij} = \frac{\partial^2 f(\bar{x})}{\partial x_i \partial x_j}$$

Exemplo # 1

Seja $f(x) = 3x_1^2x_2^3 + x_2^2x_3^3$. Então

$$\nabla f(x) = \begin{bmatrix} 6x_1x_2^3 \\ 9x_1^2x_2^2 + 2x_2x_3^3 \\ 3x_2^2x_3^2 \end{bmatrix}.$$

$$H(x) = \begin{bmatrix} 6x_2^3 & 18x_1x_2^2 & 0 \\ 18x_1x_2^2 & 18x_1^2x_2 + 2x_3^3 & 6x_2x_3^2 \\ 0 & 6x_2x_3^2 & 6x_2^2x_3 \end{bmatrix}.$$

Matriz Definida Positiva e Semidefinida Positiva

Um matriz M $n \times n$ é chamada

- Definida positiva se $x^T Mx > 0 \forall x \in \mathbb{R}^n, x \neq 0$
- Semidefinida positiva se $x^T Mx \geq 0 \forall x \in \mathbb{R}^n, x \neq 0$
- Definida negativa se $x^T Mx < 0 \forall x \in \mathbb{R}^n, x \neq 0$
- Semidefinida negativa se $x^T Mx \leq 0 \forall x \in \mathbb{R}^n, x \neq 0$
- Indefinida se este $x, y \in \mathbb{R}^n$ para o qual $x^T Mx > 0$ e $y^T My < 0$

Nós dissemos que M é *SPD* se M é simétrica e definida positiva. Similarmente, nós dissemos que M é *SPSD* se M é simétrica e semidefinida positiva.

Example 3

$$M = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

é definida positiva

Exemplo # 2

$$M = \begin{bmatrix} 8 & -1 \\ -1 & 1 \end{bmatrix}$$

é definida positivo. Para ver isto, observe que para $x \neq 0$,

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x^T M x = 8x_1^2 - 2x_1x_2 + x_2^2 = 7x_1^2 + (x_1 - x_2)^2 > 0$$

Existência de soluções ótimas

Muitos dos tópicos aqui estão concentrados com

- existência de soluções ótimas
- caracterização de solução ótima
- algoritmos para computar a solução ótima

Para ilustrar a questão surgida no primeiro tópico, considere o seguinte problema de optimização:

$$(P_2) \quad \underset{x}{\text{minimize}} \quad \frac{1+x}{2x}$$

subject to $x \geq 1,$

Aqui não há solução ótima por que a região factível não é limitada

Existência de soluções ótimas

$$(P_2) \quad \underset{x}{\text{minimize}} \quad \frac{1}{x}$$

subject to $1 \geq x < 2,$

Aqui não há solução ótima por que a região factível não é fechada ($x < 2$)

$$(P_2) \quad \underset{x}{\text{minimize}} \quad f(x)$$

subject to $1 \leq x \leq 2,$

$$f(x) = \begin{cases} 1/x, & \text{if } x < 2 \\ 1, & \text{if } x = 2 \end{cases}$$

Condições de otimalidade para problemas irrestritos

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$

s. t. $x \in X.$

onde $x = [x_1, \dots, x_n] \in \mathcal{R}^n$, $f(x) : \mathcal{R}^n \rightarrow \mathcal{R}$, e X é um conjunto fechado (usualmente $X = \mathcal{R}^n$)

Definição

A direção \bar{d} é chamada uma direção descendente de $f(x)$ em $x = \bar{x}$ se

$$f(\bar{x} + \epsilon \bar{d}) < f(\bar{x})$$

$\forall \epsilon > 0$ e suficientemente pequeno

Condições de otimalidade para problemas irrestritos

Teorema

Supondo que $f(x)$ seja diferenciável em \bar{x} . Se há um vetor d tal que $\nabla f(\bar{x})^T d < 0$, então $\forall \lambda > 0$ e suficientemente pequeno, $f(\bar{x} + \lambda d) < f(\bar{x})$, e assim d é uma direção descendente de $f(x)$ em \bar{x} .

Prova

$$f(\bar{x} + \lambda d) = f(\bar{x}) + \lambda \nabla f(\bar{x})^T d + \lambda \|d\| \alpha(\bar{x}, \lambda d),$$

onde $\alpha(\bar{x}, \lambda d) \rightarrow 0$ como $\lambda \rightarrow 0$. Rearranjando

$$\frac{f(\bar{x} + \lambda d) - f(\bar{x})}{\lambda} = \nabla f(\bar{x})^T d + \|d\| \alpha(\bar{x}, \lambda d),$$

Uma vez que $\nabla f(\bar{x})^T d < 0$ e $\alpha(\bar{x}, \lambda d) \rightarrow 0$ quando $\lambda \rightarrow 0$, $f(\bar{x} + \lambda d) - f(\bar{x}) < 0$ $\forall \lambda > 0$ suficientemente pequeno.

Condições de otimalidade para problemas irrestritos

Corolário

Supondo $f(x)$ seja diferenciável em \bar{x} . Se \bar{x} é um mínimo local, então $\nabla f(\bar{x}) = 0$.

Prova

Se fosse verdade que $\nabla f(\bar{x}) \neq 0$, então $d = -\nabla f(\bar{x})$ seria uma direção descendente, ao passo que \bar{x} não será um mínimo local.

Importante

O corolário acima é a condição necessária de primeira ordem para um problema minimização irrestrito. O teorema a seguir é uma condição de otimalidade de segunda ordem

Condições de otimalidade para problemas irrestritos

Teorema

Suponha que $f(x)$ seja duas vezes diferenciável em $\bar{x} \in X$. Se \bar{x} é um mínimo local, então $\nabla f(\bar{x}) = 0$ e $H(\bar{x})$ é semidefinida positiva.

Prova

Da condição necessária de primeira ordem, $\nabla f(\bar{x}) = 0$. Suponha que $H(\bar{x})$ não é definida positiva. Então lá existe d tal que $d^T H d < 0$. Nós temos

$$f(\bar{x} + \lambda d) = f(\bar{x}) + \lambda \nabla f(\bar{x})^T d + \frac{1}{2} \lambda^2 d^T H(\bar{x}) d + \lambda^2 \|d\|^2 \alpha(\bar{x}, \lambda d)$$

$$f(\bar{x} + \lambda d) = f(\bar{x}) + \frac{1}{2} \lambda^2 d^T H(\bar{x}) d + \lambda^2 \|d\|^2 \alpha(\bar{x}, \lambda d)$$

onde $\alpha(\bar{x}, \lambda d) \rightarrow 0$, quando $\lambda \rightarrow 0$. Rearranjando,

$$\frac{f(\bar{x} + \lambda d) - f(\bar{x})}{\lambda^2} = \frac{1}{2} d^T H(\bar{x}) d + \|d\|^2 \alpha(\bar{x}, \lambda d)$$

Uma vez que $d^T H(\bar{x}) d < 0$ e $\alpha(\bar{x}, \lambda d) \rightarrow 0$, $f(\bar{x} + \lambda d) - f(\bar{x}) < 0 \ \forall \lambda > 0$ e suficientemente pequeno, produzindo a contradição desejada.

Exemplo # 3

$$f(x) = \frac{1}{2}x_1^2 + x_1x_2 + 2x_2^2 - 4x_1 - 4x_2 - x_2^3$$

Então

$$\nabla f(x) = \left(x_1 + x_2 - 4, x_1 + 4x_2 - 4 - 3x_2^2 \right)^T,$$

e

$$H(x)) = \begin{bmatrix} 1 & 1 \\ 1 & 4 - 6x_2 \end{bmatrix}$$

$\nabla f(x) = 0$ tem exatamente duas soluções: $\bar{x} = (4, 0)$ e $\tilde{x} = (3, 1)$. Mas

$$H(\tilde{x}) = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix}$$

é indefinida, portanto, a única solução candidata para um mínimo local é $\bar{x} = (4, 0)$

Exemplo # 4

Encontre os candidatos a mínimos e máximos locais da função

$$f(x) = (2x_1 - x_2)^2 + (3x_1 - x_3)^2 + (3x_2 - 2x_3)^2$$

Condições de otimalidade para problemas irrestritos

Teorema

Supondo que $f(x)$ é duas vezes diferenciável em \bar{x} . Se $\nabla f(\bar{x}) = 0$ e $H(\bar{x})$ é definida positiva, então \bar{x} é um mínimo local (estrito).

Prova

Exercício para casa

Observação

- Se $\nabla f(\bar{x}) = 0$ e $H(\bar{x})$ é definida negativa, então \bar{x} é um máximo local
- Se $\nabla f(\bar{x}) = 0$ e $H(\bar{x})$ é semidefinida positiva, nós não podemos dizer com certeza que \bar{x} é um mínimo local

Gradiente descendente

O problema que nós

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$
$$\text{s. t.} \quad x \in \mathcal{R}^n.$$

onde $f(x)$ é diferenciável.

Se $x = \bar{x}$ é um dado ponto, $f(x)$ pode ser aproximado por sua expansão linear

$$f(\bar{x} + d) \approx f(\bar{x}) + \nabla f(\bar{x})^T d$$

Se d é pequeno, isto é, $\|d\|$ é pequeno. Observe que se a aproximação na equação acima é boa, então nós queremos escolher d tal que o produto interno $\nabla f(\bar{x})^T d$ é tão pequena quanto possível. Vamos normalizar d tal que $\|d\| = 1$. Então entre todas as direções d com norma $\|d\| = 1$, a direção

$$\tilde{d} = \frac{-\nabla f(\bar{x})}{\|\nabla f(\bar{x})\|}$$

torna o produto interno menor com o gradiente $\nabla f(\bar{x})$.

Gradiente descendente

Este fato segue da seguinte desigualdade:

$$\nabla f(x)^T d \geq -\|\nabla f(\bar{x})\| \cdot \|d\| = \nabla f(\bar{x})^T \left(\frac{-\nabla f(\bar{x})}{\|\nabla f(\bar{x})\|} \right) = \nabla f(\bar{x})^T \tilde{d}$$

Para esta razão a direção não normalizada:

$$\tilde{d} = -\nabla f(\bar{x})$$

é chamada a direção de maior descida no ponto \bar{x}

Observe que $\tilde{d} = -\nabla f(\bar{x})$ é uma direção descendente enquanto $\nabla f(\bar{x}) \neq 0$.

Para ver isto, simplesmente observe que $d^T \nabla f(\bar{x}) = -(\nabla f(\bar{x}))^T \nabla f(\bar{x}) < 0$ enquanto $\nabla f(\bar{x}) \neq 0$.

Gradiente descendente - Pseudo código

- Passo 0 - Dado x^0 , faça $k := 0$
- Passo 1 - $d^k := -\nabla f(x^k)$. Se $d^k = 0$, então pare
- Passo 2 - Solucione $\min_{\alpha} f(x^k + \alpha d^k)$ para o tamanho do passo α^k , talvez escolhida por uma busca linear exata ou inexata
- Passo 3 - Faça $x^{k+1} \leftarrow x^k + \alpha^k d^k$, $k \leftarrow k + 1$. Vá para o passo 1.

Pseudocódigo - Gradiente Descendente

- Escolher x_0, α, ϵ
- **Repetir** até convergência:
 - Calcular gradiente: $g_k = \nabla f(x_k)$
 - Atualizar: $x_{k+1} = x_k - \alpha g_k$
- **Parar** se $\|g_k\| < \epsilon$

Python - Gradiente Descendente

```
import numpy as np

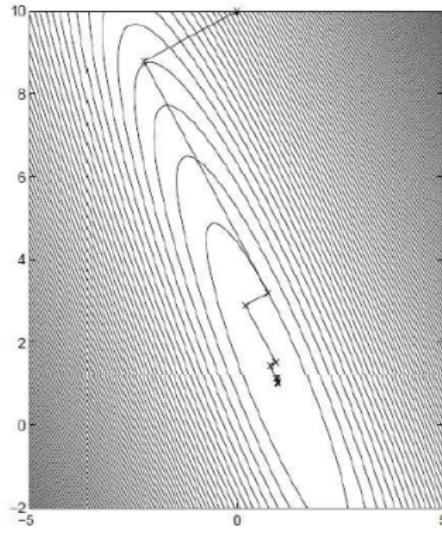
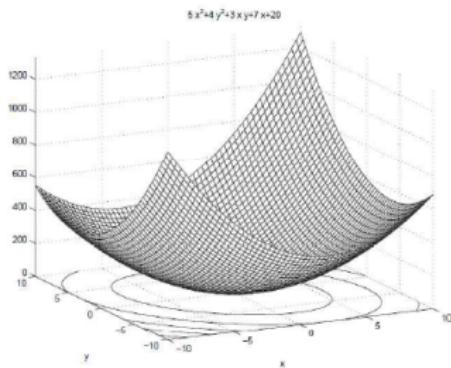
def f(x): return x**2 + 2*x + 1
def grad_f(x): return 2*x + 2

x = 5.0
alpha = 0.1
for _ in range(20):
    x = x - alpha * grad_f(x)
    print(f"x = {x:.4f}, f(x) = {f(x):.4f}")
```

Gradiente descendente - Comportamento típico

$$f(x_1, x_2) = 5x_1^2 + x_2^2 + 4x_1x_2 - 14x_1 - 6x_2 + 20$$

Esta função tem sua solução ótima em $x^* = (x_1^*, x_2^*) = (1, 1)$ e $f(1, 1) = 10$



Busca Linear

Supondo que $f(x)$ é uma função convexa continuamente diferenciável e desejamos solucionar:

$$\bar{\alpha} := \arg \min_{\alpha} f(\bar{x} + \alpha \bar{d}),$$

Busca Linear

O problema

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$

s. t. $x \in X.$

onde \bar{x} é nossa iteração atual, \bar{d} é a direção atual gerada por um algoritmo que busca minimizar $f(x)$

Suponha que \bar{d} é uma direção descendente de $f(x)$ em $x = \bar{x}$, isto é

$$f(\bar{x} + \epsilon \bar{d}) < f(\bar{x})$$

$\forall \epsilon > 0$ e suficientemente pequena.

Algoritmo de Bisseção

Seja

$$h(\alpha) := f(\bar{x} + \alpha \bar{d})$$

$h(\alpha)$ é uma função convexa na variável escala α , e nosso problema é solucionado por

$$\bar{\alpha} = \arg \min_{\alpha} h(\alpha)$$

É elementar mostrar que

$$h'(\alpha) = \nabla f(\bar{x} + \alpha \bar{d})^T d$$

Proposição

$$h'(0) < 0$$

Como $h'(\alpha)$ é uma função crescente monotônica de α , nós temos que:

Proposição

$h'(\alpha)$ é uma função crescente monotônica de α

Definição e Intuição

- O método da bisseção é um algoritmo de busca de raízes baseado no Teorema do Valor Intermediário.
- Parte de um intervalo $[a, b]$ em que $f(a)f(b) < 0$ (ou seja, há pelo menos uma raiz).
- A cada iteração divide o intervalo ao meio e seleciona o subintervalo onde há mudança de sinal.

Algoritmo de Bisseção

Como $h'(\alpha)$ é uma função crescente monotônica, nós podemos aproximadamente computar $\bar{\alpha}$, o ponto que satisfaz $h(\bar{\alpha}) = 0$, por um método de bisseção adequado. Supondo que nós sabemos um valor $\hat{\alpha}$ tal que $h'(\hat{\alpha}) > 0$. Desde que $h'(0) < 0$ e $h'(\hat{\alpha}) > 0$, o valor do meio $\tilde{\alpha} = \frac{0+\hat{\alpha}}{2}$ é um ponto de teste adequado. Observe o seguinte:

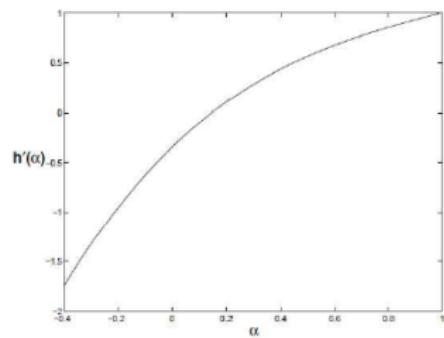
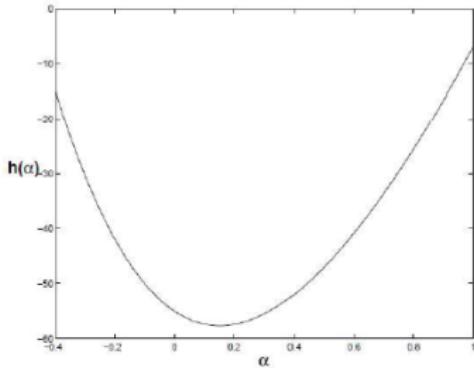
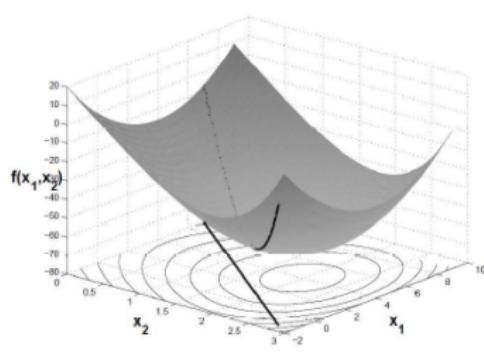
- se $h'(\tilde{\alpha}) = 0$, podemos parar
- se $h'(\tilde{\alpha}) > 0$, nós podemos colocar $\bar{\alpha}$ no intervalo $(0, \tilde{\alpha})$
- se $h'(\tilde{\alpha}) < 0$, nós podemos colocar $\bar{\alpha}$ no intervalo $(\tilde{\alpha}, \bar{\alpha})$

Este conduz para o seguinte algoritmo de bisseção para minimização de $h(\alpha) = f(\bar{x} + \alpha \bar{d})$ por solucionar a equação $h'(\alpha) \approx 0$

Algoritmo de Bisseção

- Passo 0 - Faça $k = 0$, $\alpha_l := 0$, $\alpha_u := \hat{\alpha}$
- Passo 1 - Faça $\tilde{\alpha} = \frac{\alpha_u + \alpha_l}{2}$ e compute $h'(\tilde{\alpha})$
Se $h'(\tilde{\alpha}) > 0$, faça $\alpha_u := \tilde{\alpha}$. Faça $k \leftarrow k + 1$
Se $h'(\tilde{\alpha}) < 0$, faça $\alpha_l := \tilde{\alpha}$. Faça $k \leftarrow k + 1$
Se $h'(\tilde{\alpha}) = 0$, pare

Algoritmo de Bisseção



Algoritmo de Bisseção

Proposição

Após toda iteração do algoritmo bisseção, o intervalo $[\alpha_l, \alpha_u]$ deve conter um ponto $\bar{\alpha}$ tal que $h'(\bar{\alpha}) = 0$

Proposição

Na k -ésima iteração do algoritmo bisseção, o comprimento do intervalo atual $[\alpha_l, \alpha_u]$ é

$$L = \left(\frac{1}{2}\right)^k (\hat{\alpha})$$

Proposição

O valor α tal que $|\alpha - \bar{\alpha}| \leq \epsilon$ pode ser encontrado em no máximo

$$\left\lceil \log_2 \left(\frac{\hat{\alpha}}{\epsilon} \right) \right\rceil$$

passos do algoritmo de bisseção.

Calculando $\hat{\alpha}$ para o qual $h'(\hat{\alpha}) > 0$

Suponde que nós não temos um valor conveniente de $\hat{\alpha}$ para o qual $h'(\hat{\alpha}) > 0$.
Uma forma é pegar um candidato inicial de $\hat{\alpha}$ e calcular $h'(\bar{\alpha})$.
Se $h'(\alpha) > 0$, então proceda para o algoritmo de bisseção; se $h'(\alpha) \leq 0$, então faça
 $\hat{\alpha} \leftarrow 2\hat{\alpha}$ e repita o processo.

Critério de parada para o algoritmo bisseção

Na prática, nós necessitamos executar o algoritmo bisseção com um critério de parada. Alguns critérios de parada relevantes são:

- Pare após um número fixo de iterações. Este é parado quando $k = \bar{K}$, onde \bar{K} é especificado pelo usuário
- Pare quando o intervalo torna-se pequeno. Isto é, pare quando $\alpha_u - \alpha_l \leq \epsilon$, onde ϵ é especificado pelo usuário
- Pare quando $|h'(\tilde{\alpha})|$ torna-se pequeno. Isto, pare quando $|h'(\tilde{\alpha})| \leq \epsilon$, onde ϵ é especificado pelo usuário

Este terceiro critério de parada tipicamente produz o melhor resultado na prática.

Algorithm Cálculo da Razão Áurea (Parte 1)

- 1: **Entrada:** Intervalo $[a, b]$, tolerância ϵ
 - 2: **Saída:** Aproximação do ponto de mínimo da função $f(x)$
 - 3: $\phi \leftarrow \frac{1+\sqrt{5}}{2}$ ▷ Razão áurea
 - 4: $resphi \leftarrow 2 - \phi$
 - 5: $x1 \leftarrow a + resphi \cdot (b - a)$
 - 6: $x2 \leftarrow b - resphi \cdot (b - a)$
 - 7: $f1 \leftarrow f(x1)$
 - 8: $f2 \leftarrow f(x2)$
-

Razão Áurea - Parte 2

Algorithm Cálculo da Razão Áurea (Parte 2)

```
1: while  $|b - a| > \epsilon$  do
2:   if  $f_1 < f_2$  then
3:      $b \leftarrow x_2$ 
4:      $x_2 \leftarrow x_1$ 
5:      $f_2 \leftarrow f_1$ 
6:      $x_1 \leftarrow a + resphi \cdot (b - a)$ 
7:      $f_1 \leftarrow f(x_1)$ 
8:   else
9:      $a \leftarrow x_1$ 
10:     $x_1 \leftarrow x_2$ 
11:     $f_1 \leftarrow f_2$ 
12:     $x_2 \leftarrow b - resphi \cdot (b - a)$ 
13:     $f_2 \leftarrow f(x_2)$ 
14:  end if
15: end while
```

Busca de Passo - Razão Áurea

```
def golden_section_search(f_alpha, a, b, tol=1e-5, max_iter=100):
    phi = (1 + np.sqrt(5)) / 2
    resphi = 2 - phi
    x1 = a + resphi * (b - a)
    x2 = b - resphi * (b - a)
    f1 = f_alpha(x1)
    f2 = f_alpha(x2)

    for _ in range(max_iter):
        if abs(b - a) < tol:
            break
        if f1 < f2:
            b, x2, f2 = x2, x1, f1
            x1 = a + resphi * (b - a)
            f1 = f_alpha(x1)
        else:
            a, x1, f1 = x1, x2, f2
            x2 = b - resphi * (b - a)
            f2 = f_alpha(x2)

    return (a + b) / 2
```

Busca de Passo - Razão Áurea

```
def golden_section_search(f_alpha, a, b, tol=1e-5, max_iter=100):
    phi = (1 + np.sqrt(5)) / 2
    resphi = 2 - phi
    x1 = a + resphi * (b - a)
    x2 = b - resphi * (b - a)
    f1 = f_alpha(x1)
    f2 = f_alpha(x2)

    for _ in range(max_iter):
        if abs(b - a) < tol:
            break
        if f1 < f2:
            b, x2, f2 = x2, x1, f1
            x1 = a + resphi * (b - a)
            f1 = f_alpha(x1)
        else:
            a, x1, f1 = x1, x2, f2
            x2 = b - resphi * (b - a)
            f2 = f_alpha(x2)

    return (a + b) / 2
```

Gradiente com Passo Otimizado

```
def gradient_descent_with_golden_search(x0, tol=1e-6, max_iter=100):
    :
    x = x0.copy()
    for i in range(max_iter):
        g = grad_f(x)
        norm_g = np.linalg.norm(g)
        if norm_g < tol:
            break

        def f_alpha(alpha):
            return f(x - alpha * g)

        alpha_opt = golden_section_search(f_alpha, 0, 1)
        x = x - alpha_opt * g

        print(f"Iter {i+1}: x = {x}, f(x) = {f(x):.6f}, ||grad|| = {norm_g:.6f}")

    return x
```

Execução do Algoritmo

```
x_inicial = np.array([0.0, 0.0])
minimo = gradient_descent_with_golden_search(x_inicial)
print(f"Minimo encontrado: x = {minimo}, f(x) = {f(minimo):.6f}")
```

Busca Linear com restrocesso (Armijo)

Condição de Armijo

Escolher α tal que $f(x_k + \alpha d_k) \leq f(x_k) + c \alpha \nabla f(x_k)^\top d_k$, com $c \in (0, 1)$.

Retrocesso (Backtracking)

Comece com $\alpha \leftarrow \alpha_0$ e reduza $\alpha \leftarrow \rho\alpha$ ($0 < \rho < 1$) até Armijo valer.

Parâmetros típicos: $\alpha_0 \in [10^{-3}, 1]$, $c \approx 10^{-4}$, $\rho \in [0.1, 0.8]$.

Condição de Armijo

A condição de Armijo garante que o tamanho do passo α_k escolhido na direção de descida d_k reduza suficientemente a função $f(x)$:

$$f(x_k + \alpha_k d_k) \leq f(x_k) + c \alpha_k \nabla f(x_k)^{\top} d_k$$

- $0 < c < 1$: parâmetro de redução (ex.: $c = 10^{-4}$).
- d_k : direção de descida ($\nabla f(x_k)^{\top} d_k < 0$).
- Garante que o passo realmente diminua a função de forma útil.

Busca Linear Retroativa (Armijo)

Algorithm Condição de Armijo

- 1: **Entrada:** x_k , direção d_k , $f(x)$, $\nabla f(x)$, parâmetros $c \in (0, 1)$, $\rho \in (0, 1)$
 - 2: **Inicialize:** $\alpha \leftarrow 1$
 - 3: **while** $f(x_k + \alpha d_k) > f(x_k) + c\alpha \nabla f(x_k)^\top d_k$ **do**
 - 4: $\alpha \leftarrow \rho \cdot \alpha$
 - 5: **end while**
 - 6: **Retorne:** α
-

Observações Importantes

- ρ controla a taxa de redução do passo (tipicamente $\rho = 0.5$).
- O método evita passos muito grandes que aumentariam a função.
- Também evita passos muito pequenos que levam a convergência lenta.
- Muito usado em métodos de otimização: Gradiente Descendente, Newton, Quasi-Newton.

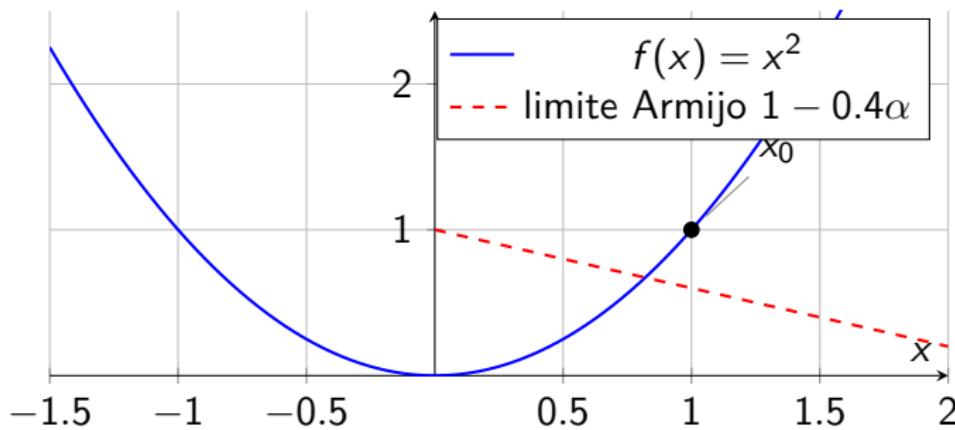
Exemplo Numérico com Backtracking

Seja $f(x) = x^2$, no ponto $x_0 = 1$, direção de descida $d = -2$.

$$f(1 + \alpha(-2)) = (1 - 2\alpha)^2$$

Condição de Armijo ($c = 0.1$):

$$(1 - 2\alpha)^2 \leq 1 - 0.4\alpha$$



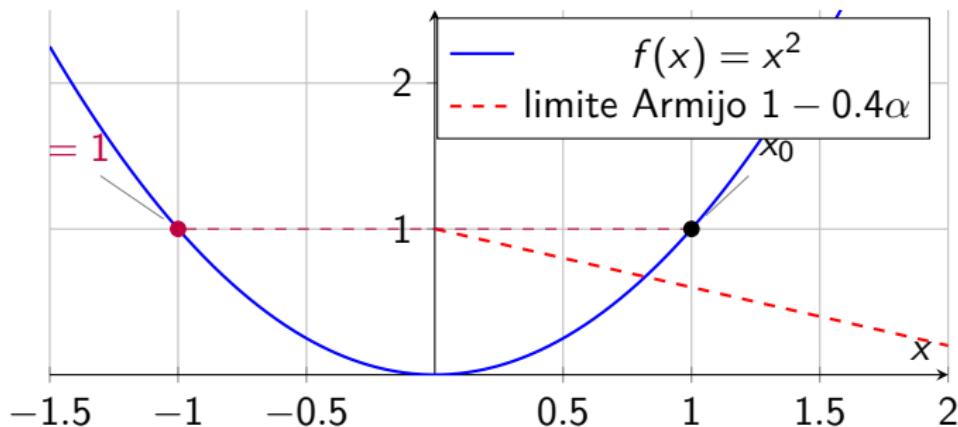
Exemplo Numérico com Backtracking

Seja $f(x) = x^2$, no ponto $x_0 = 1$, direção de descida $d = -2$.

$$f(1 + \alpha(-2)) = (1 - 2\alpha)^2$$

Condição de Armijo ($c = 0.1$):

$$(1 - 2\alpha)^2 \leq 1 - 0.4\alpha$$



Tentativa 1: $\alpha = 1$ $f(-1) = 1$, mas $1 \not\leq 0.6$ (rejeitado).

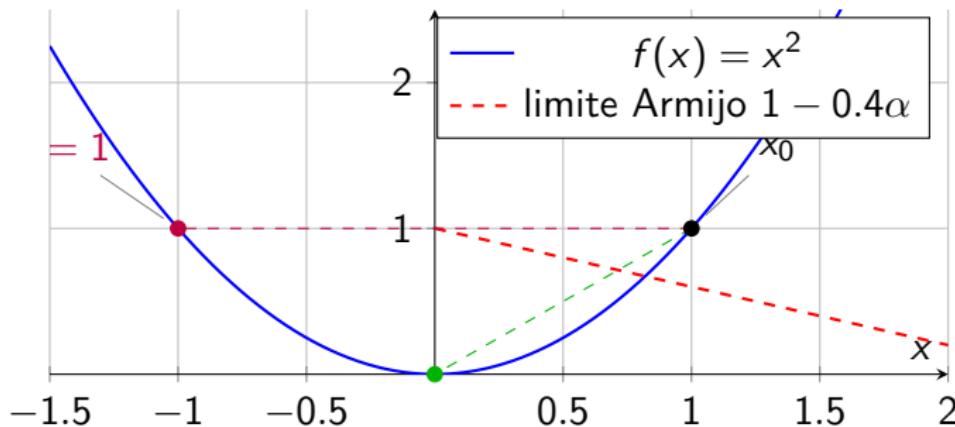
Exemplo Numérico com Backtracking

Seja $f(x) = x^2$, no ponto $x_0 = 1$, direção de descida $d = -2$.

$$f(1 + \alpha(-2)) = (1 - 2\alpha)^2$$

Condição de Armijo ($c = 0.1$):

$$(1 - 2\alpha)^2 \leq 1 - 0.4\alpha$$



Tentativa 2: $\alpha = 0.5$ $f(0) = 0$, e $0 \leq 0.8$ (aceito).

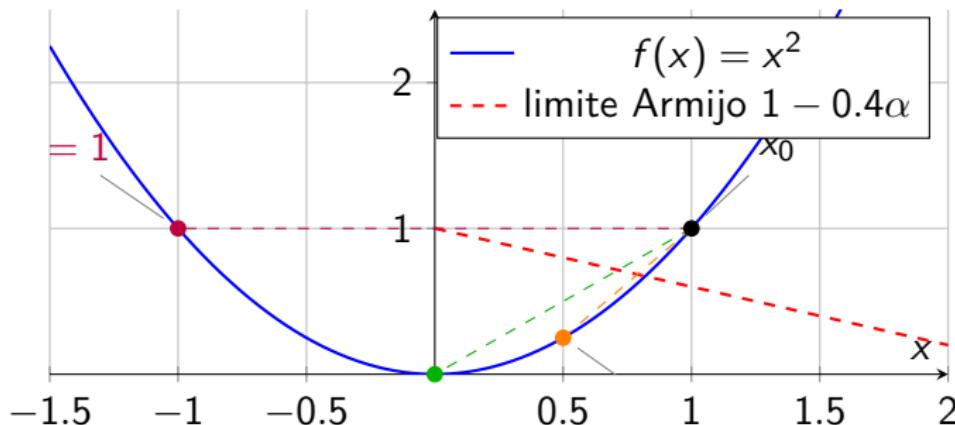
Exemplo Numérico com Backtracking

Seja $f(x) = x^2$, no ponto $x_0 = 1$, direção de descida $d = -2$.

$$f(1 + \alpha(-2)) = (1 - 2\alpha)^2$$

Condição de Armijo ($c = 0.1$):

$$(1 - 2\alpha)^2 \leq 1 - 0.4\alpha$$



Comparação: $\alpha = 0.25$ $f(0.5) = 0.25$, também válido mas passo menor.

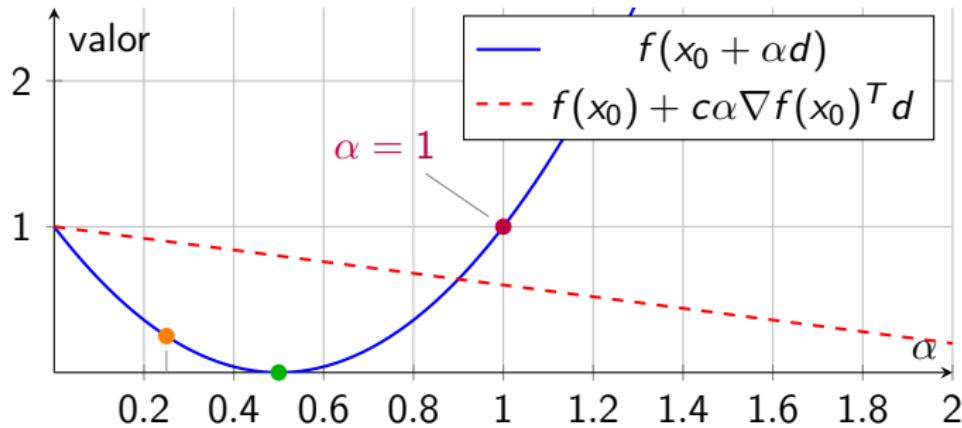
Condição de Armijo em função de α

No exemplo: $x_0 = 1$, $d = -2$, temos

$$f(x_0 + \alpha d) = (1 - 2\alpha)^2$$

e a reta de Armijo:

$$g(\alpha) = 1 - 0.4\alpha$$



Aqui vemos que $\alpha = 1$ falha (curva acima da reta), enquanto $\alpha = 0.5$ e $\alpha = 0.25$ satisfazem a condição.

Método de Newton

Supondo que nós queremos resolver

$$(P_1) \quad \underset{x}{\text{minimize}} \quad f(x)$$

s. t. $x \in X.$

Em $x = \bar{x}$, $f(x)$ pode ser aproximado por:

$$f(x) \approx h(x) := f(\bar{x}) + \nabla f(\bar{x})^T(x - \bar{x}) + \frac{1}{2}(x - \bar{x})H(x)(x - \bar{x}),$$

a qual é uma expansão quadrática de Taylor de $f(x)$ em $x = \bar{x}$. Aqui $\nabla f(x)$ é o gradiente de $f(x)$ e $H(x)$ é a Hessiana de $f(x)$.

Observe que $h(x)$ é uma função quadrática, a qual é minimizada por solucionando $\nabla h(x) = 0$.

Método de Newton

Uma vez que o gradiente de $h(x)$ é:

$$\nabla h(x) = \nabla f(\bar{x}) + H(\bar{x})(x - \bar{x})$$

Nos portanto estamos motivado a solucionar:

$$\nabla f(\bar{x}) + H(\bar{x})(x - \bar{x}) = 0$$

que produz

$$(x - \bar{x}) = -H(\bar{x})^{-1}\nabla f(\bar{x})$$

A direção $-H(\bar{x})^{-1}\nabla f(\bar{x})$ é chamada de direção de Newton, ou passo de Newton em $x - \bar{x}$

Este conduz para o seguinte algoritmo para solução de (P_1)

Método de Newton

- Passo 0 - Dado x^0 , faça $k \leftarrow 0$
- Passo 1 - $d^k = -H(x^k)^{-1}\nabla f(x^k)$. Se $d^k = 0$, então pare.
- Passo 2 - Escolha o tamanho do passo $\alpha^k = 1$
- Passo 3 - Faça $x^{k+1} \leftarrow x^k + \alpha^k d^k$, $k \leftarrow k + 1$. Vá para o passo 1.

Observe o seguinte

- O método assume $H(x^k)$ é não singular a cada iteração
- Não há garantia que $f(x^{k+1}) \leq f(x^k)$
- O passo 2 pode ser aumentado por uma busca linear de $f(x^k + \alpha d^k)$ para encontrar um valor ótimo do tamanho do passo, parâmetro α

Proposição

Se $H(x)$ é simétrico e definido positivo e $d := -H(x)^{-1}\nabla f(x) \neq 0$, então d é uma direção descendente, isto é, $f(x + \alpha d) < f(x)$ para todos os valores de α suficientemente pequeno.

Método de Newton

- A matriz hessiana não precisa ser invertida explicitamente, em vez disso resolve-se o sistema linear.

$$\nabla f(\bar{x}) + H(\bar{x})(x - \bar{x}) = 0$$

$$H(\bar{x})(x - \bar{x}) = -\nabla f(\bar{x})$$

$$H(\bar{x})\delta = -\nabla f(\bar{x})$$

$$x^{k+1} \leftarrow x^k + \alpha^k \delta$$

- Convergência quadrática mas necessita de uma boa estimativa inicial da solução
- Necessita da segunda derivada da função

Método de Newton - Exemplo

Usar o método de Newton para minimizar $f(x) = 0.5x_1^2 + 2.5x_2^2$

Gradiente e matriz hessiana são dados por

$$\nabla f(x) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$$

$$H(x) = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix}$$

Escolhendo $x^0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$ obtemos $\nabla f(x) = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$

Resolvendo o sistema linear $H(x^0)\delta_0 = -\nabla f(x^0)$, obtemos a direção $\delta_0 = \begin{bmatrix} -5 \\ -1 \end{bmatrix}$ e assumindo o passo igual a 1, temos

$$x^1 = x^0 + \alpha_0 * \delta_0$$

$$x^1 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} + 1 * \begin{bmatrix} -5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

esta é uma solução exata do problema.

Método de Newton Modificado

A lei de ajuste do método de Newton torna-se

$$x^{k+1} = x^k - \alpha_k M_k^{-1} \nabla f(x_k)$$

onde

$$\begin{cases} M_k = H(x^k) & \text{se } \lambda_{\min}^k > 0 \\ M_k = H(x^k) + (\epsilon - \lambda_{\min}^k) I & \text{se } \lambda_{\min}^k \leq 0 \end{cases}$$

com λ_{\min}^k o autovalor mínimo de H_k . Ainda não existem resultados que conduzam à determinação automática de um valor ótimo para ϵ

- É importante mencionar ainda que em lugar de positivar H_k poder-se-ia, em princípio, utilizar qualquer outra matriz definida positiva de dimensões apropriadas.
- A razão para optar pelo processo de positivação da hessiana é a analogia com o tradicional método de Levenberg-Marquardt (que será visto a seguir) que pode ser interpretado como uma combinação entre a lei de ajuste do método do gradiente e a lei de ajuste do método de Newton.

Método de Newton Modificado

A lei de ajuste do método de Newton torna-se

$$x^{k+1} = x^k - \alpha_k M_k^{-1} \nabla f(x_k)$$

onde

$$\begin{cases} M_k = H(x^k) & \text{se } \lambda_{min}^k > 0 \\ M_k = H(x^k) + (\epsilon - \lambda_{min}^k) I & \text{se } \lambda_{min}^k \leq 0 \end{cases}$$

com λ_{min}^k o autovalor mínimo de H_k . Ainda não existem resultados que conduzam à determinação automática de um valor ótimo para ϵ

- É importante mencionar ainda que em lugar de positivar H_k poder-se-ia, em princípio, utilizar qualquer outra matriz definida positiva de dimensões apropriadas.
- A razão para optar pelo processo de positivação da hessiana é a analogia com o tradicional método de Levenberg-Marquardt (que será visto a seguir) que pode ser interpretado como uma combinação entre a lei de ajuste do método do gradiente e a lei de ajuste do método de Newton.

Método de Levenberg Marquardt - LM

Definição do Problema

O problema para o qual o algoritmo LM fornece uma solução é chamado Minimização dos Quadrados Minímos Não Linear.

Isto implica que a função a ser minimizada é da forma abaixo

$$f(x) = \frac{1}{2} \sum_{j=1}^m r_j^2(x)$$

onde $x = (x_1, x_2, \dots, x_n)$ é um vetor, e cada r_j é uma função $\mathcal{R}^n \rightarrow \mathcal{R}$. Os r_j são referidos como resíduos e é assumido que $m \geq n$

$f(x)$ pode ser reescrita como $f(x) = \frac{1}{2} \|r(x)\|^2$.

Método de Levenberg Marquardt - LM

O gradiente de $f(x)$ pode ser escrito como:

$$\nabla f(x) = \begin{bmatrix} r_1(x) \frac{\partial r_1(x)}{\partial x_1} + r_2(x) \frac{\partial r_2(x)}{\partial x_1} + \cdots + r_m(x) \frac{\partial r_m(x)}{\partial x_1} \\ r_1(x) \frac{\partial r_1(x)}{\partial x_2} + r_2(x) \frac{\partial r_2(x)}{\partial x_2} + \cdots + r_m(x) \frac{\partial r_m(x)}{\partial x_2} \\ \vdots \\ r_1(x) \frac{\partial r_1(x)}{\partial x_n} + r_2(x) \frac{\partial r_2(x)}{\partial x_n} + \cdots + r_m(x) \frac{\partial r_m(x)}{\partial x_n} \end{bmatrix}$$

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_m(x) \end{bmatrix}$$

$$\nabla r(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \frac{\partial r_1(x)}{\partial x_2} + & \cdots & \frac{\partial r_1(x)}{\partial x_n} \\ \frac{\partial r_2(x)}{\partial x_1} & \frac{\partial r_2(x)}{\partial x_2} + & \cdots & \frac{\partial r_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_m(x)}{\partial x_1} & \frac{\partial r_m(x)}{\partial x_2} + & \cdots & \frac{\partial r_m(x)}{\partial x_n} \end{bmatrix}$$

$$\nabla f(x) = \nabla r(x)^T r$$

Método de Levenberg Marquardt - LM

A hessiana $f(x)$ pode ser escrito como ($m = 2, n = 2$):

$$\nabla f(x) = \begin{bmatrix} r_1(x) \frac{\partial r_1(x)}{\partial x_1} + r_2(x) \frac{\partial r_2(x)}{\partial x_1} \\ r_1(x) \frac{\partial r_1(x)}{\partial x_2} + r_2(x) \frac{\partial r_2(x)}{\partial x_2} \end{bmatrix}$$

$$r(x) = \begin{bmatrix} r_1(x) \\ r_2(x) \end{bmatrix}$$

$$\nabla r(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \nabla r_2(x)^T \end{bmatrix} = \begin{bmatrix} \frac{\partial r_1(x)}{\partial x_1} & \frac{\partial r_1(x)}{\partial x_2} \\ \frac{\partial r_2(x)}{\partial x_1} & \frac{\partial r_2(x)}{\partial x_2} \end{bmatrix}$$

$$\nabla^2 r(x) = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$$

$$h_{11} = \frac{\partial r_1(x)}{\partial x_1} \frac{\partial r_1(x)}{\partial x_1} + r_1(x) \frac{\partial r_1(x)^2}{\partial^2 x_1} + \frac{\partial r_2(x)}{\partial x_1} \frac{\partial r_2(x)}{\partial x_1} + r_2(x) \frac{\partial r_2(x)^2}{\partial^2 x_1}$$

$$h_{12} = \frac{\partial r_1(x)}{\partial x_2} \frac{\partial r_1(x)}{\partial x_1} + r_1(x) \frac{\partial r_1(x)}{\partial x_2} \frac{\partial r_1(x)}{\partial x_1} + \frac{\partial r_2(x)}{\partial x_2} \frac{\partial r_2(x)}{\partial x_1} + r_2(x) \frac{\partial r_2(x)}{\partial x_2} \frac{\partial r_2(x)}{\partial x_1}$$

Método de Levenberg Marquardt - LM

O gradiente de $f(x)$ pode ser escrito como ($m = 2, n = 2$):

$$\nabla f(x) = \nabla r(x)^T r(x)$$

$$H(x) = \nabla r(x)^T \nabla r(x) + \sum_{k=1}^2 r_k(x) \nabla^2 r_k(x)$$

Quando os erros residuais são suficientemente pequenos, a matriz hessiana pode ser aproximada pelo primeiro termo da equação acim

$$H(x) \approx \nabla r(x)^T \nabla r(x)$$

Esta aproximação geralmente é valida em um mínimo de $f(x)$ para a maioria dos própositos, e é a base para o método de Gaus-Newton. A lei de atualização torna-se então:

$$x^{k+1} = x^k - [\nabla r(x)^T \nabla r(x)]^{-1} \nabla r(x)^T r(x)$$

Método de Levenberg Marquardt - LM

A modificação de Levenberg-Marquardt para o método de Gauss-Newton é:

$$x^{k+1} = x^k - [\nabla r(x)^T \nabla r(x) + \mu I]^{-1} \nabla r(x)^T r(x)$$

O efeito da matriz adicional μI é adicionar μ a cada autovalor de $J^T J$

Uma vez que a matriz $J^T J$ é semi-definida positiva e portanto mínimo possível é zero, qualquer valor positivo, pequeno, mas numericamente significativo, de μ será suficiente para restaurar a matriz aumentada e produzir uma direção descendente de busca.

LM: pseudocódigo

Algorithm Levenberg–Marquardt

```
1: Dado  $x_0$ ,  $\lambda_0 > 0$ ,  $\eta > 1$ 
2: for  $k = 0, 1, \dots$  do
3:   Compute  $r(x_k)$  e  $J(x_k)$ 
4:   Resolva  $(J^\top J + \lambda_k I)p_k = -J^\top r$ 
5:    $x_{k+1} = x_k + p_k$ 
6:   if  $\|r(x_{k+1})\| < \|r(x_k)\|$  then                                ▷ melhora
7:      $\lambda_{k+1} = \lambda_k/\eta$ 
8:   else
9:      $\lambda_{k+1} = \eta \lambda_k$ ,       $x_{k+1} = x_k$  (rejeita passo)
10:  end if
11:  if  $\|\nabla f(x_{k+1})\| < \varepsilon$  then
12:    pare
13:  end if
14: end for
```

LM: código Python (ajuste de curva)

```
import numpy as np

def lm(f_res, jac, x0, lam0=1e-2, eta=10.0, maxit=100,
       tol=1e-6):
    x = x0.astype(float)
    lam = lam0
    for k in range(maxit):
        r = f_res(x)
        J = jac(x)
        g = J.T @ r
        if np.linalg.norm(g) < tol:
            break
        A = J.T @ J + lam * np.eye(len(x))
        p = -np.linalg.solve(A, g)
        x_new = x + p
        if np.linalg.norm(f_res(x_new)) < np.linalg.
            norm(r):
            x, lam = x_new, lam/eta
        else:
```

Script Python

```
import numpy as np

def lm(f_res, jac, x0, lam0=1e-2, eta=10.0, maxit=100, tol=1e-10):
    x = x0.astype(float)
    lam = lam0
    for k in range(maxit):
        r = f_res(x)
        J = jac(x)
        g = J.T @ r
        if np.linalg.norm(g) < tol:
            break
        A = J.T @ J + lam * np.eye(len(x))
        p = -np.linalg.solve(A, g)
        x_new = x + p
    return x
```

Script Python

```
if np.linalg.norm(f_res(x_new)) < np.linalg.norm(r):
    x, lam = x_new, lam/eta
else:
    lam = lam*eta
return x
```

Método de Davidson-Fletcher-Powell - DFD

- Este método, assim como o método BFGS, é classificado como método quase-Newton
- A idéia por trás dos métodos quase-Newton é fazer uma aproximação iterativa da inversa da matriz hessiana, de forma que:

$$\lim_{i \rightarrow \infty} M_i = [\nabla^2 f(x)]^{-1}$$

- São considerados os métodos teoricamente mais sofisticados na solução de problemas de otimização não-linear irrestrita e representam o ápice do desenvolvimento de algoritmos através de uma análise detalhada de problemas quadráticos.
- Para problemas quadráticos, gera as direções do método do gradiente conjugado (que será visto posteriormente) ao mesmo tempo que constrói a inversa da Hessiana.
- A cada passo a inversa da Hessiana é aproximada pela soma de duas matrizes simétricas de posto 1, procedimento que é geralmente chamado de correção de posto 2 (rank 2 correction procedure).

Método de Davidson-Fletcher-Powell - DFD

- Construção da inversa

$$M_{i+1} = M_i + \frac{p_i p_i^T}{p_i^T q_i} - \frac{M_i q_i q_i^T M_i}{q_i^T M_i q_i}, \quad i = 0, 1, \dots, n$$

onde $p_i = \alpha_i d_i$, e $q_i = g_{i+1} - g_i = \nabla^2 f(x)p_i$

Note que a avaliação em dois pontos fornece informações sobre a matriz hessiana $\nabla^2 f(x)$

Método de Davidson-Fletcher-Powell - DFD

- 1: Entrada: Atribua um valor inicial para $x \in \mathcal{R}^n$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\epsilon > 0$
- 2: Defina $d_0 = g_0$, $M_0 = I$, faça $i = 0$, $g_0 = -\nabla f(x)$
- 3: **while** condição de parada **do**
- 4: Determine a direção $d_i = M_i g_i$,
- 5: Se $(i \bmod P = 0)$, faça $d_i = g_i$ e $M_i = I$;
- 6: Utilize um procedimento de busca unidimensional para encontrar α_i ;
- 7: Atualize: $x^{k+1} = x^k - \alpha_i d_i$;
- 8: Calcule $p_i = \alpha_i d_i$, g_{i+1} ;
- 9: Faça $q_i = g_{i+1} - g_i$
- 10: Calcule M_{i+1}
- 11: Faça $i = i + 1$
- 12: **end while**

Algorithm 2: Pseudocódigo do DFB

Método de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

$$M_{i+1} = M_i + \frac{p_i p_i^T}{p_i^T q_i} \left[1 + \frac{q_i^T M_i q_i}{p_i^T q_i} \right] - \frac{M_i q_i p_i^T + p_i q_i^T M_i}{p_i^T q_i}, \quad i = 0, 1, \dots, n$$

onde $p_i = \alpha_i d_i$, e $q_i = g_{i+1} - g_i = \nabla^2 f(x) p_i$

Método de Broyden-Fletcher-Goldfarb-Shanno (BFGS)

- 1: Entrada: Atribua um valor inicial para $x \in \mathcal{R}^n$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\epsilon > 0$
- 2: Defina $d_0 = g_0$, $M_0 = I$, faça $i = 0$, $g_0 = -\nabla f(x)$
- 3: **while** condição de parada **do**
- 4: Determine a direção $d_i = M_i g_i$,
- 5: Se ($i \bmod P = 0$), faça $d_i = g_i$ e $M_i = I$;
- 6: Utilize um procedimento de busca unidimensional para encontrar α_i ;
- 7: Atualize: $x^{k+1} = x^k - \alpha_i d_i$;
- 8: Calcule $p_i = \alpha_i d_i$, g_{i+1} ;
- 9: Faça $q_i = g_{i+1} - g_i$
- 10: Calcule M_{i+1}
- 11: Faça $i = i + 1$
- 12: **end while**

Algorithm 3: Pseudocódigo do BFGS

BFGS: Código Python

```
def bfgs(f, g, x0, maxit=200, tol=1e-6):
    n = x0.size
    H = np.eye(n)
    x = x0.astype(float)
    for k in range(maxit):
        grad = g(x)
        if np.linalg.norm(grad) < tol: break
        p = -H @ grad
        alpha = line_search(f, g, x, p)
        s = alpha * p
        x_new = x + s
        y = g(x_new) - grad
        ys = y.dot(s)
```

BFGS: Python (Rosenbrock)

```
if ys <= 1e-12:  
    x = x_new; continue  
I = np.eye(n)  
V = I - np.outer(s, y)/ys  
H = V @ H @ V.T + np.outer(s, s)/ys  
x = x_new  
return x
```

Método Secante de um Passo

- O termo método de secante provém do fato de que as derivadas são aproximadas por secantes avaliadas em dois pontos da função (neste caso a função é o gradiente).
- Uma vantagem deste método é que sua complexidade é de ordem $O(n)$, ou seja, é linear em relação ao número n de parâmetros, enquanto a complexidade dos métodos DFP e BFGS é de ordem $O(n^2)$
- A principal razão da redução do esforço computacional deste método em relação aos anteriores (DFP e BFGS), é que agora a direção de atualização é calculada somente a partir de vetores determinados pelos gradientes, e não há mais a armazenagem da aproximação da inversa da hessiana

Método Secante de um Passo

- A nova direção d_{i+1} é obtida como segue:

$$d_{i+1} = -g_i + A_i s_i + B_i q_i$$

onde $s_i = x^{i+1} - x^i = p_i$, $q_i = g_{i+1} - g_i$, $p_i = \alpha_i d_i$

$$A_i = - \left[1 + \frac{q_i^T q_i}{s_i^T q_i} \right] \frac{s_i^T g_i}{s_i^T q_i} + \frac{q_i^T g_i}{s_i^T q_i}$$

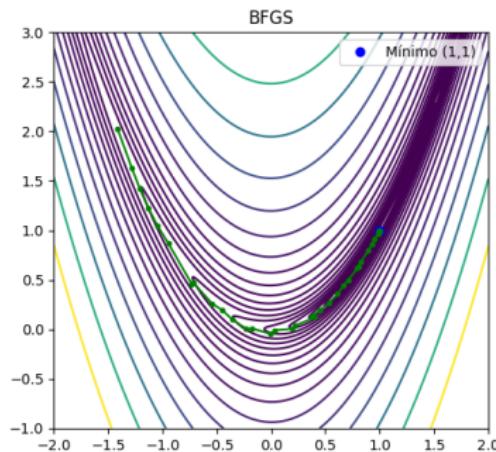
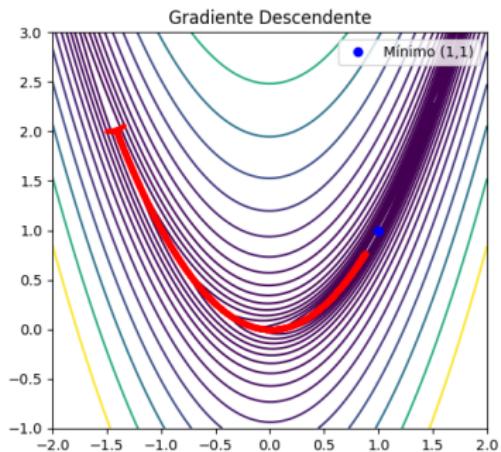
$$B_i = \frac{s_i^T g_i}{s_i^T q_i}$$

Método Secante de um Passo

- 1: Entrada: Atribua um valor inicial para $x \in \mathcal{R}^n$ para o vetor de parâmetros e um valor arbitrariamente pequeno para uma constante $\epsilon > 0$
- 2: Defina $d_0 = g_0, p_0 = \alpha_0 d_0$, faça $i = 0, g_0 = -\nabla f(x)$
- 3: **while** condição de parada **do**
- 4: Determine a direção $d_i = -g_i + A_i s_i + B_i q_i$,
- 5: Se $(i \bmod P = 0)$, faça $d_i = g_i$;
- 6: Utilize um procedimento de busca unidimensional para encontrar α_i ;
- 7: Atualize: $x^{k+1} = x^k - \alpha_i d_i$;
- 8: Calcule $p_i = \alpha_i d_i, g_{i+1}$;
- 9: Faça $q_i = g_{i+1} - g_i$
- 10: Faça $i = i + 1$
- 11: **end while**

Algorithm 4: Pseudocódigo do Método das Secantes

BBGS versus Gradiente



Gradiente Conjugado: Polak-Ribiere (PR) e Fletcher-Reeves (FR)

- Existe um consenso geral da comunidade de análise numérica que a classe de métodos de otimização chamados métodos do gradiente conjugado, tratam de problemas de grande escala de maneira efetiva.
- Os métodos do gradiente conjugado possuem sua estratégia baseada no modelo geral de otimização apresentado no algoritmo padrão e do gradiente, mas escolhem a direção de busca d_i , o passo α_i e o coeficiente de momento β_i mais eficientemente utilizando informações de segunda ordem.
- É projetado para exigir menos cálculos que o método de Newton e apresentar taxas de convergência maiores que as do método do gradiente.
- É baseado no método das direções conjugadas proposto para tratar problemas quadráticos:

$$f(x) = \frac{1}{2}x^T Qx - b^T x$$

- A lei de ajuste do método das direções conjugadas é dada por: $x^{k+1} = x^k - \alpha_k d_k$,
$$\alpha_k = \frac{d_k^T \nabla f(x)d_k}{d_k^T Q d_k}$$

Gradiente Conjugado: Polak-Ribiere (PR) e Fletcher-Reeves (FR)

- Antes de aplicarmos a lei de ajuste dada pela equação acima, é necessário obter as direções Q -conjugadas $d_i \in \mathbb{R}^n$, $i = 0, \dots, n - 1$. Uma maneira de determinar estas direções é tomá-las na forma

$$\begin{cases} d_0 = -\nabla f(x^0) \\ d_{i+1} = -\nabla f(x^{i+1}) + \beta_i d_i & i \geq 0 \end{cases}$$

$$\text{com } \beta_i = \frac{\nabla f(x_{i+1})^T Q d_i}{d_i^T Q d_i}$$

- Para problemas não-quadráticos a matriz Q deve ser aproximada pela matriz hessiana calculada no ponto x^i

Gradiente Conjugado: Polak-Ribiere (PR) e Fletcher-Reeves (FR)

- A aplicação destes algoritmos a problemas não quadráticos envolve um procedimento de busca unidimensional do passo de ajuste (taxa de aprendizagem) e a aproximação do parâmetro β utilizando informações de primeira ordem (gradientes).

Uma destas aproximações é dada pelo método de Polak-Ribière (PR)

Se ($i \bmod n \neq 0$), faça $d_{i+1} = g_i + \beta_i d_i$, onde $\beta_i = \frac{g_{i+1}^T(g_{i+1} - g_i)}{g_i^T g_i}$

Senão, faça: $d_i = g_i$

Outra aproximação é dada pelo método de Fletcher-Reeves (FR)

Se ($i \bmod n \neq 0$), faça $d_{i+1} = g_i + \beta_i d_i$, onde $\beta_i = \frac{\|g_{i+1}\|^2}{\|g_i\|^2}$

Senão, faça: $d_i = g_i$

Gradiente Conjugado: Polak-Ribiere (PR)

- 1: Atribua um valor incial $x^0 \in \mathcal{R}^n$ para o vetor de parâmetros e um valor arbitrariamente pequeno para a constante $\epsilon > 0$
- 2: Calcule $\nabla f(x^0)$, faça $d_0 = \nabla f(x^0)$ e $i = 0$
- 3: **while** Condição de parada não for satisfeita **do**
- 4: Utilize um procedimento de busca unidimensional para encontrar um α_i ; que seja solução ótima do problema
- 5: Faça $x^{i+1} = x^i + \alpha_i d_i$
- 6: Calcule $g_i = -\nabla f(x_i)$
- 7: Se $(i \bmod n \neq 0)$, faça $d_{i+1} = g_i + \beta_i d_i$, onde $\beta_i = \frac{g_{i+1}^T(g_{i+1}-g_i)}{g_i^T g_i}$
- 8: Senão, faça: $d_i = g_i$
- 9: *i* = *i* + 1
- 10: **end while**

Algorithm 5: Algoritmo Polak-Ribi  re

Gradiente Conjugado: Fletcher-Reeves (FR)

- 1: Atribua um valor incial $x^0 \in \mathbb{R}^n$ para o vetor de parâmetros e um valor arbitrariamente pequeno para a constante $\epsilon > 0$
- 2: Calcule $\nabla f(x^0)$, faça $d_0 = \nabla f(x^0)$ e $i = 0$
- 3: **while** Condição de parada não for satisfeita **do**
- 4: Utilize um procedimento de busca unidimensional para encontrar um *alpha*; que seja solução ótima do problema
- 5: Faça $x^{i+1} = x^i + \alpha_i d_i$
- 6: Calcule $g_i = -\nabla f(x_i)$
- 7: Se $(i \bmod n \neq 0)$, faça $d_{i+1} = g_i + \beta_i d_i$, onde $\beta_i = \frac{\|g_{i+1}\|^2}{\|g_i\|^2}$
- 8: Senão, faça: $d_i = g_i$
- 9: $i = i + 1$
- 10: **end while**

Algorithm 6: Algoritmo Polak-Ribière

Gradiente Conjugado: Exemplo

Usar o CG para minimizar $f(x) = 0.5x_1^2 + 2.5x_2^2$

Gradiente é igual a $\nabla f(x) = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix}$

Escolhendo $x^0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$ obtemos a direção de procura inicial (correspondente ao gradiente negativo)

$$d_0 = -g_0 = -\nabla f(x_0) = \begin{bmatrix} -5 \\ -5 \end{bmatrix}$$

O mínimo exato ao longo da linha de procura é $\alpha_0 = 1/3$, pelo que a próxima solução aproximada é $x_1 = \begin{bmatrix} 3.333 \\ -0.666 \end{bmatrix}$

Calculando o novo gradiente $g_1 = -\nabla f(x_1) = \begin{bmatrix} 3.333 \\ -3.333 \end{bmatrix}$

Gradiente Conjugado: Exemplo

Neste ponto, em vez de procurar a direção do gradiente negativo

$$\beta_1 = \frac{g_1^T g_1}{g_0^T g_0} = 0.444$$

que origina a nova direção de procura

$$d_1 = g_1 + \beta_1 d_0 = \begin{bmatrix} 3.333 \\ -3.333 \end{bmatrix} + 0.444 \begin{bmatrix} -5 \\ -5 \end{bmatrix} = \begin{bmatrix} 5.556 \\ 1.111 \end{bmatrix}$$

O passo ao longo desta linha que minimiza a função é $\alpha_1 = 0.6$, que origina a solução exata $x_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, tal como seria de esperar para um função quadrática.

Cálculo Numérico

Diferenças finitas - Revisão

Prof. Dr. Clodoaldo A M Lima

Escola de Artes Ciências e Humanidades / Sistema de Informação

26 de agosto de 2025

Ideia de Diferenças Finitas

- Para funções reais de variável real:

$$f'(x) \approx \frac{f(x + h) - f(x - h)}{2h}$$

(diferença central).

- Para funções de várias variáveis:

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + he_i) - f(x - he_i)}{2h}$$

onde e_i é o vetor canônico.

- A precisão depende de h : muito grande gera erro de truncamento, muito pequeno gera erro numérico.

Gradiente via Diferenças Finitas

Definição

O gradiente de $f : \mathbb{R}^n \rightarrow \mathbb{R}$ é:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right]^T$$

Aproximação Numérica

$$\frac{\partial f}{\partial x_i}(x) \approx \frac{f(x + he_i) - f(x - he_i)}{2h}$$

Hessiana via Diferenças Finitas

Definição

A Hessiana de $f : \mathbb{R}^n \rightarrow \mathbb{R}$ é a matriz:

$$H_{ij}(x) = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

Aproximação Numérica

- Diagonal:

$$\frac{\partial^2 f}{\partial x_i^2}(x) \approx \frac{f(x + he_i) - 2f(x) + f(x - he_i)}{h^2}$$

- Fora da diagonal:

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) \approx \frac{f(x + he_i + he_j) - f(x + he_i - he_j) - f(x - he_i + he_j) + ...}{4h^2}$$

$$\frac{f(x - he_i - he_j)}{4h^2}$$

Exemplo Numérico

Função escolhida:

$$f(x, y) = x^3 e^y + \sin(xy)$$

No ponto $(1, 0.5)$ e $h = 10^{-5}$:

- Gradiente analítico:

$$\nabla f(1, 0.5) = (5.277, 2.509)$$

- Hessiana analítica:

$$H = \begin{bmatrix} 9.656 & 5.016 \\ 5.016 & 1.342 \end{bmatrix}$$

- Diferenças finitas produzem valores quase idênticos.

Script Python

```
import numpy as np

def f(x):
    return x[0]**3 * np.exp(x[1]) + np.sin(x[0]*x[1])

def gradiente(f, x, h=1e-5):
    n = len(x); g = np.zeros(n)
    for i in range(n):
        e = np.zeros(n); e[i] = 1
        g[i] = (f(x+h*e) - f(x-h*e))/(2*h)
    return g

def hessiana(f, x, h=1e-5):
    n = len(x); H = np.zeros((n,n)); fx = f(x)
    for i in range(n):
        e = np.zeros(n); e[i] = 1
        H[i, i] = (f(x+h*e) - 2*fx + f(x-h*e))/(h**2)
```

Exercícios Propostos

Use diferenças finitas para aproximar gradiente e Hessiana das funções abaixo. Compare com os valores analíticos.

- ① $f(x, y) = x^2 + y^2$ no ponto $(1, 2)$.
- ② $f(x, y) = \ln(1 + x^2 + y^2)$ no ponto $(0.5, -0.5)$.
- ③ $f(x, y, z) = xe^y + y \cos(z)$ no ponto $(1, 0, \pi/4)$.
- ④ $f(x, y) = \sin(x) \cos(y)$ no ponto $(\pi/4, \pi/6)$.
- ⑤ $f(x, y) = x^4 + y^4 - 3xy$ no ponto $(1, 1)$.

Tarefas:

- Calcule o gradiente e a Hessiana analiticamente.
- Estime com diferenças finitas.
- Compare os resultados e discuta a influência de h .

Pseudocódigo - Método de Newton (Relembrando)

- Escolher x_0 , tolerância ϵ
- **Repetir:**
 - Calcular $\nabla f(x_k)$ e $H(x_k)$
 - Resolver $H(x_k)d_k = -\nabla f(x_k)$
 - Atualizar: $x_{k+1} = x_k + d_k$
- Parar se $\|\nabla f(x_k)\| < \epsilon$

Python - Método de Newton

```
def newton_method(f, grad, hess, x0, tol=1e-6):
    x = x0
    for _ in range(10):
        g = grad(x)
        H = hess(x)
        dx = np.linalg.solve(H, -g)
        x = x + dx
        if np.linalg.norm(g) < tol:
            break
    return x
```

Método de Newton com Positivação da Hessiana

Diferenças finitas - Revisão

Escola de Artes Ciências e Humanidades / Sistema de Informação

Objetivo

Implementar o método de Newton para otimização de funções:

- Aproximação numérica do gradiente e Hessiana.
- Regularização para garantir Hessiana definida positiva.
- Uso de backtracking line search para garantir convergência estável.

Código - Parte 1: Gradiente e Hessiana

```
1 import numpy as np
2
3 # Gradiente aproximado usando diferenças centrais
4 def approx_grad(f, x, h=1e-6):
5     x = np.asarray(x, dtype=float)
6     n = x.size
7     g = np.zeros(n, dtype=float)
8     for i in range(n):
9         xp = x.copy(); xm = x.copy()
10        xp[i] += h
11        xm[i] -= h
12        g[i] = (f(xp) - f(xm)) / (2*h)
13    return g
14
15 # Hessiana aproximada usando diferenças finitas
16 # centrais
17 def approx_hess(f, x, h=1e-4):
18     x = np.asarray(x, dtype=float)
19     n = x.size
```

Código - Parte 2: Hessiana (continuação)

```
for i in range(n):
    xi = x.copy()
    xi[i] += h
    f_ip = f(xi)
    xi[i] -= 2*h
    f_im = f(xi)
    # Segunda derivada parcial em relação a xi
    H[i, i] = (f_ip - 2*fx + f_im) / (h**2)

for j in range(i+1, n):
    # Segunda derivada mista
    xijp = x.copy(); xijm = x.copy()
    ximjp = x.copy(); imj = x.copy()
    xijp[i] += h; xijp[j] += h
    xijm[i] += h; xijm[j] -= h
    ximjp[i] -= h; ximjp[j] += h
    imj[i] -= h; imj[j] -= h
    H[i, j] = (f(xijp) - f(xijm) - f(ximjp) +
                f(imj)) / (4*h*h)
```

Código - Parte 3: Positivação da Hessiana

```
# Garante que a Hessiana seja definida positiva
def make_pos_def(H, tau=1e-8, max_tries=20):
    H = np.array(H, dtype=float)
    H = 0.5 * (H + H.T) # Força simetria
    try:
        eigs = np.linalg.eigvalsh(H)
    except np.linalg.LinAlgError:
        eigs = np.linalg.eigvals(H).real
    min_eig = np.min(eigs)

    if min_eig > 0:
        return H, 0.0

    # Adiciona lambda*I até ficar positiva
    lam = (-min_eig) + max(tau, 1e-8)
    for _ in range(max_tries):
        H_reg = H + lam * np.eye(H.shape[0])
        try:
            eigs = np.linalg.eigvalsh(H_reg)
```

Código - Parte 4: Backtracking e Método de Newton

```
# Busca adaptativa de passo (backtracking)
def backtracking_line_search(f, x, p, grad, alpha0
    =1.0, rho=0.5, c=1e-4):
    alpha = alpha0
    fx = f(x)
    gTp = np.dot(grad, p)
    while alpha > 1e-12:
        if f(x + alpha*p) <= fx + c * alpha * gTp:
            return alpha
        alpha *= rho
    return alpha

# Método de Newton
def newton_opt(f, x0, tol=1e-6, max_iter=50,
                fd_grad_h=1e-6, fd_hess_h=1e-4,
                use_backtracking=True, verbose=True):
    x = np.asarray(x0, dtype=float)
    for k in range(1, max_iter+1):
        fx = f(x)
```

Código - Parte 5: Continuação do Método de Newton

```
1      H = approx_hess(f, x, h=fd_hess_h)
2      Hpos, lam = make_pos_def(H)
3
4      try:
5          p = -np.linalg.solve(Hpos, g)
6      except np.linalg.LinAlgError:
7          p, *_ = np.linalg.lstsq(Hpos, -g, rcond=None)
8
9      alpha = 1.0
10     if use_backtracking:
11         alpha = backtracking_line_search(f, x, p,
12                                         g, alpha0=1.0)
13
14     x = x + alpha * p
15
16     return x
```

Código - Parte 6: Exemplo de Uso

```
1 if __name__ == "__main__":
2     def f_ex(x):
3         # Função exemplo: f(x,y) = x^2 + x*y + y^2 +
4         # sin(x)
5         return x[0]**2 + x[0]*x[1] + x[1]**2 + np.sin(
6             x[0])
7
8
9     x0 = np.array([1.0, 1.0])
10    x_opt = newton_opt(f_ex, x0, tol=1e-8, max_iter
11        =50, verbose=True)
12
13    print("\nPonto ótimo aproximado:", x_opt)
14    print("Valor mínimo f:", f_ex(x_opt))
```