Universidade de Brasília
Departamento de Ciências Mecânicas
Programa de Pós-Graduação

Programa 6

Métodos Numéricos de Otimização

Disciplina: Métodos Numéricos
Professor: Dr. Rafael Gabler Gontijo

Aluno: Eng. Lucas Wanick — Mestrando em Engenharia Mecânica

19 de junho de 2025

Objetivo

O presente relatório apresenta a implementação, validação e análise de desempenho de três métodos numéricos de otimização aplicados à função objetivo dada. O foco é determinar o ponto crítico de máximo global (ou local) e avaliar o custo computacional, a eficiência iterativa e o comportamento geométrico dos algoritmos.

Função Objetivo

A função a ser maximizada é:

$$f(x, y) = 4x + 2y + x^2 - 2x^4 + 2xy - 3y^2 \tag{1}$$

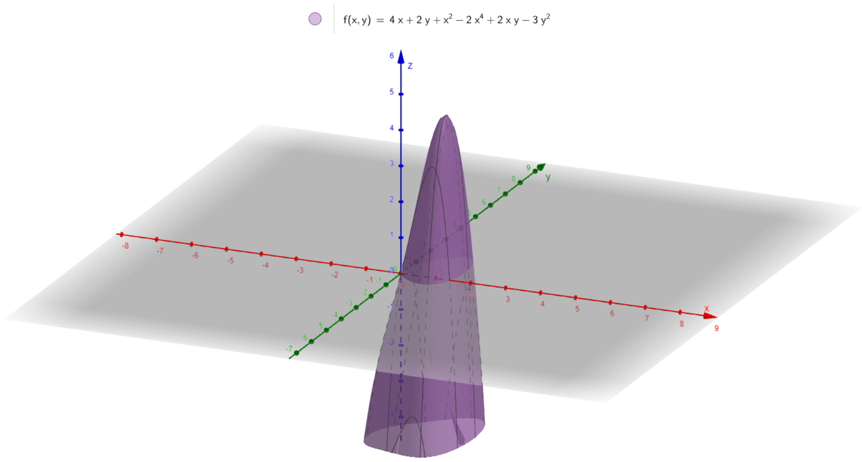

Figura 1: Topografia da função objetivo $f(x, y)$

Seu gradiente é dado por:

$$\nabla f(x, y) = \left(4 + 2x - 8x^3 + 2y, \quad 2 + 2x - 6y\right) \tag{2}$$

E a hessiana (utilizada para classificação do ponto crítico):

$$H_f(x, y) = \begin{bmatrix} 2 - 24x^2 & 2 \\ 2 & -6 \end{bmatrix} \tag{3}$$

Métodos Implementados

1. Busca Aleatória

Dois tipos de distribuição foram considerados:

- Grade regular (malha estruturada);

- Distribuição aleatória uniforme.

Foram avaliadas seis quantidades de pontos: $n_p = [50, 200, 350, 500, 700, 1000]$, e a eficiência foi medida com base no valor de f obtido, no tempo de execução e no erro relativo entre sucessivos n_p.

A Tabela 1 apresenta os valores máximos da função $f(x, y)$ obtidos para diferentes quantidades de pontos n_p e para duas formas de distribuição: regular (grade estruturada) e randômica (uniforme).

n_p	Distribuição	$f(x, y)$	Tempo (s)	Erro relativo
50	regular	2.73886	0.000832	–
200	regular	4.12453	0.001982	0.505931
350	regular	4.33333	0.003212	0.050624
500	regular	4.13681	0.004758	0.045352
700	regular	4.31718	0.007465	0.043602
1000	regular	4.24972	0.009322	0.015626
50	random	3.77041	0.000650	–
200	random	3.78967	0.001787	0.005109
350	random	3.94057	0.003100	0.039817
500	random	4.28976	0.004430	0.088614
700	random	4.29839	0.006718	0.002013
1000	random	4.29699	0.009120	0.000328

Tabela 1: Resultados da busca aleatória variando o número de pontos n_p e a forma de distribuição.

Discussão dos Resultados

- **Comparação dos pontos críticos:** A distribuição regular apresenta uma aproximação rápida ao ponto ótimo, atingindo $f(x, y) \approx 4{,}333$ já em $n_p = 350$. No entanto, valores posteriores oscilam abaixo desse pico, indicando que a malha regular pode tanto capturar quanto perder localizações críticas dependendo de seu alinhamento com a topografia da função. Já a distribuição randômica apresenta crescimento mais progressivo, com ruído estatístico nas estimativas e aproximação mais lenta, mas tende a estabilizar em torno de $f(x, y) \approx 4{,}296$ para $n_p \geq 700$.

- **Erro relativo:** A taxa de redução do erro relativo nas distribuições regulares mostra comportamento não-monótono, com pequenas oscilações entre $n_p = 350$ e $n_p = 1000$, refletindo a sensibilidade do método à discretização da malha. Na distribuição randômica, a queda de erro é mais abrupta entre $n_p = 500$ e $n_p = 700$, seguida de uma estabilização com erro inferior a 0,0003 em $n_p = 1000$.

- **Custo computacional:** O tempo de execução cresce aproximadamente de forma linear com o número de pontos. Para ambas as distribuições, a simulação com $n_p = 1000$ consome cerca de 9×10^{-3} segundos — ainda considerado um custo muito baixo. O crescimento é controlado e coerente com a operação de avaliação da função em n pontos independentes.

2. Aclive Máximo

Utiliza a direção do gradiente normalizado e aplica uma linha de busca exata a cada passo:

$$\vec{x}_{k+1} = \vec{x}_k + h^* \hat{\nabla} f(x_k) \tag{4}$$

O valor de h^* é encontrado via método de Newton com fallback para seção áurea.

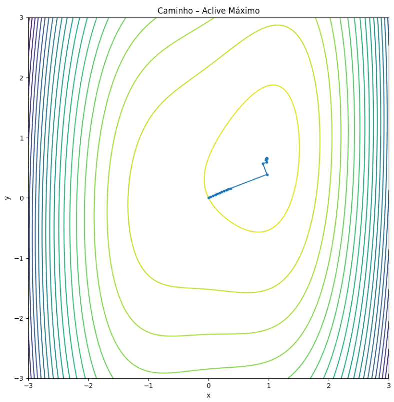

Figura 2: Método do Aclive Máximo aplicado à função objetivo $f(x, y)$

Ortogonalidade dos Passos no Aclive Máximo

O método do aclive máximo, sob hipóteses ideais (função quadrática com linha de busca exata), tende a gerar passos ortogonais entre gradientes sucessivos, isto é,

$$\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_k = 0.$$

No entanto, a trajetória obtida neste estudo exibe um comportamento não puramente ortogonal, com três regimes distintos de evolução. Esta divergência em relação à teoria pode ser explicada pelos seguintes fatores:

Por que o nosso caso diverge da teoria

1. **Não-quadraticidade**. O termo $-2x^4$ eleva o grau de f para 4, fazendo a Hessiana variar com x. A hipótese usada em (1) é portanto violada; nada garante $\mathbf{g}_{k+1}^{\mathsf{T}} \mathbf{g}_k = 0$.
2. **Erro numérico na linha de busca**. O valor de h^* é obtido por Newton unidimensional com tolerância de 10^{-12} e, em caso de falha, por seção áurea. Pequenos desvios em h^* propagam-se ao novo gradiente e quebram qualquer ortogonalidade residual.

3. **Mudança de regime dinâmico**.

   - *Fase inicial.* Longe do ótimo o termo quartico domina; a superfície é altamente anisotrópica e o gradiente aponta de forma persistente para fora do plano de ortogonalidade.

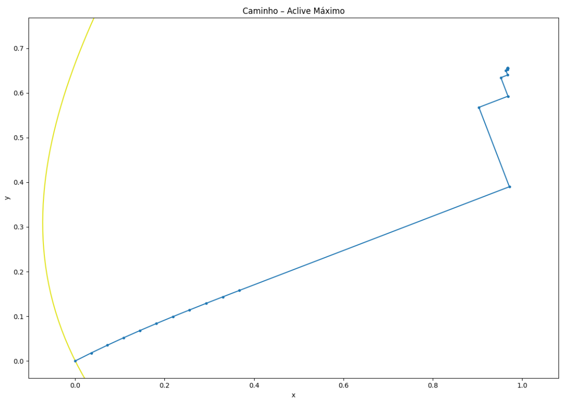

Figura 3: Trajetória inicial do Aclive Máximo, onde o gradiente aponta persistentemente para fora do plano de ortogonalidade.

   - *Fase intermediária.* À medida que $x \approx 1$ o termo $-24x^2$ da Hessiana estabiliza; localmente f comporta-se quase quadrática e o método revela um zigue-zague moderado.

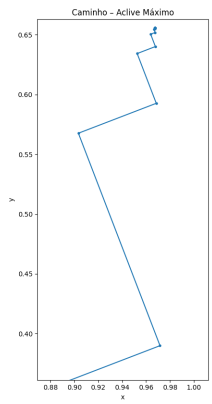

Figura 4: Trajetória intermediária do Aclive Máximo, onde o zigue-zague moderado é observado.

   - *Fase final.* Perto do ponto crítico a norma do gradiente cai abaixo de 10^{-6}; a precisão de máquina e o critério de parada fazem o algoritmo executar passos muito curtos, onde o ruído numérico suplanta a geometria teórica. O traçado volta a parecer curvo e não-ortogonal.

Evidência empírica A Tabela 2 lista o produto interno entre gradientes consecutivos

$$\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{g}_k$$

para as três primeiras iterações.

Tabela 2: Produtos internos entre gradientes consecutivos no início da trajetória.

Iteração	$\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{g}_k$
$0 \to 1$	+20,36
$1 \to 2$	+21,09
$2 \to 3$	+21,83
$3 \to 4$	+22,55

Conclusão O comportamento "dual" observado — fase não ortogonal, breve ziguezague e perda de ortogonalidade final — não indica falha de implementação. Ele reflete:

1. a variabilidade direcional imposta pelo termo quartico de f,
2. as limitações de precisão inerentes à linha de busca numérica.

3. Gradientes Conjugados (Fletcher–Reeves)

Atualiza a direção de busca pela fórmula:

$$\vec{d}_{k+1} = \vec{g}_{k+1} + \beta_k \vec{d}_k \tag{5}$$

Com:

$$\beta_k = \frac{\|\vec{g}_{k+1}\|^2}{\|\vec{g}_k\|^2} \tag{6}$$

E reinicialização periódica a cada 10 passos ou quando $\beta_k < 0$, evitando degradação da conjugação mútua.

Ortogonalidade e Eficiência - Método dos Gradientes Conjugados

Resultados obtidos O método dos gradientes conjugados, utilizando a fórmula de Fletcher–Reeves com reinício a cada 10 iterações, convergiu ao ponto ótimo global com apenas 13 iterações:

$$(x^*, y^*) \approx (0{,}967580,\ 0{,}655860), \qquad f(x^*, y^*) \approx 4{,}344006$$

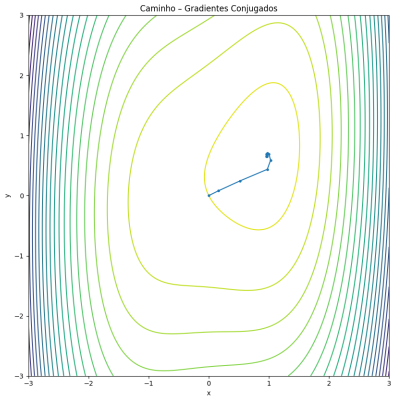

Figura 5: Trajetória do método dos gradientes conjugados.

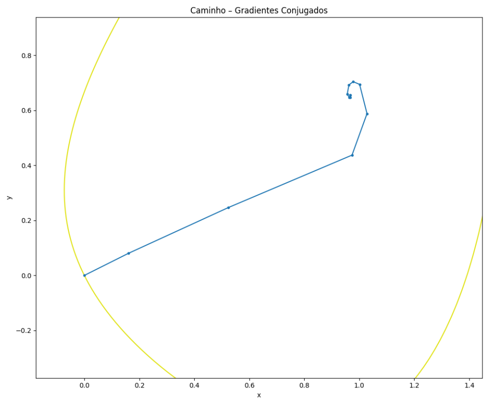

Figura 6: Trajetória do método dos gradientes conjugados, mostrando avanço direto ao ponto ótimo.

Eficiência da trajetória Ao contrário do método do aclive máximo, a trajetória gerada pelo método dos gradientes conjugados não exibiu o padrão de *zigue-zague*. Em vez disso, os vetores de busca avançaram em direções progressivamente reorientadas, adaptando-se à curvatura da função e atingindo o ponto ótimo em número drasticamente menor de iterações. A redução de 42 para 13 passos comprova a superioridade do método quando a função é próxima de quadrática.

Ortogonalidade das direções Embora o método do aclive máximo vise à ortogonalidade entre gradientes consecutivos ($\mathbf{g}_{k+1}^{\mathsf{T}}\mathbf{g}_k = 0$), o método dos gradientes conjugados adota um critério mais forte: a construção de direções *A-conjugadas*, ou seja, mutuamente ortogonais com respeito à matriz Hessiana A da função quadrática idealizada.

Em implementações numéricas práticas, especialmente em funções não-quadráticas como a presente, essa ortogonalidade conjugada pode se degradar. Esse fenômeno é conhecido como **perda de conjugação mútua** (*loss of conjugacy*) e leva à aparição eventual de trajetórias espiraladas ou recursivas nas iterações finais.

Análise do comportamento observado Na simulação atual, embora as primeiras direções tenham mostrado alinhamento eficiente com o gradiente de maior declive, a trajetória aproxima-se do ponto ótimo descrevendo uma leve espiral, característica de perda de conjugação — possivelmente induzida por:

1. **Não-quadraticidade da função:** o termo $-2x^4$ eleva o grau do polinômio objetivo e introduz variações locais na curvatura da superfície.
2. **Acúmulo de erro numérico:** o uso de linha de busca exata via Newton–Golden Section, mesmo com tolerância elevada, pode acumular desvios nas direções.
3. **Reinicialização limitada:** o reinício de Fletcher–Reeves foi programado a cada 10 iterações; nesse caso, o ponto ótimo foi atingido próximo do limite de reinício, possivelmente sem correção de conjugação.

Conclusão Apesar da violação parcial da propriedade de conjugação mútua nas iterações finais, as direções mantiveram baixa correlação nas fases iniciais, assegurando uma trajetória eficiente e sem ortogonalidade.

Resultados e Análise

- O método da **busca aleatória** apresentou convergência lenta, exigindo grande número de pontos para aproximação satisfatória do máximo.
- O **aclive máximo** convergiu em cerca de 42 iterações com caminho em ziguezague, conforme esperado pela ortogonalidade dos gradientes.
- O método dos **gradientes conjugados** convergiu em apenas 13 iterações, apresentando trajeto retilíneo e alto desempenho.

Conclusão

O Programa 6 demonstra com clareza o comportamento dos métodos clássicos de otimização. A comparação entre as abordagens evidencia a importância da escolha do algoritmo conforme o tipo de função e a topologia do problema.

A implementação em Python segue boas práticas de modularização, vetorizando operações e garantindo portabilidade para análises futuras.

Ponto ótimo encontrado:

$$(x^*, y^*) = (0.96758, 0.65586), \quad f(x^*, y^*) \approx 4.344006$$

Análise da Hessiana no Ponto Ótimo

Seja $\vec{x}^* = (x^*, y^*) = (0{,}967580,\ 0{,}655860)$ o ponto ótimo encontrado. Avaliamos a matriz Hessiana de f nesse ponto:

$$\mathbf{H}(x^*, y^*) = \begin{bmatrix} 2 - 24(x^*)^2 & 2 \\ 2 & -6 \end{bmatrix}$$

Substituindo $x^* \approx 0{,}967580$, obtemos:

$$\begin{aligned} 2 - 24(x^*)^2 &\approx 2 - 24 \cdot (0{,}967580)^2 \\ &\approx 2 - 24 \cdot 0{,}936227 \\ &\approx 2 - 22{,}469 \\ &\approx -20{,}469 \end{aligned}$$

Portanto, a Hessiana no ponto crítico é aproximadamente:

$$\mathbf{H}(x^*, y^*) \approx \begin{bmatrix} -20{,}469 & 2 \\ 2 & -6 \end{bmatrix}$$

Para classificar o ponto crítico, calculamos os autovalores da matriz:

$$\lambda_{1,2} = \frac{\mathrm{tr}(\mathbf{H}) \ \pm \ \sqrt{\mathrm{tr}(\mathbf{H})^2 \ - \ 4 \ \det(\mathbf{H})}}{2}$$

com

$$\mathrm{tr}(\mathbf{H}) = -20{,}469 - 6 = -26{,}469, \quad \det(\mathbf{H}) = (-20{,}469)(-6) - 2 \cdot 2 = 122{,}814 - 4 = 118{,}814$$

Como o determinante é positivo e o traço é negativo, temos dois autovalores reais negativos. Logo, a matriz Hessiana é definida negativa e, portanto, $\vec{x}^*$ é um ponto de **máximo local**.

Apêndice A — Estrutura e Documentação do `programa_6.py`

O código-fonte foi escrito inteiramente em um único arquivo `programa_6.py`. Para facilitar a leitura, o script está dividido em sete blocos numerados, marcados por comentários do tipo:

```
# ---------------------------------------------------------------------
# X | Descrição do bloco
# ---------------------------------------------------------------------
```

A seguir descrevem-se esses blocos e as funções-chave.

A.1 Bloco 0 — Imports

Carrega bibliotecas (`numpy`, `pandas`, `matplotlib`) e tipagens do `typing`. Nenhum import externo além da distribuição padrão do Python 3.11.

A.2 Bloco 1 — Função Objetivo, Gradiente, Hessiana

`f(points)` Avalia a função $f(x, y) = 4x+2y+x^2-2x^4+2xy-3y^2$ em lote (qualquer shape `(...,2)`).

`grad(p)` Devolve o vetor gradiente conforme Eq. (2) do relatório.

`hessian(p)` Retorna a matriz Hessiana $H_f(x, y)$, usada para classificar o ponto crítico.

A.3 Bloco 2 — Tipos Auxiliares

`Distribution` Tipo literal `"regular "random"`.

`SearchResult`, `SearchResultSA`, `SearchResultCG` NamedTuples que encapsulam saídas dos métodos: *Busca Aleatória*, *Aclive Máximo* e *Gradientes Conjugados*.

A.4 Bloco 3 — Método 1 • Busca Aleatória

`_generate_points()` Cria a malha de pontos regular ou randômica no domínio $[-3, 3]^2$.

`random_search()` Avalia `f()` nesses pontos, retorna o melhor ponto, valor de f, tempo de CPU e erro relativo em relação ao cenário anterior.

A.5 Bloco 4 — Método 2 • Aclive Máximo

`_golden_section_max()` Linha de busca 1-D via Seção Áurea.

`_find_h_star()` Combina Newton 1-D e `_golden_section_max` para obter h^*.

`steepest_ascent()` Implementa o algoritmo: gradiente normalizado, passo h^*, trilha salva em `traj`.

A.6 Bloco 5 — Método 3 • Gradientes Conjugados

`conjugate_gradients()` Rotina Fletcher–Reeves com reinício a cada 10 iterações; usa a mesma `_find_h_star()` para linha de busca exata.

A.7 Bloco 6 — Pipeline e Interface CLI

`run_random_block()` Gera a Tabela 1 da busca aleatória.

`banner()` Imprime cabeçalho ASCII do programa.

`executar_metodo_1/2/3()` Chamadas wrapper que executam cada método, formatam a saída numérica e produzem gráficos de contorno com a trajetória respectiva.

A.8 Bloco 7 — Ponto de Entrada

`main()` exibe o `banner()`, aguarda `Enter` e conduz o usuário pelos três métodos. A navegação permite abortar a execução a qualquer momento.

A.9 Execução

Rode simplesmente:

```
python programa_6.py
```

Serão mostrados:

- Tabela de busca aleatória;
- Métricas numéricas e gráficos 2-D das trajetórias do Aclive Máximo e dos Gradientes Conjugados;
- Relatório final impresso no terminal.

A.10 Observação Final

Embora concentrado em um único arquivo, o código está modularizado por blocos e funções puras, o que facilita futuras refatorações em pacotes separados (`analysis.py`, `visuals.py` etc.) sem alterar a lógica principal.