PUC-RJ Pontifícia Universidade Católica do Rio de Janeiro
MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica
Trabalho 01 - Otimização sem Restrições

Professor: Ivan Menezes

Felipe da Costa Pereira - mat. 2212376
felipecostapereira@gmail.com

9 de outubro de 2022

1 Introdução

Otimização sem restrição (OSR) consiste em encontrar o mínimo de uma função $f(\vec{x})$ onde não há restrição em relação ao domínio das variáveis $\vec{x}$. A fim de se encontar o minimo da função $f(\vec{x})$ a partir de um ponto de partida ($\vec{x_0}$) , os métodos apresentados nesse trabalho consistem na repetição de duas etapas principais até que um critério de parada seja atingido:

1. Selecionar uma direção $\vec{d}$ a partir do ponto $\vec{x_0}$
2. Encontrar o mínimo da função f nessa direção, chegando a um novo ponto $\vec{x_1} = \vec{x_0} + \alpha\vec{d}$, onde α é um número real.
3. Tomar $\vec{x_1}$ como o novo ponto de partida $\vec{x_0}$
4. Repetir os passos de 1 a 3 até que uma condição de parada seja atingida: mínimo encontrado ($|\vec{\nabla f}| = 0$), ou máximo número de iterações atingido.

Dessa forma o problema de minimização da função $f(\vec{x})$ se torna um problema de sucessivas determinações de direções de busca e suas respectivas buscas lineares nesssas direções.

2 Objetivos

Os principais objetivos deste trabalho são:

- Implementar numericamente os algoritmos de otimização: Univariante, Powell, Steepest Descent, Fletcher-Reeves, Newton-Raphson e BFGS.
- Avaliar a influência dos parâmetros dos algoritmos nas métricas de convergência e comparar essas úlltimas com os valores esperados da teoria.
- Aplicar os algoritmos implementados na solução do problema de OSR em três casos: uma função quadrática, uma não quadrática e uma terceira função que representa um problema de engenharia (minimizazção da energia de um sistema massa-mola visando encontrar seu ponto de equilíbrio estático)

As funções a serem minimizadas neste trabalho são:

Problema 01a)

$$f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2 \tag{1a}$$

a partir dos pontos iniciais $x^0 = \{2, 2\}^t$ e $x^0 = \{-1, -3\}^t$

Problema 01b)

$$f(x_1, x_2) = (1 + a - bx_1 - bx_2)^2 + (b + x_1 + ax_2 - bx_1x_2)^2 \tag{1b}$$

com $a = 10$ e $b = 1$ a partir dos pontos iniciais $x^0 = \{10, 2\}^t$ e $x^0 = \{-2, -3\}^t$

Problema 02:

$$\Pi(x_1, x_2) = 450(\sqrt{(30 + x_1)^2 + x_2^2} - 30)^2 + 300(\sqrt{(30 - x_1)^2 + x_2^2} - 30)^2 - 360x_2 \tag{2}$$

a partir dos ponto inicial $x^0 = \{0.01, -0.10\}^t$

3 Algoritmos de Busca Linear

Os algoritmos de minimização são executaodos em duas etapas conforme citado anteriormente: uma primeira etapa consiste em determinar direções de busca $\vec{d}$ e minimizar a função f nessa direção, de forma unidimensional, o que significa encontrar o valor de α que minimiza a função $f(x = x_0 + \alpha\vec{d})$ ao longo da direção $\vec{d}$. A busca linear é relizada em duas etapas: Passo constante e refinamento do cálculo de α.

3.1 Passo Constante

A primeira etapa da busca linear consiste numa busca inexata que visa encontrar um intervalo de valores do passo α, $[\alpha_L, \alpha_H]$ que represente uma redução suficientemente grande da função f. Essa implementação numérica incrementa o passo de um valor $d\alpha$ até que a função pare de diminuir.

3.2 Bisseção e Seção Áurea

Após a etapa do passo constante, os métodos da Bisseção ou Seção Áurea são aplicados para encontrar o valor de α, entre os valores determinados no intervalo da etapa 1. Em ambos os casos, o intervalo de ocorrência do valor mínimo de f é sucessivamente reduzido até que seja muito pequeno e considera-se, dado esse critério numérico, solucionado o problema da minimização de f nessa direção, encontrando o passo α_k correspondente a esse mínimo.

O método da bisseção divide sucessivamente o intervalo descartando a parte superior ou inferior do mesmo avaliando o valor da função f na vizinhança esquerda e direita de um α_M médio do intervalo. Já o método da seção áurea utiliza a razão áurea para descarte dos intervalos onde f aumenta. Este último método realiza mais passos, porém possui a vantagem de avaliar menos vezes a função f uma vez que os valores avaliados no passo anterior podem ser reutilizados no passo seguinte do algoritmo, o que pode ser interessante em problemas onde a avaliação da função f é computacionalmente cara.

4 Algoritmos de Direção

Conforme descrito anteriormtente, os métodos de direção determinam direções de busca do mínimo de f a partir de um ponto $\vec{x_k}$. A partir da busca linear encontra-se o valor do passo nessa direção que minimiza f, (α_k), os algoritmos de direção calculam, então, a próxima direção onde se deve minimizar f e a partir do novo ponto $\vec{x_{k+1}} = \vec{x_k} + \alpha_k\vec{d}$. O processo se repete até que uma condição de parada seja atingida. Os algotitmos utilizados nesse trabalho são brevemente descritos a seguir.

4.1 Univariante

O método univariante é o mais simples, onde as direções de busca são as direções canônicas, $\vec{d_k} = \vec{e_k}$.

4.2 Powell

O método de Powell utiliza em seu algoritmo direções denominadas "movimento padrão". O método de Powell gera direçẽs Q-conjugadas, o que representa uma aceleração em relação ao método univariante. O método de Powell converge para o mínimo de uma função quadrática n variáveis em um número finito de passos dado por $n + 1$ (notas de aula).

4.3 Steepest Descent

O métdo Steepest Descent é um método onde as direções são dadas pela direção oposta ao gradiente da função f. Ou seja, $\vec{d_k} = -\vec{\nabla} f(\vec{x_k})$.

4.4 Fletcher-Reeves

Este método consiste em uma adaptação do método dos Gradientes Conjugados que o torna capaz de ser usado para minimização de uma função qualquer. Para anto, duas modificações precisam ser realizadas, a avaliacao da matriz Q (ou Hessiana para funcao nao quadratica) é substituida por uma busca linear e o parametro β modificado (notas de aula).

4.5 Newton-Raphson

O princípio deste método é minimizar uma função f através de uma aproximação local por uma função quadrática. A direção de minimização de f é obtida a partir da derivada da expansão de f numa série de Taylor de ordem 2 em relação a $\vec{d_k}$, de onde se obtém: $\vec{d_k} = -H^{-1}\vec{\nabla} f(\vec{x_k})$, onde H é a matriz Hessiana da função f.

4.6 BFGS

5 Metodologia

Para atingir os objetivos do trabalho, foram programados scripts em linguagem Matlab, organizados conforme esquematizado na figura 1.

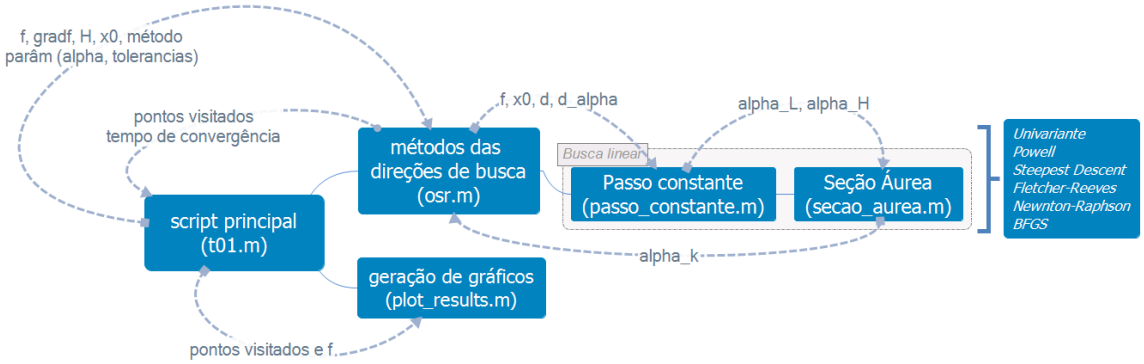


Figura 1: Fluxo dos scripts

Um script principal *t01.m* escolhe os parâmetros do algoritmo: α da busca linear, máximo número de iterações e tolerâncias para avaliar a convergência. Além disso esse script cria as funções, seus gradientes, matriz Hessiana e pontos iniciais, a serem passados como parâmetros para o script que implementa os algoritmos, conforme ilustrado na listagem 1 da seção Anexos.

Em seguida, o script *t01.m* chama o script *osr.m* para cada método e plota as curvas de nível da função f e todos os pontos visitados durante a busca do algoritmo, através do script *plot_result.m* (listagem 2)

O script *osr.m* implementa de fato os algoritmos a partir dos parâmetros recebidos, retornando todos os valores de $\vec{x}$ visitados e o tempo de execução. A listagem de código 3 ilustra a implementação do algoritmo de Powell.

Outros dois scripts invocados na solução dos problemas propostos são *passo_constante.m* e *secao_aurea.m*, que realizam a etapa de busca linear para cada direção de busca. Esses códigos são listados nas listagens 4 e 5, respectivamente.

6 Resultados

Visando a convergência de todos os métodos, foram testados alguns valores dos parâmetros dos algoritmos. A melhor combinação dos parâmetros, para as funções dos problemas 1 e 2 em termos de convergência dos algoritmos está listada na tabela 1.

Tabela 1: Melhores parâmetros para convergência dos algoritmos

	$d\alpha$	TOL(gradiente)	TOL2(busca linear)	max_iter
Problema 01	0.002	1e-4	1e-7	100
Problema 02	0.005	5e-4	1e-7	100

As tabelas 2, 3 e 4 apresentam os principais resultados dos algoritmos implementados. Além dos pontos de mínimo encontrado, as tabelas apresentam também o número de passos para a convergência e o tempo de execução.

Tabela 2: Resultados para a função 1a

método	x^0	x^{min}	passos	Δt(ms)
Univariante	$\{2, 2\}^t$	$\{-0.7142, -0.1428\}^t$	34	4.8
Univariante	$\{-1, -3\}^t$	$\{-0.7144, -0.1429\}^t$	36	6.9
Powell	$\{2, 2\}^t$	$\{-0.7143, -0.1429\}^t$	6	24.8
Powell	$\{-1, -3\}^t$	$\{-0.7143, -0.1429\}^t$	12	7.2
Steepest Descent	$\{2, 2\}^t$	$\{-0.7142, -0.1428\}^t$	25	6.5
Steepest Descent	$\{-1, -3\}^t$	$\{-0.7143, -0.1429\}^t$	7	3.0
FlecherReeves	$\{2, 2\}^t$	$\{-0.7143, -0.1429\}^t$	2	2.5
FlecherReeves	$\{-1, -3\}^t$	$\{-0.7143, -0.1429\}^t$	2	1.7
NewtonRaphson	$\{2, 2\}^t$	$\{-0.7143, -0.1429\}^t$	1	2.5
NewtonRaphson	$\{-1, -3\}^t$	$\{-0.7143, -0.1429\}^t$	1	1.1
BFGS	$\{2, 2\}^t$	$\{-0.7143, -0.1429\}^t$	2	2.2
BFGS	$\{-1, -3\}^t$	$\{-0.7143, -0.1429\}^t$	2	1.5

No caso da função do item 1a, o método de Powell levou mais passos do que o esperado em teoria para uma função quadrática, jque nesse caso é esperado que o mesmo atinja a convergência em até 9 passos.

Tabela 3: Resultados para a função 1b

método	x^0	x^{min}	passos	Δt(ms)
Univariante	$\{10, 2\}^t$	$\{13, 4\}^t$	45	9.9
Univariante	$\{-2, -3\}^t$	$\{7, -2\}^t$	45	10.8
Powell	$\{10, 2\}^t$	$\{13, 4\}^t$	24	11.6
Powell	$\{-2, -3\}^t$	$\{7, -2\}^t$	18	8.0
Steepest Descent	$\{10, 2\}^t$	$\{13, 4\}^t$	46	2.3
Steepest Descent	$\{-2, -3\}^t$	$\{7, -2\}^t$	59	2.6
FlecherReeves	$\{10, 2\}^t$	$\{13, 4\}^t$	41	1.6
FlecherReeves	$\{-2, -3\}^t$	$\{7, -2\}^t$	16	0.9
NewtonRaphson	$\{10, 2\}^t$	$\{10, 1\}^t$	1	0.9
NewtonRaphson	$\{-2, -3\}^t$	$\{12.9999, 4.0001\}^t$	6	4.1
BFGS	$\{10, 2\}^t$	$\{13, 4\}^t$	7	4.1
BFGS	$\{-2, -3\}^t$	$\{7, -2\}^t$	6	5.7

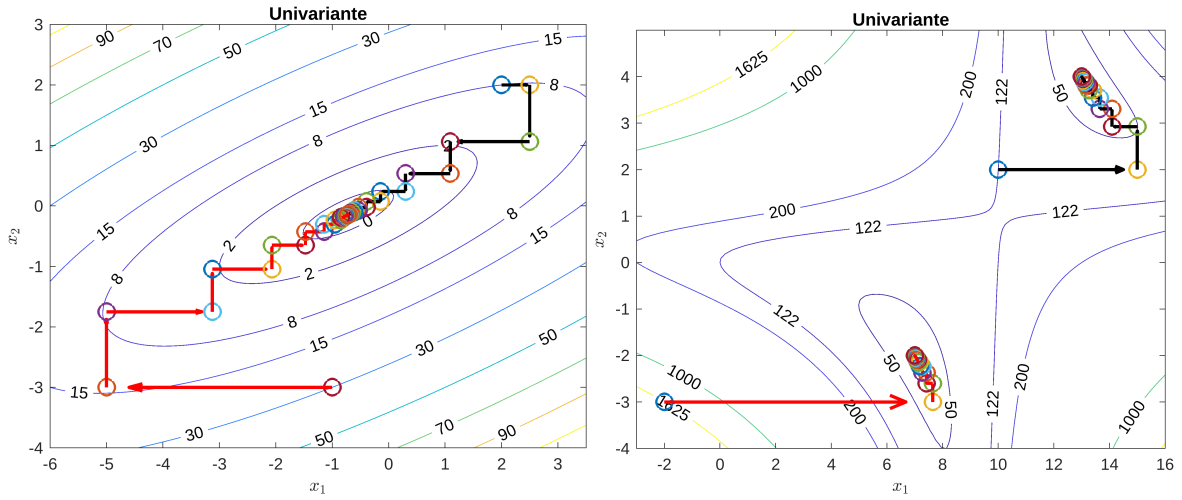
No caso da função do item 1b nota-se que o método de Newton-Raphson não convegiu para o primeiro ponto. Na verdade o algoritmo parou no critério de norma do gradiente no primeiro passo, o que leva a uma convergência para um valor equivocado. De fato a busca linear nesse caso retornou um valor tal que, ao final do primeiro passo o algoritmo caiu num ponto o gradiente de f é muito pequeno. Tal problema pode ser visualizado também graficamente na figura 6b.

Tabela 4: Resultados para a função 2

método	x^0	x^{min}	passos	Δt(ms)
Univariante	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	11	2.1
Powell	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	12	6340.8(**)
Steepest Descent	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	6	0.3
FlecherReeves	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	10	0.4
NewtonRaphson	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	3	4.4
BFGS	$\{0.01, -0.10\}^t$	$\{-0.205, 7.789\}^t$	3	0.3

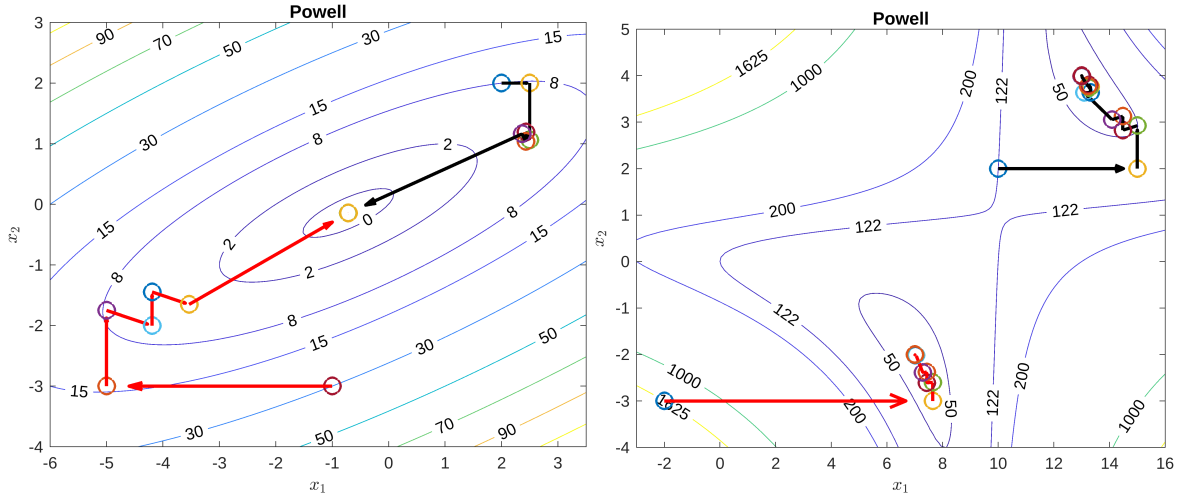
No caso da função do item 2 nota-se que o elevado tempo de execução do método de Powell em relação aos demais.

As figuras



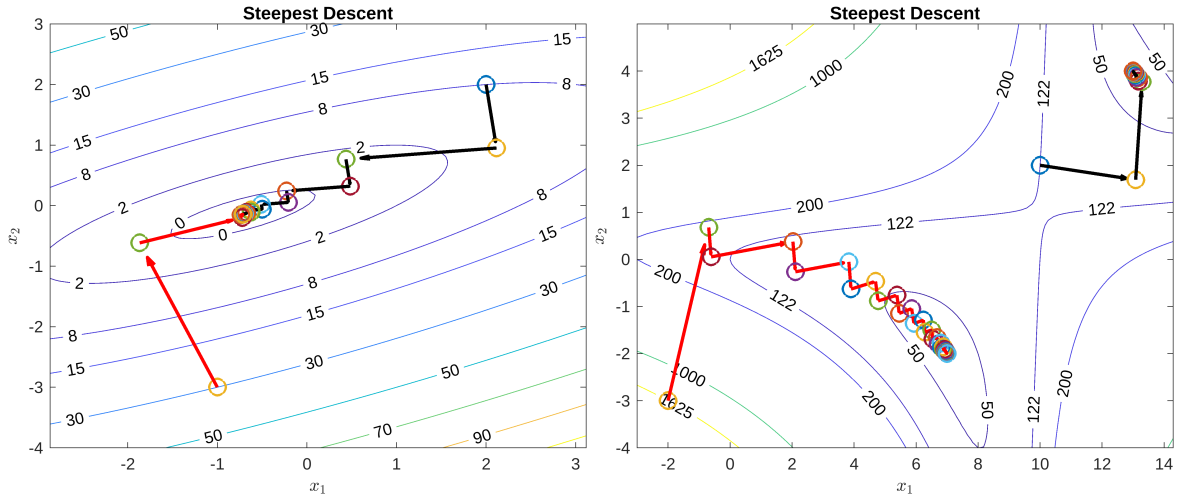
(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 2: Resultados do método univariante, para as duas funções e pontos iniciais



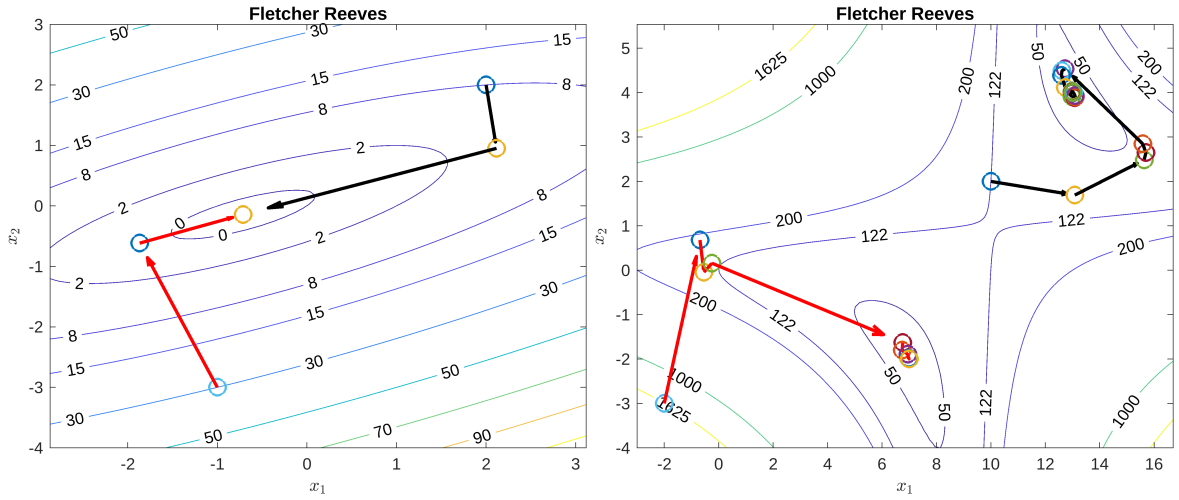
(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 3: Resultados do método de Powell, para as duas funções e pontos iniciais



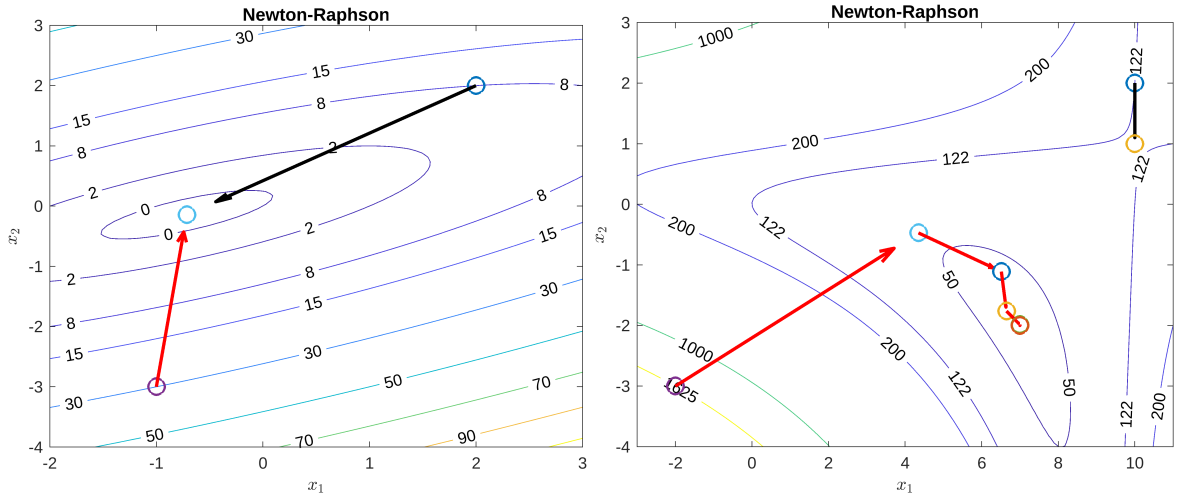
(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 4: Resultados do método Steepest Descent, para as duas funções e pontos iniciais



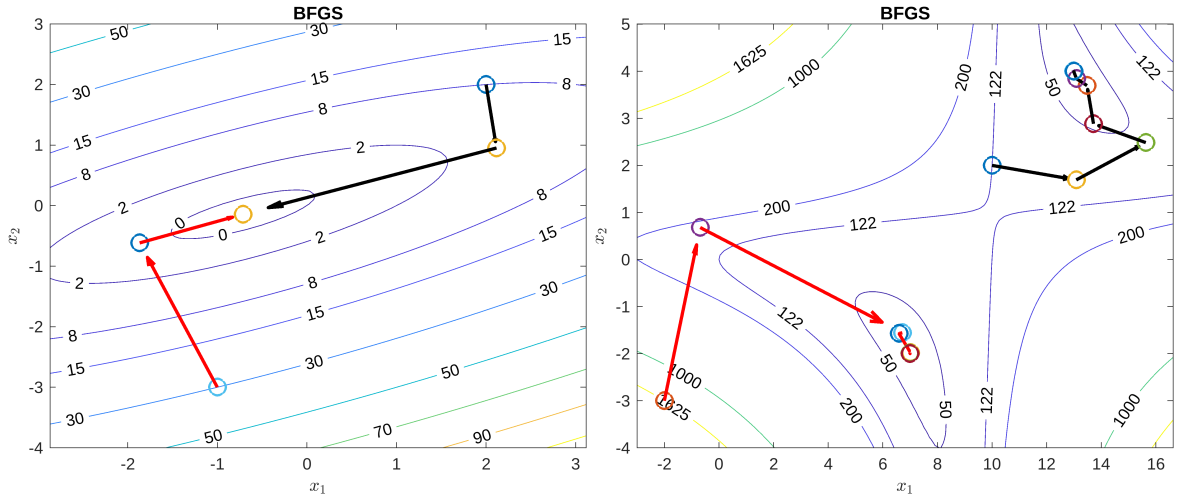
(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 5: Resultados do método Fletcher-Reeves, para as duas funções e pontos iniciais



(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 6: Resultados do método Newton-Raphson, para as duas funções e pontos iniciais



(a) Curvas de nível de f_{1a} e pontos x_k **(b)** Curvas de nível de f_{1b} e pontos x_k

Figura 7: Resultados do método BFGS, para as duas funções e pontos iniciais

7 Anexo - Códigos

asds

Código 1: script t01.m setando parâmetros e criando as funções

```
% dados do item 01a, f, grad f, hess f e x0
fa = @(x) x(1)^2-3*x(1)*x(2)+4*x(2)^2+x(1)-x(2);
gfa = @(x) [2*x(1)-3*x(2)+1 ; -3*x(1)+8*x(2)-1];
Ha = @(x) [2 -3;-3 8];
x01 = [2;2];
x02 = [-1;-3];
% parametros dos algoritmos
iter_max = 100;
a = 0.002; % passo
TOL = 1e-4; % parada do gradiente
TOL2 = 1e-7; % busca linear
methods = ["Univariante","Powell","Steepest Descent","Fletcher Reeves","Newton-Raphson","BFGS"];
```

Código 2: script t01.m chamando o script osr.m para a função do item 1a para cada um dos 6 métodos estudados

```
fprintf('\n******************* ITEM 01A **********************\n');
for method = 1:6
    fprintf('---%s---\n', methods(method));

    fprintf('x0=[%2d,%2d]: ',x01(1), x01(2));
    [x_1,t] = osr (fa, gfa, Ha, x01, method, iter_max, a, TOL, TOL2);
    fprintf('(%.1fms), xmin=[%0.4f,%0.4f], f=%0.4f\n', t*1000, x_1(1,end), x_1(2,end), fa(x_1(:,end)));

    fprintf('x0=[%2d,%2d]: ',x02(1), x02(2));
    [x_2,t] = osr (fa, gfa, Ha, x02, method, iter_max, a, TOL, TOL2);
    fprintf('(%.1fms), xmin=[%0.4f,%0.4f], f=%0.4f\n', t*1000, x_2(1,end), x_2(2,end), fa(x_2(:,end)));

    plot_result(min([x_1(1,:), x_2(1,:)])-dx,max([x_1(1,:), x_2(1,:)])+dx,min([x_1(2,:), x_2(2,:)])-dx,max([x_1(2,:), x_2(2,:)])+dx, x_1,
        x_2, methods(method), 1)
    exportgraphics(gcf,strcat('./figures/img01A_m0',num2str(method),'.png'),'Resolution',500)
end
```

Código 3: script osr.m implementando o método de Powell

```
function [x_,time_elap] = osr (f, gf, H, x0, method, iter_max, a, TOL, TOL2)
% 1. Univariante
% 2. Powell
% 3. Steepest Descent
% 4. Flecher?Reeves
% 5. Newton?Raphson
% 6. BFGS

k=0;
conv=0; %flag convergencia
tstart = tic;
switch method
    case 2
    % 2. Powell
        x_ = x0;
        x = x0;
        while k < iter_max
            j = 1;
            n = 2;
            y = [[1;0],[0;1]];
            while j <= n
                [alpha_L, alpha_H] = passo_constante(f, x, y(:,1), a);
                alpha_k = secao_aurea(f, x, y(:,1), TOL2, alpha_L, alpha_H);
                k=k+1;
                x = x + alpha_k*y(:,1);
                x_ = [x_,x];
                [alpha_L, alpha_H] = passo_constante(f, x, y(:,2), a);
                alpha_k = secao_aurea(f, x, y(:,2), TOL2, alpha_L, alpha_H);
                k=k+1;
                x = x + alpha_k*y(:,2);
                x_ = [x_,x];
                d = x-x0;
                [alpha_L, alpha_H] = passo_constante(f, x, d, a);
                alpha_k = secao_aurea(f, x, d, TOL2, alpha_L, alpha_H);
                k=k+1;
                x0 = x + alpha_k*d;
                x=x0;
                x_ = [x_,x];

                y(:,1) = y(:,2);
                y(:,2) = d;

                j = j+1;
            end
            if norm(gf(x)) < TOL
                fprintf('%d steps!', k);
                conv=1;
                break;
            end
        end
        if conv == 0
            fprintf('Nao convergiu apos %d steps', k);
        end
```

Código 4: script osr.m implementando o método de Powell

```
        function [alpha_L, alpha_H] = passo_constante(f, x0, d, a)
        alpha = 0;
        f_min = Inf;
        f_val = f(x0);
        alphas = [];
        f1 = f(x0 - a*d);
        f2 = f(x0 + a*d);
        if f1 < f2
            a=-a; % desce a esq (d-)
        end
        while f_val <= f_min
            x = x0 + alpha * d;
            f_val = f(x);
            if f_val < f_min
                f_min = f_val;
            end
            alphas = [alphas; alpha];
            alpha = alpha + a;
        end
        alpha_L = alphas(end-1);
        alpha_H = alphas(end);
        if a < 0
            alpha_H = alphas(end-1);
            alpha_L = alphas(end);
        end
    end
```

Código 5: script osr.m implementando o método de Powell

```
function alpha_k = secao_aurea (f, x0, d, TOL, alpha_L, alpha_H)
        ra = (sqrt(5)-1)/2;
        b = norm(alpha_L-alpha_H);
        alpha_E = alpha_L + (1-ra)*b;
        alpha_D = alpha_L + ra*b;
        f1 = f(x0 + alpha_E * d);
        f2 = f(x0 + alpha_D * d);
        while b > TOL
              if f1 > f2
                    alpha_L = alpha_E;
                    alpha_E = alpha_D;
                    b = norm(alpha_L-alpha_H);
                    alpha_D = alpha_L + ra*b;
                    % avaliar menos vezes a funcao f
                    f1 = f2;
                    f2 = f(x0 + alpha_D * d);
              else
                    alpha_H = alpha_D;
                    alpha_D = alpha_E;
                    b = norm(alpha_L-alpha_H);
                    alpha_E = alpha_L + (1-ra)*b;
                    % avaliar menos vezes a funcao f
                    f2 = f1;
                    f1 = f(x0 + alpha_E * d);
              end
        end
        alpha_k = (alpha_L+alpha_H)/2;
end
```