Lista 03

Otimização Convexa

Patrícia Kovaleski

23 de novembro de 2018

1 Questão 6.1

Nesta questão é pedido para minimizarmos a função:

$$f(x) = \frac{1}{2}x^T \mathbf{Q} x + b^T x \tag{1}$$

para valores de $\mathbf{Q}$ e b definidos utilizando o método do gradiente conjugado.

O método do gradiente conjugado foi implementado de acordo com o Algoritmo 6.2 do livro *Practical Optimization de Antoniou & Lu* e está descrito, juntamente com os demais métodos desenvolvidos nesta lista de exercícios, no arquivo *functions.py*, em anexo. No trecho de código abaixo temos o resultado da execução do script *lista03.py* quando executado para a Questão 6.1 a partir dos pontos:

$$p_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad p_2 = \begin{bmatrix} -10 \\ -10 \\ -10 \\ -10 \\ -10 \\ -10 \\ -10 \\ -10 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} \quad p_3 = \begin{bmatrix} -2.43025 \\ -1.5579 \\ 3.8982 \\ -8.87454 \\ 0.58752 \\ -2.26325 \\ -2.36888 \\ -5.30948 \\ 2.83216 \\ -2.4562 \\ 2.17550 \\ 4.31107 \\ -0.79168 \\ -7.42939 \\ 6.46193 \\ -4.20968 \end{bmatrix}$$

escolhidos arbitrariarmente de forma a explorar a estabilidade do algoritmo.

Listing 1: Resposta do script 'lista03.py' quando executado para a Questão 6.1.

```
python lista03.py -exe_num 6.1
Ponto Inicial: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
x: [ 0.03423704 0.02423337 0.02423337 0.03423704 -0.00143237 -0.02088798
   -0.02088798 -0.00143237 0.03321913 0.02738606 0.02738606 0.03321913
   -0.00663057 -0.00486331 -0.00486331 -0.00663057],
fx: -0.11908, num_iter: 8, time: 0.47207 ms

Ponto Inicial: [-10, -10, -10, -10, -10, -10, -10, -10, 2, 2, 2, 2, 2, 2, 2,
    2]
x: [ 0.03423704 0.02423337 0.02423337 0.03423704 -0.00143237 -0.02088798
   -0.02088798 -0.00143237 0.03321913 0.02738606 0.02738606 0.03321913
   -0.00663057 -0.00486331 -0.00486331 -0.00663057],
fx: -0.11908, num_iter: 8, time: 0.42605 ms

Ponto Inicial: [-2.43025375 -1.5579628 3.8982761 -8.87454095 0.58752312
   -2.26325137 -2.36888457 -5.30948481 2.83216087 -2.4562392 2.17550919
   4.31107924 -0.79168831 -7.42939936 6.46193689 -4.20968789]
x: [ 0.03423727 0.02423323 0.02423308 0.03423702 -0.00143275 -0.02088792
   -0.02088776 -0.00143256 0.03321916 0.02738572 0.0273858 0.0332194
   -0.00663063 -0.00486316 -0.00486328 -0.00663092],
fx: -0.11908, num_iter: 13, time: 0.61893 ms
```

Vemos que o método do gradiente conjugado alcançou o mínimo da função para os três pontos iniciais analisados, necessitando de um baixo número de iterações. Além de sua rápida convergência este método tem como pontos positivos não utilizar uma busca em linha nem calcular a inversa da Hessiana, passos que costumam prejudicar a estabilidade dos algoritmos em certos casos, além de serem custosos computacionalmente.

2 Questão 6.2

Utilize o algoritmo de Fletcher-Reeves para minimizar a função de Rosenbrock:

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \tag{2}$$

a partir dos pontos iniciais: $[2, -2]^T$, $[-2, 2]^T$ e $[-2, -2]^T$.

Na Figura 1 temos o gráfico da função (2) para $x_1, x_2 \in [-5, 5]$. Seu mínimo encontra-se no ponto $x^* = [1, 1]$, atingindo o valor $f(x^*) = 0$. Já no trecho de código abaixo temos o resultado do algoritmo quando executado a partir de cada ponto inicial pedido.

Listing 2: Resposta do script 'lista03.py' quando executado para a Questão 6.2.

```
python lista03.py -exe_num 6.2
Ponto Inicial: [-2.0, 2.0]
```

```
x: [1.00037855 1.00075785], fx: 0.00000, num_iter: 2660, time: 1943.95900 ms

Ponto Inicial: [2.0, -2.0]
x: [0.99980486 0.9996088 ], fx: 0.00000, num_iter: 179, time: 117.55705 ms

Ponto Inicial: [-2.0, -2.0]
x: [0.99907078 0.99813964], fx: 0.00000, num_iter: 764, time: 571.67101 ms
```

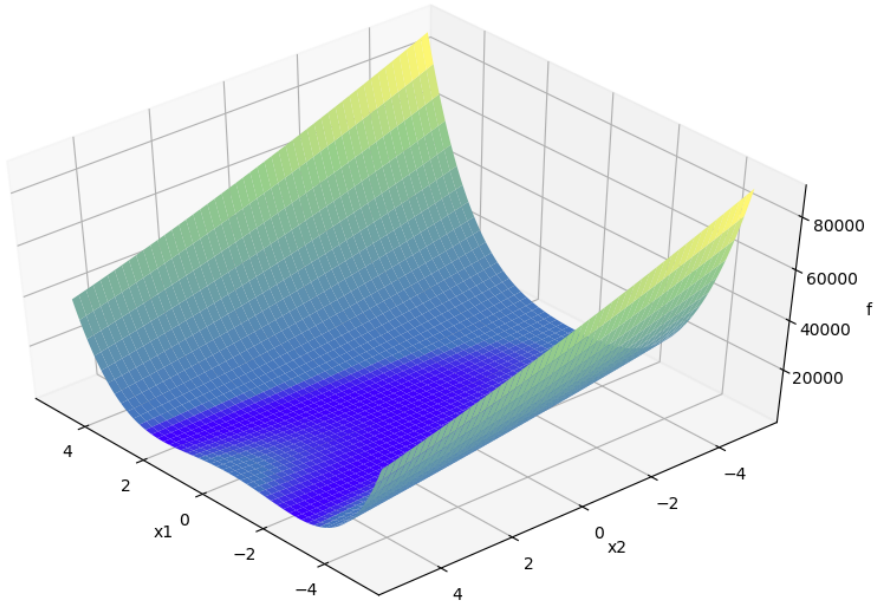

Figura 1: Gráfico da função (2) para $x_1, x_2 \in [-5, 5]$.

O mínimo foi atingido para todos os casos analisados com um número de iterações relativamente alto. Neste método, diferente do gradiente conjugado, é preciso aplicar uma função de busca em linha em sua iteração e esta escolha pode afetar a velocidade de convergência do algoritmo. Neste caso, foi aplicado o *Backtracking Line Search* devido ao seu bom desempenho nas Listas anteriores e por não depender de um intervalo inicial de busca.

3 Questão 6.3

Nesta questão é pedido para minimizarmos a função:

$$f(x) = 5x_1^2 - 9x_1x_2 + 4.075x_2^2 + 1 \tag{3}$$

a partir do ponto inicial: $[1, 1]^T$.

(a) Utilizando duas iterações do algoritmo de gradiente conjugado

(b) Compare os resultados da primeira iteração do algoritmo de gradiente conjugado com a obtida pelo método de *Steepest Descent*

(c) Compare os resultados da segunda iteração do algoritmo de gradiente conjugado com a obtida pelo método de *Steepest Descent*

Na Figura 2 temos o gráfico da função (3) para $x_1, x_2 \in [-25, -15]$. Seu mínimo é encontrado no ponto $x^* = [-16.3 - 18.0]$, atingindo o valor $f(x^*) = -8.15$. Já no trecho de código abaixo temos o resultado do algoritmo quando executado a partir do ponto inicial desejado para um número máximo de iterações igual a 1, 2 e 15000.

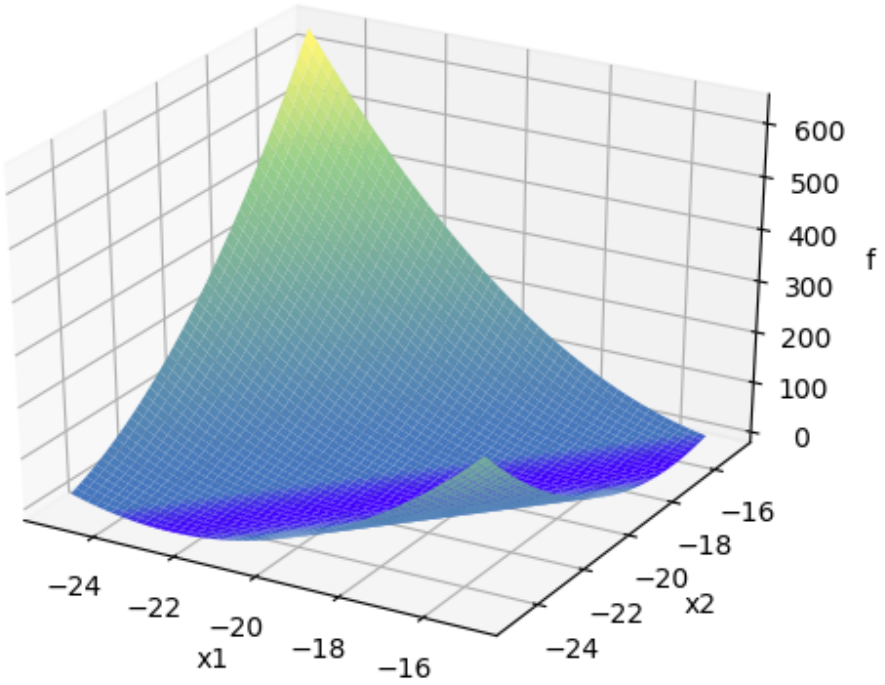

Figura 2: Gráfico da função (3) para $x_1, x_2 \in [-25, -15]$.

Listing 3: Resposta do script 'lista03.py' quando executado para a Questão 6.3.

```
python lista03.py -exe_num 6.3

Max iterations: 1
Exercise 6.3 - Conjugate Gradient
```

```
x: [0.87651718 1.0524802 ], fx: 0.92921, num_iter: 1, time: 0.27299 ms

Exercise 6.3 - Steepest Descent without line search
Reached max iterations!
x: [0.87651718 1.0524802 ], fx: 0.92921, num_iter: 1, time: 0.26107 ms



Max iterations: 2
Exercise 6.3 - Conjugate Gradient
x: [-16.3 -18. ], fx: -8.15000, num_iter: 2, time: 0.19312 ms

Exercise 6.3 - Steepest Descent without line search
Reached max iterations!
x: [0.72660018 0.6997343 ], fx: 0.78573, num_iter: 2, time: 0.15903 ms



Max iterations: 15000
Exercise 6.3 - Conjugate Gradient
x: [-16.3 -18. ], fx: -8.15000, num_iter: 2, time: 0.21601 ms

Exercise 6.3 - Steepest Descent without line search
x: [-16.29989602 -17.99988456], fx: -8.15000, num_iter: 1700, time: 60.10795
    ms
```

max iter	Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
1	*Conjugate Gradient*	[0.87652, 1.05248]	0.92921	1	0.27299
	Steepest Descent	[0.87652, 1.05248]	0.92921	1	0.26107
2	*Conjugate Gradient*	[-16.3, -18.0]	-8.15	2	0.19312
	Steepest Descent	[0.72660, 0.69973]	0.78573	2	0.15903
15000	*Conjugate Gradient*	[-16.3, -18.0]	-8.15	2	0.21601
	Steepest Descent	[-16.29990, -17.99988]	-8.15	1700	60.10795

Tabela 1: Comparação dos resultados obtidos para os métodos *Conjugate Gradient* e *Steepest Descent* quando executados com o limite de 1, 2 e 15000 iterações.

Na Tabela 1 temos a comparação dos resultados obtidos para cada método quando executado para os diferentes limites de número de iterações. Vemos que quando limitados a uma iteração os métodos atingiram os mesmos valores. Isso ocorre pois a primeira direção de minimização é igual para ambos os algoritmos.

Porém, a partir da segunda iteração as direções são atualizadas de forma diferente e os resultados começam a divergir. Percebemos que o gradiente conjugado consegue um resultado

muito melhor, atingindo, na verdade, o ponto ótimo em apenas 2 iterações, enquanto o *Steepest Descent* necessitou de 1700 iterações.

4 Questão 7.7

Nesta questão é pedido para implementarmos o algoritmo de Quasi-Newton baseado na DFP e minimizarmos a função (2) para diferentes pontos iniciais, comparando os resultados com os obtidos pela Questão 6.2.

O gráfico da função (2) pode ser observado na Figura 1 e conforme analisado anteriormente, esta função possui um mínimo em $x^* = [1, 1]$, atingindo o valor $f(x^*) = 0$. Por ter como o intuito a comparação de resultados com a Questão 6.2, foram utilizados os mesmos pontos iniciais para avaliação: $[2, -2]^T$, $[-2, 2]^T$ e $[-2, -2]^T$. No trecho de código abaixo temos o resultado do algoritmo quando executado a partir de cada ponto inicial.

Listing 4: Resposta do script 'lista03.py' quando executado para a Questão 7.7.

```
python lista03.py -exe_num 7.7
Ponto Inicial: [-2.0, 2.0]
x: [1.00000899 1.00001691], fx: 0.00000, num_iter: 168, time: 16.64305 ms

Ponto Inicial: [2.0, -2.0]
x: [0.99999824 0.99999619], fx: 0.00000, num_iter: 17, time: 1.43600 ms

Ponto Inicial: [-2.0, -2.0]
x: [1.08957035 1.0382005 ], fx: 2.22702, num_iter: 289, time: 20.78009 ms
```

Ponto Inicial	Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
$[2, -2]^T$	Fletcher-Reeves	[1.00037, 1.00075]	0.00000	2660	1943.95900
	Quasi-Newton	[1.00000, 1.00001]	0.00000	168	16.64305
$[-2, 2]^T$	Fletcher-Reeves	[0.99980 0.99960]	0.00000	179	117.55705
	Quasi-Newton	[0.99999, 0.99999]	0.00000	17	1.43600
$[-2, -2]^T$	Fletcher-Reeves	[0.99907, 0.99813]	0.00000	764	571.67101
	Quasi-Newton	[1.08957, 1.0382]	2.22702	289	20.78009

Tabela 2: Comparação dos resultados obtidos para os métodos de Quasi-Newton e Fletcher-Reeves quando executados para diferentes pontos iniciais.

Vemos que o método de Quasi-Newton encontrou o mínimo da função para os dois primeiros pontos, atingindo um valor próximo ao mínimo para o terceiro e último ponto. Na Tabela 2 podemos comparar os resultados obtidos nesta Questão com aqueles obtidos pelo método de Fletcher-Reeves na Questão 6.2.

Observamos também que o método de Quasi-Newton possui uma complexidade computacional muito menor, executando iterações mais leves e em menor número. Entretanto, ele não obteve o ótimo para todos os pontos, diferente do método de Fletcher-Reeves. Logo,

para definirmos qual algoritmo seria o mais eficiente precisamos ponderá-los de acordo com o caso ao qual serão aplicados.

5 Questão 7.8

Minimize a função (3) aplicando o algoritmo de BFGS a partir do ponto inicial $[0, 0]^T$. Compare os resultados com aqueles obtidos pela Questão 6.3 (BFP) e 5.4 (*Steepest Descent* sem busca em linha).

O gráfico da função (3) pode ser visto na Figura 2 e conforme discutido anteriormente, seu mínimo é encontrado no ponto $x^* = [-16.3 - 18.0]$ atingindo o valor $f(x^*) = -8.15$. No trecho de código abaixo temos o resultado obtido por cada método quando executado a partir do ponto inicial especificado.

Listing 5: Resposta do script 'lista03.py' quando executado para a Questão 7.8.

```
python lista03.py -exe_num 7.8
Exercise 7.8 - BFGS
x: [-16.3 -18. ], fx: -8.15000, num_iter: 2, time: 0.83804 ms

Exercise 7.8 - DFP
x: [-16.3 -18. ], fx: -8.15000, num_iter: 2, time: 0.58484 ms

Exercise 7.8 - Steepest Descent without line search
x: [-16.29991 -17.99990], fx: -8.15000, num_iter: 3756, time: 166.75210 ms
```

Todos os algoritmos avaliados atingiram o ponto mínimo da função, porém, é notável a maior eficiencia dos métodos de Quasi-Newton (BFGS e DFP). Estes foram capazes de convergir em apenas 2 passos, enquanto o *Steepest Descent* necessitou de mais de 3700 iterações. Já dentre os métodos de Quasi-Newton o desempenho foi equivalente, não sendo possível apontar a superioridade de algum para o caso avaliado.

Os métodos de Quasi-Newton, como indicado pelo nome, são inspirados pelos métodos de Newton, porém não utilizam o cálculo direto da Hessiana e de sua inversa. Eles obtem uma aproximação para a inversa da Hessiana utilizando direções conjugadas, de modo que em cada iteração esta aproximação fica mais precisa. Estes métodos são extremamente eficientes, alcançando o mínimo em poucas iterações, como visto nas Questões resolvidas acima. Outro ponto positivo é o fato de não necessitarem de uma expressão explícita para o cálculo da segunda derivada da função.

Já o *Steepest Descent*, como estudado anteriormente, também não efetua o cálculo da Hessiana e de sua inversa. Porém, suas atualizações são bem menos efetivas, necessitando de um número consideravelmente maior de iterações. Neste método não são utilizadas direções conjugadas, que costumam ajudar na convergência.