Lista 01

Otimização Convexa

Patrícia Kovaleski

19 de outubro de 2018

1 Questão 4.2

Sabe-se que o polinômio de quinta ordem:

$$f(x) = -5x^5 + 4x^4 - 12x^3 + 11x^2 - 2x + 1 \tag{1}$$

é uma função unimodal para o intervalo $[-0.5, 0.5]$. Na figura 1 temos o gráfico da função (1) para este intervalo. Vemos que o valor mínimo da função encontra-se um pouco abaixo de 1.0 para um valor de x de aproximadamente 0.1.

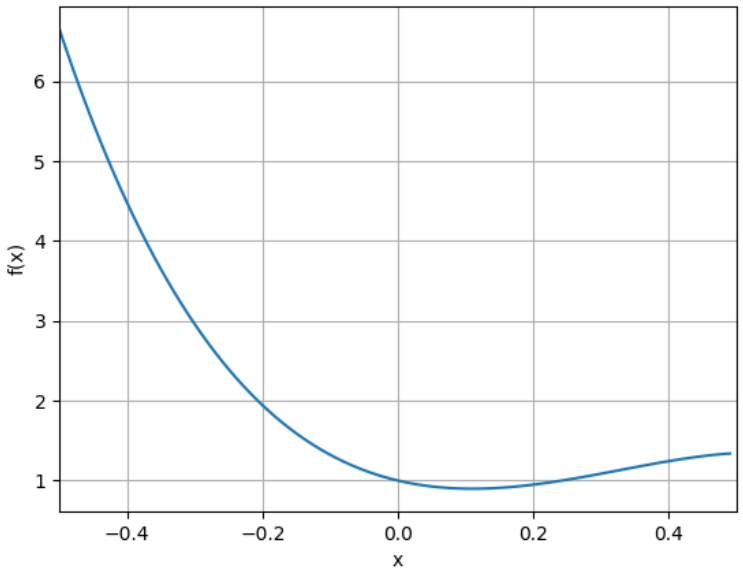

Figura 1: Gráfico da função (1) para o intervalo $[-0.5, 0.5]$.

1.1 Itens (a)-(g)

Nos itens (a) - (f) desta questão é pedido para minimizarmos função f(x) (1) utilizando os seguintes algoritmos:

(a) Busca Dicotômica

(b) Busca de Fibonacci

(c) Busca na Seção Dourada

(d) Método de Interpolação Quadrática

(e) Método de Interpolação Cúbica

(f) Algoritmo de Davies, Swann, e Campey

(g) Backtracking Line Search

para o intervalo $[-0.5, 0.5]$ com um intervalo de incerteza menor que 10^{-5}.

No trecho de código abaixo temos o resultado obtido ao executarmos cada um dos algoritmos de busca para o problema em questão. Para os métodos aplicados em (a)-(d), é utilizado o intervalo total (no caso, $[-0.5, 0.5]$) como espaço inicial de busca. Já para os métodos em (e)-(g) é preciso fornecer um ou mais pontos de entrada como parâmetros iniciais para sua resolução. Desta forma, cada um destes foram executados três vezes para diferentes valores de entrada escolhidos arbitrariarmente.

Vale ressaltar que o algoritmo de Davies, Swann, e Campey, (f), não obteve resultados satisfatórios ao se utilizar valores negativos ou nulo como ponto de entrada.

Listing 1: Resposta do script 'lista01.py' quando executado para a Questão 4.2.

```
$ python lista01.py -exe_num 4.2 -min -0.5 -max 0.5
Exercise 4.2:

Dichotomous Search
x: 0.10986, f(x): 0.89763, num_iter: 17, time: 0.07319 ms

Fibonacci Search
x: 0.10986, f(x): 0.89763, num_iter: 23, time: 0.08082 ms

Golden Section Search
x: 0.10986, f(x): 0.89763, num_iter: 24, time: 0.04697 ms

Quadratic Interpolation Search
x: 0.10986, f(x): 0.89763, num_iter: 10, time: 0.04005 ms

Cubic Interpolation Search
x: 0.11016, f(x): 0.89763, num_iter: 4, time: 0.12708 ms, for [-0.2, 0.1,
   0.2]
```

```
x: 0.10460, f(x): 0.89784, num_iter: 4, time: 0.07486 ms, for [-0.3, 0.2,
   0.3]
x: 0.10986, f(x): 0.89763, num_iter: 3, time: 0.05698 ms, for [0.1, 0.3,
   0.5]

Davies, Swann and Campey Search
x: 0.10986, f(x): 0.89763, num_iter: 5, time: 0.04697 ms, for 0.1
x: 0.10986, f(x): 0.89763, num_iter: 5, time: 0.03386 ms, for 0.2
x: 0.10986, f(x): 0.89763, num_iter: 5, time: 0.08202 ms, for 0.5

Backtraking Line Search
x: 0.10989, f(x): 0.89763, num_iter: 8, time: 0.26417 ms, for -0.5
x: 0.10989, f(x): 0.89763, num_iter: 4, time: 0.15712 ms, for 0.2
x: 0.10989, f(x): 0.89763, num_iter: 7, time: 0.20504 ms, for 0.5
```

O código implementado para resolver cada um dos algoritmos mencionados acima encontra-se no script *functions.py*, em anexo.

1.2 Item (h)

Comparar a eficiência computacional dos métodos (a)-(g) em termos do número de avaliações de funções.

Na Tabela 1 temos a comparação dos métodos de busca aplicados nos itens (a)-(g) para o valor mínimo da função encontrado, número de iterações do algoritmo, tempo total de execução e tempo médio por iteração em microsegundos. Para os métodos cujos resultados variam dependendo dos pontos de entrada ((e)-(g)) é considerado a média dos valores encontrados.

Método	Mínimo	# iterações	Tempo total de execução / média por iteração (μs)
(a) Busca Dicotômica	0.89763	17	73.19 / 4.31
(b) Busca de Fibonacci	0.89763	23	80.82 / 3.51
(c) Busca na Seção Dourada	0.89763	24	46.97 / 1.96
(d) Método de Interpolação Quadrática	0.89763	10	40.05 / 4.01
(e) Método de Interpolação Cúbica	0.89770	3.7	86.31 / 23.33
(f) Algoritmo de Davies, Swann, e Campey	0.89763	5	54.28 / 10.86
(g) Backtracking Line Search	0.89763	6.3	208.78 / 33.14

Tabela 1: Comparação dos métodos de busca aplicados nos itens (a)-(g) para a função (1).

Na Busca Dicotômica cada iteração avalia a função duas vezes, totalizando 34 avaliações, e reduz o intervalo de busca pela metade. Já na Busca de Fibonacci a função é avaliada apenas uma vez por iteração, o que pode levar a um ganho considerável de eficiência em relação a Busca Dicotômica para casos no qual a função a ser minimizada possui alto custo computacional.

A principal desvantagem da Busca de Fibonacci é que o número de iterações deve ser fornecido como entrada. Um método de busca semelhante que pode ser executado até se alcançar a precisão desejada é a Busca na Seção Dourada. Este método obteve o mesmo resultado, com praticamente o mesmo número de iterações e avaliações que a Busca de Fibonacci mas em 60% de seu tempo de execução; mostrando-se mais eficiente para este caso.

No Método de Interpolação Quadrática a função é avaliada três vezes na primeira iteração e uma nas seguintes, totalizando 12 avaliações. Este método realiza menos avaliações que a Busca na Seção Dourada com aproximadamente o mesmo tempo de execução. Isto mostra que cada iteração da Interpolação Quadrática é mais pesada computacionalmente, porém, pode ser mais eficiente pelo menor número de avaliações da função custo.

O Método de Interpolação Cúbica, por sua vez, obteve o menor número de iterações dentre todos os algoritmos, porém, também obteve um dos maiores tempos de execução. Ou seja, apesar de necessitar de menos iterações para alcançar a convergência, pois a interpolação cúbica é mais precisa que a quadrática, elas possuem um custo computacional muito maior.

Já o Algoritmo de Davies, Swann, e Campey também necessitou de poucas iterações para alcançar a convergência a um relativamente alto custo computacional por iteração.

Por fim, o Backtracking Line Search consegue atingir a convergência com um baixo número de iterações, porém, ele obteve o maior tempo total de execução dentre todos os métodos.

2 Questão 4.3

A função

$$f(x) = ln^2(x-2) + ln^2(10-x) - x^{0.2} \tag{2}$$

é unimodal para o intervalo $[6, 9.9]$. Repita a Questão 4.2 para função (2).

Na figura 2 temos o gráfico da função (2) para o intervalo $[6, 9.9]$. Vemos que o valor mínimo da função encontra-se próximo a 2.0 para um valor de x de aproximadamente 8.5.

2.1 Itens (a)-(g)

Repetindo a Questão 4.2, nos itens (a) - (f) é pedido para minimizarmos função f(x) (2) utilizando os seguintes algoritmos:

(a) Busca Dicotômica

(b) Busca de Fibonacci

(c) Busca na Seção Dourada

(d) Método de Interpolação Quadrática

(e) Método de Interpolação Cúbica

(f) Algoritmo de Davies, Swann, e Campey

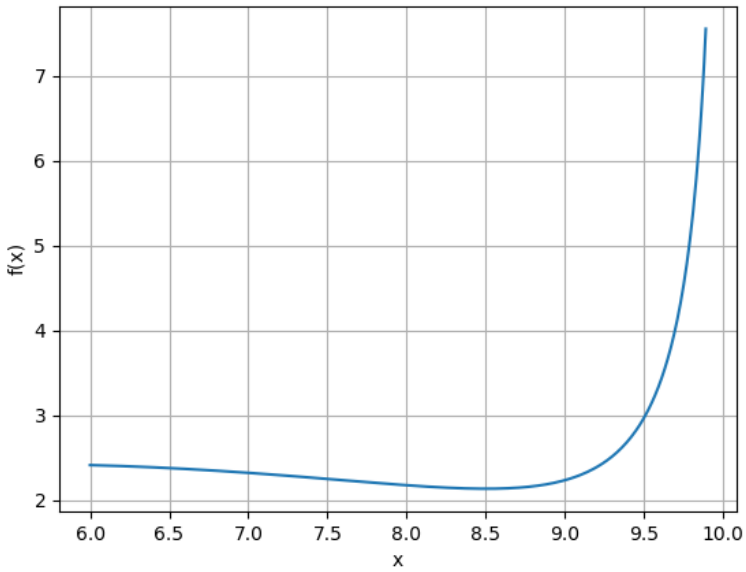

Figura 2: Gráfico da função (2) para o intervalo $[6, 9.9]$.

(g) Backtracking Line Search

para o intervalo $[6, 9.9]$ com um intervalo de incerteza menor que 10^{-5}.

No trecho de código abaixo temos o resultado obtido ao executarmos cada um dos algoritmos de busca para o problema em questão.

Listing 2: Resposta do script 'lista01.py' quando executado para a Questão 4.3.

```
$ python lista01.py -exe_num 4.3 -min 6 -max 9.9
Exercise 4.3:

Dichotomous Search
x: 8.50158, f(x): 2.13384, num_iter: 19, time: 0.09108 ms

Fibonacci Search
x: 8.50159, f(x): 2.13384, num_iter: 27, time: 0.08988 ms

Golden Section Search
x: 8.50159, f(x): 2.13384, num_iter: 27, time: 0.05007 ms

Quadratic Interpolation Search
x: 8.45813, f(x): 2.13429, num_iter: 49, time: 0.13995 ms

Cubic Interpolation Search
x: 8.58529, f(x): 2.13570, num_iter: 17, time: 0.35596 ms, for [7, 8, 9]
x: 8.14698, f(x): 2.15693, num_iter: 9, time: 0.16618 ms, for [6.9, 7, 8]
```

```
x: 8.08467, f(x): 2.16426, num_iter: 24, time: 0.42987 ms, for [6, 8, 9]
x: 8.47301, f(x): 2.13403, num_iter: 4, time: 0.07296 ms, for [8, 8.5, 9]

Davies, Swann and Campey Search
x: 8.50159, f(x): 2.13384, num_iter: 7, time: 0.11301 ms, for 7
x: 8.50159, f(x): 2.13384, num_iter: 6, time: 0.07010 ms, for 8.5
x: 8.50159, f(x): 2.13384, num_iter: 6, time: 0.05507 ms, for 9

Backtraking Line Search
x: 8.50159, f(x): 2.13384, num_iter: 15, time: 0.31900 ms, for 6
x: 8.50159, f(x): 2.13384, num_iter: 14, time: 0.30589 ms, for 7
x: 8.50159, f(x): 2.13384, num_iter: 13, time: 0.42582 ms, for 8
x: 8.50159, f(x): 2.13384, num_iter: 13, time: 0.32091 ms, for 9
admins-MacBook-Air:lista01 admin$
```

2.2 Item (h)

Comparar a eficiência computacional dos métodos (a)-(g) em termos do número de avaliações de funções.

Na Tabela 2 temos a comparação dos métodos de busca aplicados nos itens (a)-(g) para o valor mínimo da função encontrado, número de iterações do algoritmo, tempo total de execução e tempo médio por iteração em microsegundos. Para os métodos cujos resultados variam dependendo dos pontos de entrada ((e)-(g)) é considerado a média dos valores encontrados.

Método	Mínimo	# iterações	Tempo total de execução / média por iteração (μs)
(a) Busca Dicotômica	2.13384	19	91.08 / 4.79
(b) Busca de Fibonacci	2.13384	27	89.88 / 3.33
(c) Busca na Seção Dourada	2.13384	27	50.07 / 1.85
(d) Método de Interpolação Quadrática	2.13429	49	139.95 / 2.86
(e) Método de Interpolação Cúbica	2.14773	13.5	256.24 / 18.98
(f) Algoritmo de Davies, Swann, e Campey	2.13384	6.3	79.39 / 12.60
(g) Backtracking Line Search	2.13384	13.75	34.29 / 2.49

Tabela 2: Comparação dos métodos de busca aplicados nos itens (a)-(g) para a função (2).

As conclusões sobre o número de avaliações da função para cada método, se mantiveram de acordo com a Questão 4.2, com as devidas proporções.

Os métodos ((a)-(c)) tiveram um desempenho semelhante ao observado na questão anterior. Porém, o Método de Interpolação Quadrática obteve um desempenho pior do que o observado anteriormente, necessitando um número muito maior de iterações para alcançar

a precisão desejada na convergência. Esta variação pode ter sido causada por alguma pequena divergência durante a execução do algoritmo devido as particularidades da função custo avaliada.

Também podemos observar como o resultado obtido pelo Método de Interpolação Cúbica pode variar dependendo dos valores escolhidos para inicialização.

Além disso, nota-se que o Backtracking Line Search obteve um dos melhores desempenhos neste caso, com um baixo número de iterações e o menor tempo total de execução dentre os métodos aplicados.

3 Questão 4.4

A função

$$f(x) = -3\sin(0.75x) + \exp^{-2x} \tag{3}$$

é unimodal para o intervalo $[0, 2\pi]$. Repita a Questão 4.2 para função (3).

Na figura 3 temos o gráfico da função (3) para o intervalo $[0, 2\pi]$. Vemos que o valor mínimo da função encontra-se próximo a -7.0 para um valor de x de aproximadamente 2.8.

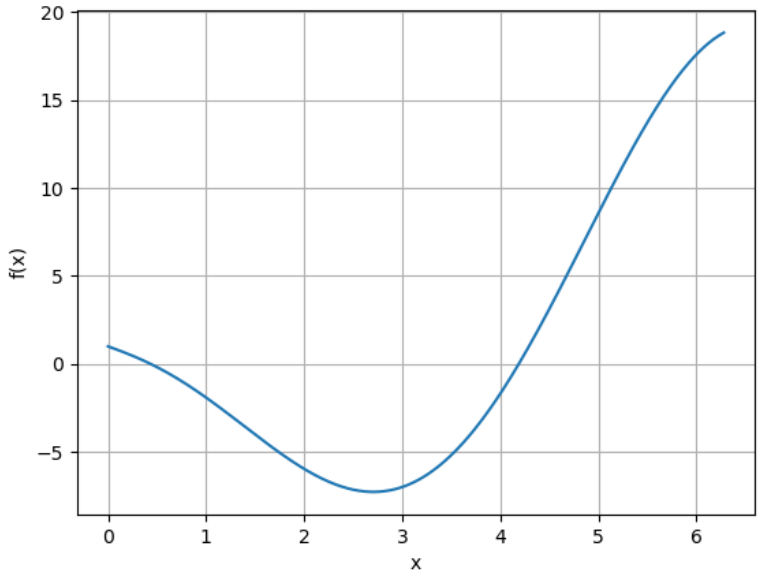

Figura 3: Gráfico da função (3) para o intervalo $[0, 2\pi]$.

3.1 Itens (a)-(g)

Repetindo a Questão 4.2, nos itens (a) - (f) é pedido para minimizarmos função f(x) (3) utilizando os seguintes algoritmos:

(a) Busca Dicotômica

(b) Busca de Fibonacci

(c) Busca na Seção Dourada

(d) Método de Interpolação Quadrática

(e) Método de Interpolação Cúbica

(f) Algoritmo de Davies, Swann, e Campey

(g) Backtracking Line Search

para o intervalo $[0, 2\pi]$ com um intervalo de incerteza menor que 10^{-5}.

No trecho de código abaixo temos o resultado obtido ao executarmos cada um dos algoritmos de busca para o problema em questão.

Listing 3: Resposta do script 'lista01.py' quando executado para a Questão 4.4.

```
$ python lista01.py -exe_num 4.4 -min 0 -max 6.283185307179586
Exercise 4.4:

Dichotomous Search
x: 2.70648, f(x): -7.27436, num_iter: 20, time: 0.06104 ms

Fibonacci Search
x: 2.70647, f(x): -7.27436, num_iter: 28, time: 0.07796 ms

Golden Section Search
x: 2.70648, f(x): -7.27436, num_iter: 28, time: 0.03791 ms

Quadratic Interpolation Search
x: 2.70648, f(x): -7.27436, num_iter: 9, time: 0.03099 ms

Cubic Interpolation Search
x: 3.14594, f(x): -6.64991, num_iter: 6, time: 0.15712 ms, for [0, 3.14159,
   6.28318]
x: 3.03849, f(x): -6.92198, num_iter: 24, time: 0.39887 ms, for [0, 1.0,
   3.14159]
x: 2.69977, f(x): -7.27422, num_iter: 4, time: 0.07105 ms, for [1, 2, 3]
x: 2.72590, f(x): -7.27320, num_iter: 5, time: 0.08392 ms, for [2, 3, 4]

Davies, Swann and Campey Search
x: 2.70648, f(x): -7.27436, num_iter: 8, time: 0.07200 ms, for 2
x: 2.70648, f(x): -7.27436, num_iter: 6, time: 0.04005 ms, for 3.14159
x: 2.70648, f(x): -7.27436, num_iter: 8, time: 0.07892 ms, for 6.28318

Backtraking Line Search
```

```
x: 2.70648, f(x): -7.27436, num_iter: 18, time: 0.32306 ms, for 0
x: 2.70648, f(x): -7.27436, num_iter: 16, time: 0.26202 ms, for 2
x: 2.70647, f(x): -7.27436, num_iter: 16, time: 0.26703 ms, for 3.14159
x: 2.70648, f(x): -7.27436, num_iter: 19, time: 0.30303 ms, for 6.28318
admins-MacBook-Air:lista01 admin$
```

3.2 Item (h)

Comparar a eficiência computacional dos métodos (a)-(g) em termos do número de avaliações de funções.

Na Tabela 3 temos a comparação dos métodos de busca aplicados nos itens (a)-(g) para o valor mínimo da função encontrado, número de iterações do algoritmo, tempo total de execução e tempo médio por iteração em microsegundos. Para os métodos cujos resultados variam dependendo dos pontos de entrada ((e)-(g)) é considerado a média dos valores encontrados.

Método	Mínimo	# iterações	Tempo total de execução / média por iteração (μs)
(a) Busca Dicotômica	-7.27436	20	61.04 / 3.05
(b) Busca de Fibonacci	-7.27436	28	77.96 / 2.78
(c) Busca na Seção Dourada	-7.27436	28	37.91 / 1.35
(d) Método de Interpolação Quadrática	-7.27436	9	30.99 / 2.86
(e) Método de Interpolação Cúbica	-7.02983	9.75	17.77 / 1.82
(f) Algoritmo de Davies, Swann, e Campey	-7.27436	7.3	63.66 / 8.72
(g) Backtracking Line Search	-7.27436	17.25	288.79 / 16.74

Tabela 3: Comparação dos métodos de busca aplicados nos itens (a)-(g) para a função (3).

As conclusões sobre o número de avaliações da função para cada método, se mantiveram de acordo com a Questão 4.2, com as devidas proporções.

Os métodos ((a)-(c)) tiveram um desempenho semelhante ao observado na questões anteriores.

Novamente, podemos observar como o resultado obtido pelo Método de Interpolação Cúbica pode variar dependendo dos valores escolhidos para inicialização.

4 Questão 4.11

(a) Use o *MATLAB* para plotar

$$f(x) = 0.7x_1^4 - 8x_1^2 + 6x_2^2 + \cos(x_1 x_2) - 8x_1 \tag{4}$$

na região $-\pi \leq x_1, x_2 \leq \pi$.

Na Figura 4 temos o gráfico da função (4) obtido utilizando a função *meshgrid*. Vemos que a função apresenta dois mínimos locais para este intervalo, sendo um deles global.

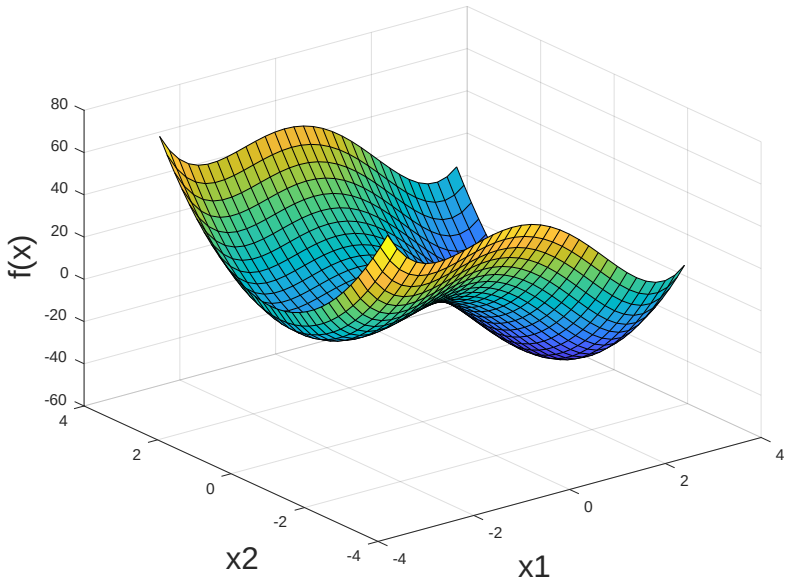

Figura 4: Gráfico da função (4).

(b) Use o *MATLAB* para gerar o contorno de f(x) (4) na mesma região.
A Figura 5 apresenta o contorno da função (4) para a região $-\pi \leq x_1, x_2 \leq \pi$.

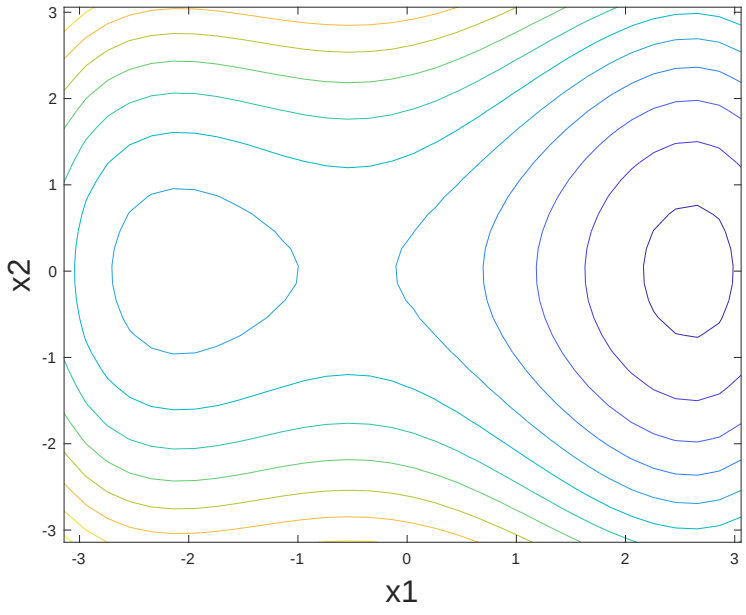

Figura 5: Contorno da função (4).

(c) Calcular o gradiente de f(x) e preparar o *MATLAB* para executar a função f(x).
As funções foram implementadas no script *exercise_4_11.m*, em anexo.

(d) Utilizar a Busca em Linha Inexata de Fletcher para atualizar o ponto x_0 ao longo da direção de busca d_0 para o problema:

Minimizar $f(x_0 + \alpha d_0), \alpha \geq 0$
Onde $x_0 = [-\pi, \pi]^T; d_0 = [1.0, -1.3]^T$

- Guardar os valores de α^* obtidos:

$$\alpha^* = [0.01613, 0.16134, 0.75264]$$

- Guardar os pontos $x_1 = x_0 + \alpha^* d_0$ obtidos:

$$x_1 = \begin{bmatrix} -3.12545 & -2.98025 & -2.38894 \\ 3.12061 & 2.93184 & 2.16314 \end{bmatrix}$$

- Avaliar $f(x_0)$ com $f(x_1)$:

$$f(x_1) = \begin{bmatrix} 71.13518 & 58.80988 & 24.76943 \end{bmatrix}$$

- Plotar o resultado da busca de linha no contorno da função

  Na Figura 6 podemos ver os três resultados encontrados para x_1 plotados sobre o gráfico do contorno de f(x).

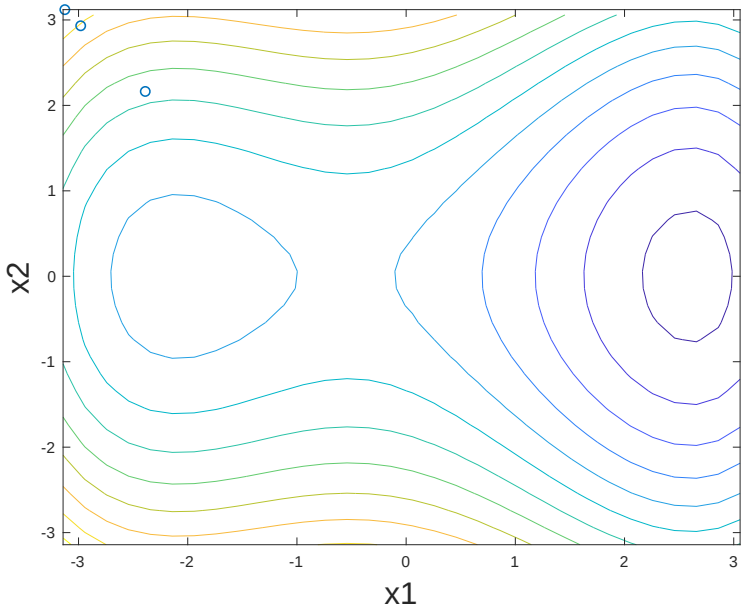

Figura 6: Pontos x_1 encontrados sob o contorno da função (4).

Se executarmos o algoritmo novamente a partir do último valor de x_1 encontrado é possível melhorar o resultado final para a direção de busca d_0. Repetindo-o

enquanto $f(x_1) < f(x_0)$ obtemos os resultados exposto na Figura 7, cujos últimos valores encontrados foram:

$$\alpha^* = 0.19921$$
$$x_1 = \begin{bmatrix} -1.02185 \\ 0.38593 \end{bmatrix}$$
$$f(x_1) = 2.40147$$

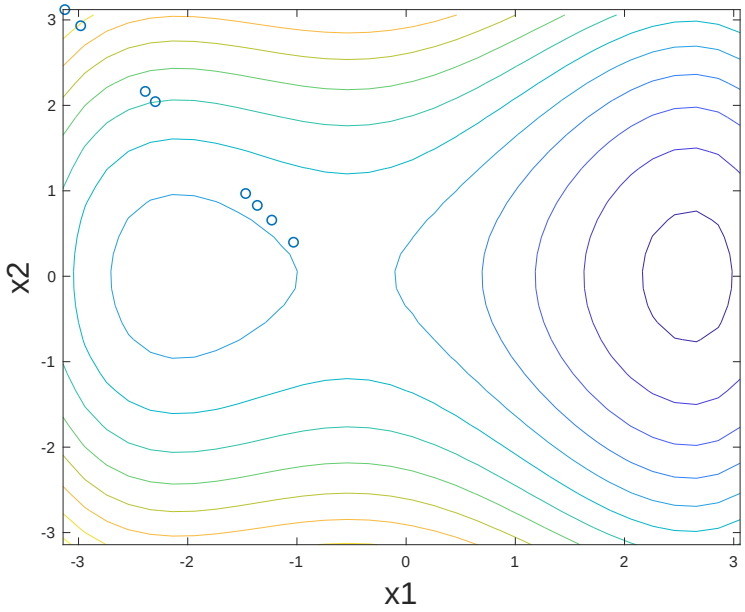

Figura 7: Pontos x_1 encontrados sob o contorno da função (4) repetindo-se a execução do algoritmo a partir do último ponto encontrado.

- Plotar $f(x_0 + \alpha d_0)$ como uma função de α no intervalo $[0, 4.8332]$.
  Na Figura 8 temos $f(x_0 + \alpha d_0)$ como uma função de α no intervalo $[0, 4.8332]$
- Repetir o passo (d) para a nova direção:

$$d_0 = [1.0, -1.1]^T$$

  - Guardar os valores de α^* obtidos:

$$\alpha^* = [0.01613, 0.16134, 0.76659]$$

  - Guardar os pontos $x_1 = x_0 + \alpha^* d_0$ obtidos:

$$x_1 = \begin{bmatrix} -3.12545 & -2.98025 & -2.37499 \\ 3.12384 & 2.96411 & 2.29833 \end{bmatrix}$$

  - Avaliar $f(x_0)$ com $f(x_1)$:

$$f(x_1) = \begin{bmatrix} 71.13518 & 59.89406 & 28.51955 \end{bmatrix}$$

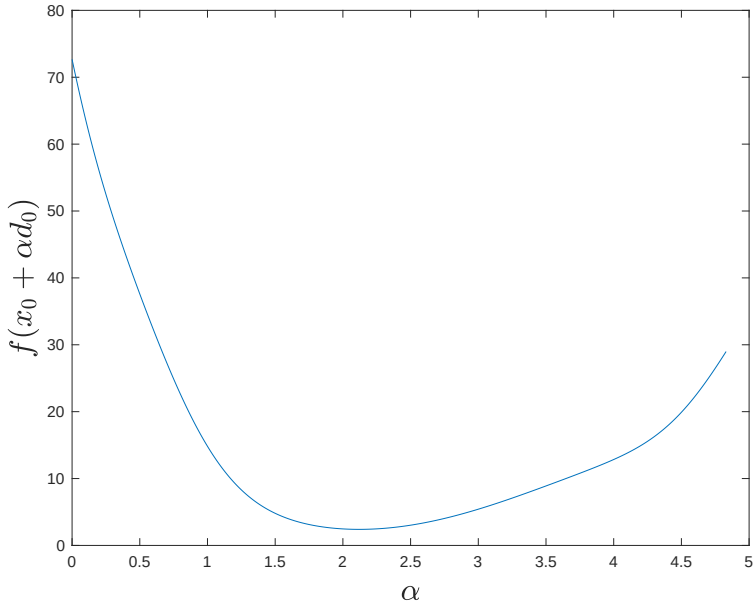

Figura 8: $f(x_0 + \alpha d_0)$ no intervalo $[0, 4.8332]$

– Plotar o resultado da busca de linha no contorno da função
Na Figura 9 podemos ver os três resultados encontrados para x_1 plotados sobre o gráfico do contorno de f(x).

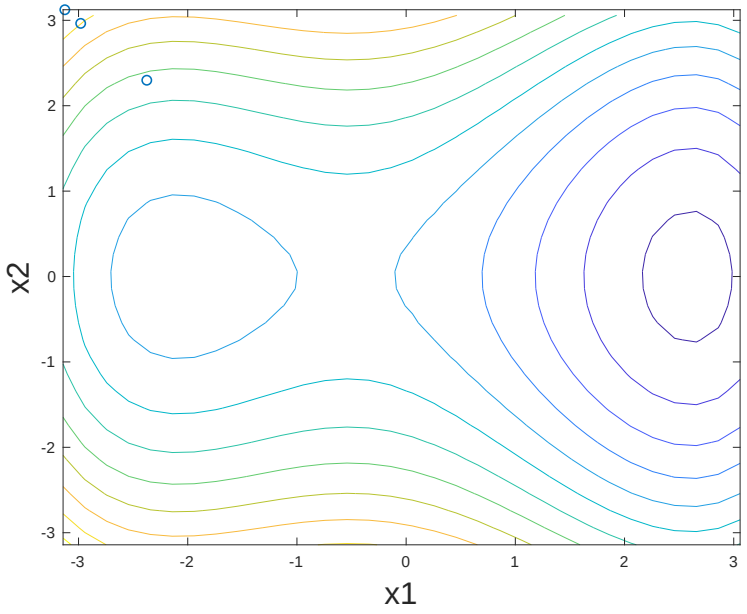

Figura 9: Pontos x_1 encontrados sob o contorno da função (4) para a nova direção de busca.

Se executarmos o algoritmo novamente a partir do último valor de x_1 encontrado é possível melhorar o resultado final para a direção de busca d_0. Repetindo-o enquanto $f(x_1) < f(x_0)$ obtemos os resultados exposto na Figura 10, cujos últimos valores encontrados foram:

$$\alpha^* = 1.16764$$

$$x_1 = \begin{bmatrix} -1.20735 \\ 1.01393 \end{bmatrix}$$

$$f(x_1) = 5.99268$$

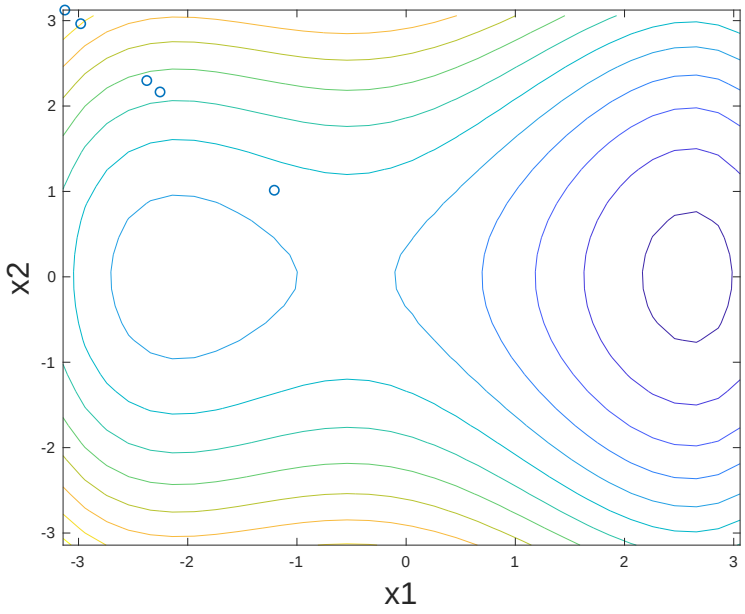

Figura 10: Pontos x_1 encontrados sob o contorno da função (4) repetindo-se a execução do algoritmo a partir do último ponto encontrado.

- Plotar $f(x_0 + \alpha d_0)$ como uma função de α no intervalo $[0, 5.7120]$.
  Na Figura 11 temos $f(x_0 + \alpha d_0)$ como uma função de α no intervalo $[0, 5.7120]$

Comparando as duas direções de busca vemos que estas são muito semelhantes, sendo a primeira um pouco superior.

Todos os códigos utilizados neste trabalho estão em anexo e também podem ser acessados em *https://github.com/patykov/OtmConvexa*.

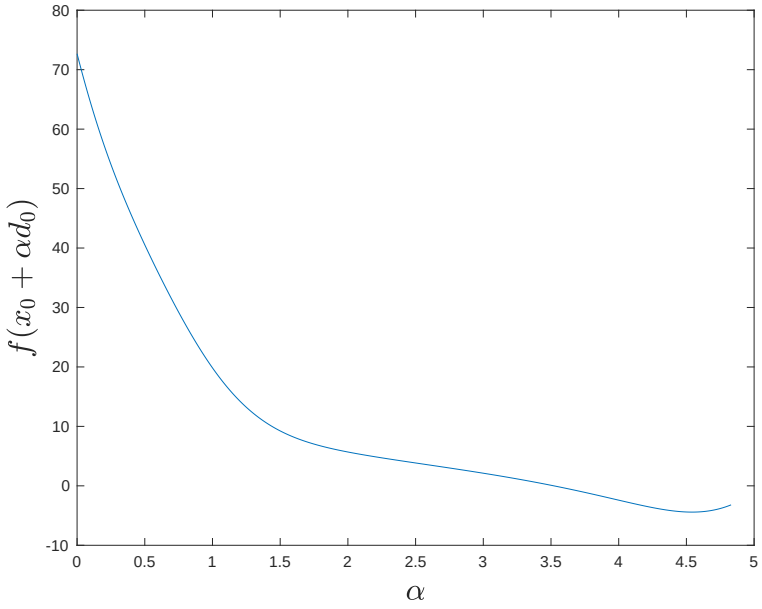

Figura 11: $f(x_0 + \alpha d_0)$ no intervalo $[0, 5.7120]$