Lista 02

Otimização Convexa

Patrícia Kovaleski

7 de novembro de 2018

Exercício do livro-texto, *Boyd & Vandenberghe*:

1 Questão 9.4

Consideremos primeiro o problema de minimizar a função:

$$f(x,y) = \sum_{i=1}^{m} \frac{x_i^2}{y_1} + 1^T y \tag{1}$$

para $y \succ 0$, que equivale a calcular:

$$\inf_{y \succ 0} \big(f(x,y)\big)$$

.

Obtendo sua derivada e igualando-a a zero, temos:

$$\begin{aligned} \frac{\partial f(x,y)}{\partial y} &= \sum_{i=1}^{m} -\frac{x_i^2}{y_i^2} + 1^T = 0 \\ y_i^2 &= x_i^2 \\ y_i &= |x_i| \end{aligned}$$

uma vez que y precisa ser positivo.

Substituindo em (1):

$$\begin{aligned} f(x,y) &= \sum_{i=1}^{m} \frac{x_i^2}{|x_i|} + \sum_{i=1}^{m} |x_i| \\ &= \sum_{i=1}^{m} |x_i| + \sum_{i=1}^{m} |x_i| \\ &= ||x||_1 + ||x||_1 \\ &= 2||x||_1 \end{aligned}$$

Assim, substituindo a função 1 na equivalência proposta na Questão 9.4, temos:

$$
\begin{aligned}
||x||_1 &= \frac{1}{2} \inf_{y \succ 0} \Big(\sum_{i=1}^{m} \frac{x_i^2}{y_1} + 1^T y \Big) \\
||x||_1 &= \frac{1}{2} \inf_{y \succ 0} \big(f(x,y) \big) \\
||x||_1 &= \frac{1}{2} \times 2||x||_1 \\
||x||_1 &= ||x||_1
\end{aligned}
\tag{2}
$$

comprovando-a.

A substituição $||z||_1 = ||Ax + b||_1$ consiste em uma transformação afim e não altera sua convexidade. Assim, sendo $f(x, y)$ uma função duplamente derivável, o problema de minimizar $||Ax+b||_1$ pode ser resolvido pelo método de *Newton* ao aplicarmos a equivalência proposta.

Exercícios do livro *Practical Optimization de Antoniou & Lu*:

2 Questão 5.7

Neste exercício é pedido para resolvermos a equação:

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

aplicando o algoritmo *Steepest Descent* para os pontos iniciais: $[4, 4]^T, [4, -4]^T, [-4, 4]^T, [-4, -4]^T$; utilizando $\epsilon = 10^{-6}$ e analisarmos as soluções encontradas para cada ponto.

Nas Figuras 1 e 2 temos o gráfico da função (3) para diferentes intervalos, de forma a visualizarmos melhor sua variação. Dois pontos de mínimo são identificados, em $[1.0, 0.0]$ e $[0.0, 1.0]$, para os quais a função atinge o valor zero.

É preciso também definirmos o algoritmo de busca em linha que será aplicado durante as iterações do *Steepest Descent*. Baseando-se nos resultados obtidos na lista de exercícios anterior (Lista01) foram escolhidos os seguintes algoritmos:

1. *Golden Section Search*
2. *Quadratic Interpolation Search*
3. *Backtraking Line Search*

devido ao bom desempenho alcançado por eles e por apresentarem abordagens bastante distintas para a solução do problema de busca em linha. No trecho de código abaixo temos o resultado obtido ao executarmos o algoritmo de *Steepest Descent* para cada um dos métodos de busca selecionados para o problema em questão.

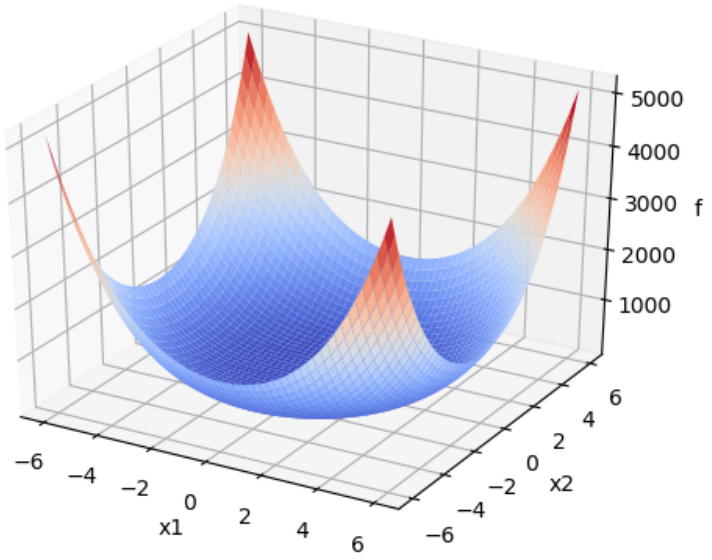

Figura 1: Gráfico da função (3) para o intervalo $x_1, x_2 \in [-6.0, 6.0]$.

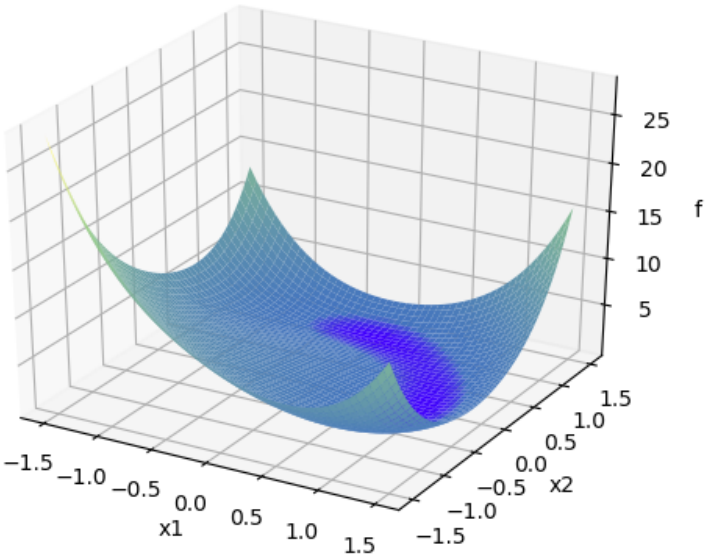

Figura 2: Gráfico da função (3) para o intervalo $x_1, x_2 \in [-1.5, 1.5]$.

Listing 1: Resposta do script 'lista02.py' quando executado para a Questão 5.7.

```
python lista02.py -exe_num 5.7 -min -10 -max 10
Exercise 5.7 - Steepest Descent

Initial point: [4.0, 4.0]
Golden Section Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 3, time: 0.97609 ms

Quadratic Interpolation Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 4, time: 1.29485 ms

Backtraking Line Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 36, time: 25.46692 ms


Initial point: [4.0, -4.0]
Golden Section Search
x: [1.00000, -0.00000], f(x): 0.00000, num_iter: 36, time: 6.89292 ms

Quadratic Interpolation Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 22, time: 7.12204 ms

Backtraking Line Search
x: [1.00000, -0.00000], f(x): 0.00000, num_iter: 26, time: 26.68309 ms


Initial point: [-4.0, 4.0]
Golden Section Search
x: [-0.00000, 1.00000], f(x): 0.00000, num_iter: 36, time: 9.41801 ms

Quadratic Interpolation Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 22, time: 9.12189 ms

Backtraking Line Search
x: [-0.00000, 1.00000], f(x): 0.00000, num_iter: 26, time: 29.35100 ms


Initial point: [-4.0, -4.0]
Golden Section Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 3, time: 1.31202 ms

Quadratic Interpolation Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 4, time: 2.23303 ms
```

```
Backtraking Line Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 7, time: 8.46219 ms
```

A Tabela 1 consolida os resultados obtidos para os diferentes métodos de busca aplicados a partir de cada ponto inicial. São comparados o ponto e o valor mínimo da função encontrados, número de iterações do algoritmo e tempo total de execução em milissegundos. Podemos observar que na maior parte das vezes o algoritmo ficou preso no ponto de cela entre os dois mínimos, incapaz de se libertar.

Ponto inicial	Busca em linha	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
$[4, 4]^T$	Busca na Seção Dourada	$[0.63, 0.63]$	0.11	3	0.976
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	4	1.295
	Backtracking Line Search	$[0.63, 0.63]$	0.11	36	25.467
$[4, -4]^T$	Busca na Seção Dourada	$[1.0, 0.0]$	0.0	36	6.893
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	22	7.122
	Backtracking Line Search	$[1.0, 0.0]$	0.0	26	26.683
$[-4, 4]^T$	Busca na Seção Dourada	$[0.0, 1.0]$	0.0	36	9.418
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	22	9.122
	Backtracking Line Search	$[0.0, 1.0]$	0.0	26	29.351
$[-4, -4]^T$	Busca na Seção Dourada	$[0.63, 0.63]$	0.11	3	1.312
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	4	2.233
	Backtracking Line Search	$[0.63, 0.63]$	0.11	7	8.462

Tabela 1: Comparação dos resultados obtidos para cada método de busca em linha aplicado no algoritmo *Steepest Descent* para diferentes pontos iniciais.

Pode-se dizer que os métodos de Busca na Seção Dourada e Backtracking Line Search tiveram um desempenho melhor, pois alcançaram um ponto mínimo quando executados a partir dos pontos iniciais $[4, -4]^T$ e $[-4, 4]^T$. Para este caso, a Busca na Seção Dourada mostrou-se a mais eficiente, pois alcançou o ponto mínimo o maior número de vezes com o menor tempo total de execução.

3 Questão 5.8

Já neste exercício é pedido para resolvermos novamente o problema proposto na Questão 5.7, porém desta vez utilizando um algoritmo de *Steepest Descent* que não utiliza um método de busca local durante suas iterações, mas sim uma solução analítica. Temos no trecho de código abaixo o resultado obtido ao o executarmos o algoritmo de *Steepest Descent* sem busca local para cada um dos pontos iniciais.

Listing 2: Resposta do script 'lista02.py' quando executado para a Questão 5.8.

```
python lista02.py -exe_num 5.8
Exercise 5.8 - Steepest Descent without line search
```

```
Initial point: [4.0, 4.0]
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 9, time: 0.49901 ms


Initial point: [4.0, -4.0]
x: [1.00000, -0.00000], f(x): 0.00000, num_iter: 44, time: 1.82700 ms


Initial point: [-4.0, 4.0]
x: [-0.00000, 1.00000], f(x): 0.00000, num_iter: 44, time: 1.54686 ms


Initial point: [-4.0, -4.0]
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 16, time: 0.58103 ms
```

Na Tabela 2 temos a comparação dos resultados obtidos para o algoritmo *Steepest Descent* sem busca em linha para cada ponto inicial. Novamente, apenas para os pontos $[4, -4]^T$ e $[-4, 4]^T$ o algoritmo foi capaz de encontrar um ponto mínimo; ficando preso no ponto de sela quando executado a partir dos demais pontos iniciais. Em relação aos resultados obtidos na questão anterior, expostos na Tabela 1, vemos que não utilizar um algoritmo de busca em linha se mostrou vantajoso pois os resultados obtidos foram semelhantes e com um custo computacional muito menor.

O menor custo computacional pode ser explicado pelo fato de que em cada iteração do algoritmo *Steepest Descent* sem busca em linha apenas um α é calculado utilizando duas avaliações da função e uma do gradiente. Já no *Steepest Descent* com busca em linha, em cada iteração a busca em linha pode calcular diversos α, avaliando a função diversas vezes e resultando em um maior custo computacional.

Ponto inicial	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
$[4, 4]^T$	$[0.63, 0.63]$	0.11	9	0.499
$[4, -4]^T$	$[1.0, 0.0]$	0.0	44	1.827
$[-4, 4]^T$	$[0.0, 1.0]$	0.0	44	1.547
$[-4, -4]^T$	$[0.63, 0.63]$	0.11	16	0.581

Tabela 2: Comparação dos resultados obtidos para o algoritmo *Steepest Descent* sem busca em linha para diferentes pontos iniciais.

4 Questão 5.17

Nesta questão é pedido para resolvermos o problema proposto na Questão 5.7, porém desta vez utilizando o algoritmo de *Basic Newton* modificado. Temos no trecho de código a seguir

os resultados obtidos para o algoritmo quando executado a partir de diferentes pontos iniciais e utilizando diferentes buscas em linha.

Listing 3: Resposta do script 'lista02.py' quando executado para a Questão 5.17.

```
python lista02.py -exe_num 5.17 -min -10 -max 10
Exercise 5.17 - Modified Newton

Initial point: [4.0, 4.0]
Golden Section Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 1, time: 1.29700 ms

Quadratic Interpolation Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 2, time: 1.25790 ms

Backtraking Line Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 5, time: 3.79777 ms


Initial point: [4.0, -4.0]
Golden Section Search
x: [1.00000, 0.00000], f(x): 0.00000, num_iter: 9, time: 2.57897 ms

Quadratic Interpolation Search
x: [1.00000, 0.00000], f(x): 0.00000, num_iter: 9, time: 1.62697 ms

Backtraking Line Search
x: [0.00000, 1.00000], f(x): 0.00000, num_iter: 10, time: 8.83102 ms


Initial point: [-4.0, 4.0]
Golden Section Search
x: [0.00000, 1.00000], f(x): 0.00000, num_iter: 9, time: 2.82192 ms

Quadratic Interpolation Search
x: [0.00000, 1.00000], f(x): 0.00000, num_iter: 9, time: 1.87707 ms

Backtraking Line Search
x: [1.00000, 0.00000], f(x): 0.00000, num_iter: 10, time: 9.83596 ms


Initial point: [-4.0, -4.0]
Golden Section Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 1, time: 1.17302 ms

Quadratic Interpolation Search
```

```
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 3, time: 1.40405 ms

Backtraking Line Search
x: [0.62996, 0.62996], f(x): 0.11012, num_iter: 6, time: 7.88879 ms
```

A Tabela 3 consolida os resultados obtidos para os diferentes métodos de busca aplicados a partir de cada ponto inicial. Novamente, o mínimo só foi alcançado quando o algoritmo é executado a partir dos pontos $[4, -4]^T$ e $[-4, 4]^T$. Porém, diferente da Questão 5.7, o mínimo foi encontrado por todos os métodos de busca em linha para os pontos iniciais mencionados. Além disso, o número de iterações necessárias e tempo total de processamento também foi bastante menor, evidenciando a maior eficiência do algoritmo de *Newton* modificado.

Ponto inicial	Busca em linha	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
$[4, 4]^T$	Busca na Seção Dourada	$[0.63, 0.63]$	0.11	1	1.297
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	2	1.258
	Backtracking Line Search	$[0.63, 0.63]$	0.11	5	3.798
$[4, -4]^T$	Busca na Seção Dourada	$[1.0, 0.0]$	0.0	9	2.579
	Interpolação Quadrática	$[1.0, 0.0]$	0.0	9	1.627
	Backtracking Line Search	$[0.0, 1.0]$	0.0	10	8.831
$[-4, 4]^T$	Busca na Seção Dourada	$[0.0, 1.0]$	0.0	9	2.822
	Interpolação Quadrática	$[0.0, 1.0]$	0.0	9	1.877
	Backtracking Line Search	$[1.0, 0.0]$	0.0	10	9.836
$[-4, -4]^T$	Busca na Seção Dourada	$[0.63, 0.63]$	0.11	1	1.173
	Interpolação Quadrática	$[0.63, 0.63]$	0.11	3	1.404
	Backtracking Line Search	$[0.63, 0.63]$	0.11	6	7.889

Tabela 3: Comparação dos resultados obtidos para cada método de busca em linha aplicado no algoritmo *Basic Newton* modificado para diferentes pontos iniciais.

5 Questão 5.20

5.1 Item (a)

Encontre o minimizador global da função objetivo:

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 100(x_1 - x_4)^4 \tag{4}$$

utilizando o fato de que cada termo da função objetivo é não-negativo.

O valor mínimo que a função pode obter, dada as considerações do enunciado da questão, é zero, pois esta seria formada de uma soma de termos não-negativos. Para atingir este valor precisamos que cada um dos quatros termos que compõem a função também sejam zero. Isto nos leva a uma solução única, $\mathbf{x} = [0, 0, 0, 0]^T$ com $f(\mathbf{x}) = 0$.

5.2 Itens (b) - (e)

(b) Resolva o problema do Item (a) utilizando o algoritmo *Steepest Descent*

(c) Resolva o problema do Item (a) utilizando o algoritmo de *Newton* modificado

(d) Resolva o problema do Item (a) utilizando o algoritmo *Gauss-Newton*

(e) Baseado nos resultados obtidos em (b)-(d), compare a eficiência computacional e acurácia da solução encontrada pelos três métodos

Assim como nas Questões anteriores, a minimização da função (4) foi realizada utilizando os algoritmos especificados nos Itens (b)-(d) com os três métodos de busca escolhidos anteriormente. No trecho de código abaixo temos o resultado obtido para cada algoritmo considerando os diferentes métodos de busca e pontos iniciais pedidos.

Listing 4: Resposta do script 'lista02.py' quando executado para a Questão 5.20.

```
python lista02.py -exe_num 5.20 -min -10 -max 10
Exercise 5.20

Initial point: [-2.0, -1.0, 1.0, 2.0]
---- Steepest Descent -----
Golden Section Search
x: [0.01686, -0.00169, 0.01183, 0.01183], f(x): 4.76085e-07, num_iter: 9882,
    time: 2697.41702 ms

Quadratic Interpolation Search
x: [0.01607, -0.00161, 0.01127, 0.01128], f(x): 3.92997e-07, num_iter:
   10613, time: 753.47090 ms

Backtraking Line Search
x: [0.01685, -0.00168, 0.01182, 0.01183], f(x): 4.75364e-07, num_iter: 9880,
    time: 13671.07081 ms



---- Modified Newton -----
Golden Section Search
x: [-0.00000, 0.00000, 0.00000, 0.00000], f(x): 1.19104e-21, num_iter: 23,
   time: 7.73907 ms

Quadratic Interpolation Search
x: [-0.00000, 0.00000, 0.00000, 0.00000], f(x): 3.64261e-22, num_iter: 23,
   time: 4.78196 ms

Backtraking Line Search
```

```
x: [-0.00000, 0.00000, 0.00000, 0.00000], f(x): 2.80440e-21, num_iter: 22,
   time: 17.78698 ms


---- Gauss Newton -----
Golden Section Search
x: [-0.00147, 0.00015, 0.00100, 0.00100], f(x): 5.99957e-09, num_iter: 7,
   time: 5.48697 ms

Quadratic Interpolation Search
x: [-0.00146, 0.00015, 0.00100, 0.00099], f(x): 5.88314e-09, num_iter: 7,
   time: 2.33507 ms

Backtraking Line Search
x: [-0.00147, 0.00015, 0.00100, 0.00100], f(x): 5.99938e-09, num_iter: 7,
   time: 5.81789 ms


Initial point: [200.0, -200.0, 100.0, -100.0]
---- Steepest Descent -----
Golden Section Search
x: [0.01770, -0.00177, 0.01242, 0.01243], f(x): 5.79064e-07, num_iter:
   10778, time: 2785.10690 ms

Quadratic Interpolation Search
x: [0.01381, -0.00138, 0.00969, 0.00969], f(x): 2.14255e-07, num_iter: 9514,
    time: 715.60502 ms

Backtraking Line Search
x: [0.01697, -0.00170, 0.01190, 0.01191], f(x): 4.88528e-07, num_iter: 9244,
    time: 11917.93203 ms


---- Modified Newton -----
Golden Section Search
x: [0.00000, -0.00000, 0.00000, 0.00000], f(x): 9.86990e-24, num_iter: 27,
   time: 8.58998 ms

Quadratic Interpolation Search
x: [0.00000, -0.00000, 0.00000, 0.00000], f(x): 1.08064e-23, num_iter: 26,
   time: 5.60498 ms

Backtraking Line Search
x: [0.00000, -0.00000, 0.00000, 0.00000], f(x): 2.11327e-23, num_iter: 26,
   time: 23.51689 ms
```

```
---- Gauss Newton -----
Golden Section Search
x: [0.00456, -0.00045, 0.00168, 0.00169], f(x): 9.63513e-09, num_iter: 9,
   time: 3.65186 ms

Quadratic Interpolation Search
x: [0.00418, -0.00041, 0.00154, 0.00155], f(x): 6.95765e-09, num_iter: 9,
   time: 2.35009 ms

Backtraking Line Search
x: [0.00530, -0.00053, 0.00196, 0.00195], f(x): 1.57986e-08, num_iter: 8,
   time: 7.87091 ms
```

Nas Tabelas 4, 5 e 6 temos os resultados obtidos para os algoritmos *Steepest Descent*, *Newton* modificado e *Gauss Newton*, respectivamente. Todos os algoritmos alcançaram bons resultados, sendo o *Newton* modificado aquele que obteve a maior precisão.

O *Steepest Descent* é o método mais simples dentre os três, no qual é preciso realizar avaliações da função e do gradiente. Devido a isto, também necessitou o maior número de iterações, cerca de 10000.

Já o algoritmo de *Newton* modificado avalia a função, gradiente, hessiana, calcula matrizes inversas e autovalores. Possui um custo computacional razoavelmente alto, porém, consegue alcançar o ponto mínimo em cerca de 25 iterações, resultando em um baixo tempo total de processamento.

Por fim, temos o algoritmo de *Gauss Newton*, que apresenta a maior complexidade computacional dentre os três. Nele é necessário avaliar a função, jacobiano, matrix inversa, produto e decomposição de matrizes. Ele também foi o método que necessitou o menor número de iterações, aproximadamente 8, e menor tempo total de processamento.

Ponto inicial	Busca em linha	$f(x_{min})$	# iter	Tempo total (ms)
$[-2, -1, 1, 2]$	Seção Dourada	$4.761e^{-07}$	9882	2697.417
	Interp. Quadrática	$3.930e^{-07}$	10613	753.471
	Backtracking	$4.754e-^{07}$	9880	13671.071
$[200, -200, 100, -100]$	Seção Dourada	$5.791e^{-07}$	10778	2785.107
	Interp. Quadrática	$2.143e^{-07}$	9514	715.605
	Backtracking	$4.885e^{-07}$	9244	11917.932

Tabela 4: Comparação dos resultados obtidos para cada método de busca em linha aplicado no algoritmo *Steepest Descent* para diferentes pontos iniciais.

Ponto inicial	Busca em linha	$f(x_{min})$	# iter	Tempo total (ms)
$[-2, -1, 1, 2]$	Seção Dourada	$1.191e^{-21}$	23	7.739
	Interp. Quadrática	$3.643e^{-22}$	23	4.782
	Backtracking	$2.804e^{-21}$	22	17.787
$[200, -200, 100, -100]$	Seção Dourada	$9.870e^{-24}$	27	8.590
	Interp. Quadrática	$1.081e^{-23}$	26	5.605
	Backtracking	$2.113e^{-23}$	26	23.517

Tabela 5: Comparação dos resultados obtidos para cada método de busca em linha aplicado no algoritmo *Newton* modificado para diferentes pontos iniciais.

Ponto inicial	Busca em linha	$f(x_{min})$	# iter	Tempo total (ms)
$[-2, -1, 1, 2]$	Seção Dourada	$5.999e^{-09}$	7	5.487
	Interp. Quadrática	$5.883e^{-09}$	7	2.335
	Backtracking	$5.999e^{-09}$	7	5.818
$[200, -200, 100, -100]$	Seção Dourada	$9.635e^{-09}$	9	3.652
	Interp. Quadrática	$6.958e^{-09}$	9	2.350
	Backtracking	$1.580e^{-08}$	8	7.871

Tabela 6: Comparação dos resultados obtidos para cada método de busca em linha aplicado no algoritmo *Gauss Newton* para diferentes pontos iniciais.

6 Considerações finais

Durante a execução das Questões acima algumas características dos métodos de busca em linha escolhidos foram observados. A Busca na Seção Dourada e a Interpolação Quadrática dependem de parâmetros arbitrários como o espaço de busca inicial. Este fator tornou-se especialmente complicado para o caso da Questão 5.20 quando executada a partir do ponto $[200, -200, 100, -100]^T$. Nela, determinar valores iniciais razoáveis é muito difícil, uma vez que a função possui múltiplas variáveis, prejudicando sua visualização. Dessa forma foi necessário realizar diversos testes até se obter resultados satisfatórios para estes métodos de busca, o que não é um ponto desejável.

Já sobre os algoritmos multidimensionais explorados, o *Newton* modificado foi o mais eficiente em todos os casos. Ele foi capaz de alcançar o ponto mínimo com uma precisão altamente superior a dos demais métodos - cerca de doze casas decimais acima. Sua eficiência foi tamanha que conseguiu inclusive abstrair o problema de sensibilidade aos parâmetros de busca inicial dos métodos da Seção Dourada e da Interpolação Quadrática. Para o caso problemático da Questão 5.20 mencionado acima, o algoritmo de *Newton* modificado foi capaz de atingir o mínimo perfeitamente, mantendo seu resultado consistente durante os diversos testes e alterações efetuadas para que os demais algoritmos convergissem.

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