Lista 04

Otimização Convexa

Patrícia Kovaleski

6 de dezembro de 2018

1 Questão 1

Para o problema em 11.16 (*Antoniou*), encontre uma solução utilizando pontos interiores e:

(a) Gradiente descendente;

(b) Gradiente conjugado;

(c) Quase-Newton;

(d) Newton.

Problema 11.16:

$$\begin{aligned} \text{minimize} \quad & f(x) = x_1 + 1.5x_2 + x_3 + x_4 && (1)\\ \text{sujeito a:} \quad & x_1 + 2x_2 + x_3 + 2x_4 = 3 && (2)\\ & x_1 + x_2 + 2x_3 + 4x_4 = 5 && (3)\\ & x_i \geq 0, \quad i = 1,2,3,4 && (4) \end{aligned}$$

Para resolvê-lo precisamos primeiro eliminar as restrições de igualdade (2) e (3). Começamos definindo a matriz $\mathbf{A}$ e os vetores $\mathbf{b}$ e $\mathbf{x}$, que representam estas restrições da forma:

$$\mathbf{Ax} = \mathbf{b}$$

$$\begin{bmatrix} 1 & 2 & 1 & 2 \\ 1 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

Sendo f convexa e duplamente diferenciável, estas restrições podem ser eliminadas ao reescrevermos $\mathbf{x}$ da forma:

$$\mathbf{x} = \mathbf{Fz} + \bar{\mathbf{x}}$$

sendo $\mathbf{F}$ a matriz espaço-nulo de $\mathbf{A}$, $\mathbf{z}$ a nova variável a ser minimizada e $\bar{\mathbf{x}}$ uma solução particular para o problema original - que atende as restrições de igualdade. Assim, o problema 11.16 pode ser reescrito como:

$$\text{minimize} \quad f(\mathbf{Fz} + \bar{\mathbf{x}}) \tag{5}$$

$$\text{sujeito a:} \quad \mathbf{F(i,\ :)z} + \bar{\mathbf{x}}(i) \geq 0, \quad i = 1,2,3,4 \tag{6}$$

Em seguida, precisamos eliminar as novas restrições de desigualdade expostas em (6). Para tal, aplicamos o método da barreira logarítmica, no qual inserimos uma função indicadora com o intuito de aproximar o efeito destas restrições. A aproximação é feita pela função logarítmo e obtemos ao final um problema sem restrições da forma:

$$\text{minimize} \quad t\mathbf{c}^T(\mathbf{Fz} + \bar{\mathbf{x}}) - \sum_{i=1}^{4} log(\mathbf{F(i,\ :)z} + \bar{\mathbf{x}}(i)) \tag{7}$$

$$\mathbf{c} = \begin{bmatrix} 1 & 1.5 & 1 & 1 \end{bmatrix}^T$$

Uma meio de encontrarmos um ponto inicial que satisfaz todas as restrições do problema original é resolvendo a igualdade $\mathbf{Ax} = \mathbf{b}$ utilizando o método de mínimos quadrados. Assim, foi-se obtido o ponto:

$$\bar{\mathbf{x}} = \begin{bmatrix} 0.23529 & 0.25490 & 0.45098 & 0.90196 \end{bmatrix}^T$$

Também será preciso definir um ponto inicial $\bar{\mathbf{z}}$ para nosso novo problema, (7), pois ele é necessário aos algoritmos que serão aplicados a seguir. Como $\bar{\mathbf{x}}$ é uma solução para o primeiro problema, uma opção trivial para iniciarmos $\bar{\mathbf{z}}$ também satisfazendo as restrições é:

$$\bar{\mathbf{z}} = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$$

No trecho de código abaixo temos os resultado do obtido por cada algoritmo quando executado para o problema da Questão 1. Os valores de t e μ utilizados para o método iterativo de cada algoritmo variaram entre $[0.01, 4]$ e $[1.2, 10]$, respectivamente, e foram decididos experimentalmente.

Listing 1: Resposta do script 'lista04.py' quando executado para a Questão 1.

```
python lista04.py -exe_num 1
Initial point: [0.23529412 0.25490196 0.45098039 0.90196078]

Steepest Descent
x: [0.23529412 0.25490196 0.45098039 0.90196078], fx: 1.97059, num_iter: 85,
    calls: 85, time: 14.38379 ms

Conjugate Gradiente
```

```
x: [7.38777932e-07 3.33333087e-01 4.37983609e-07 1.16666632e+00], fx:
   1.66667, num_iter: 435, calls: 77, time: 80.15609 ms

Quasi-Newton
x: [5.58793211e-07 3.33333147e-01 3.72528971e-07 1.16666639e+00], fx:
   1.66667, num_iter: 160, calls: 30, time: 55.75991 ms

Newton
x: [3.23117481e-05 3.33322563e-01 2.63241467e-05 1.16664812e+00], fx:
   1.66669, num_iter: 103, calls: 30, time: 313.95507 ms
```

Na Tabela 1 temos a comparação dos resultados obtidos para cada algoritmo pedido na questão. O mínimo da função foi alcançado para os métodos do Gradiente Conjugado, Quasi-Newton e Newton. Sendo o método de Newton o mais eficiende dentre os três. Já o Gradiente Descendente não alcançou um resultado satisfatório. Durante sua execução ele teve dificuldades em gerar passos que respeitassem as restrições do problema, fazendo com que ele ficasse preso no mesmo lugar.

Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
Gradiente Desc.	[0.2353, 0.2549, 0.4510, 0.9020]	1.9706	85	14.3838
Gradiente Conj.	[0.0000, 0.3333, 0.0000, 1.1667]	1.6667	435	80.1561
Quasi-Newton	[0.0000, 0.3333, 0.0000, 1.1667]	1.6667	160	55.7599
Newton	[0.0000, 0.3333, 0.0000, 1.1667]	1.6667	103	313.9551

Tabela 1: Comparação dos resultados obtidos para cada algoritmo quando executados para o problema da Questão 1.

2 Questão 2

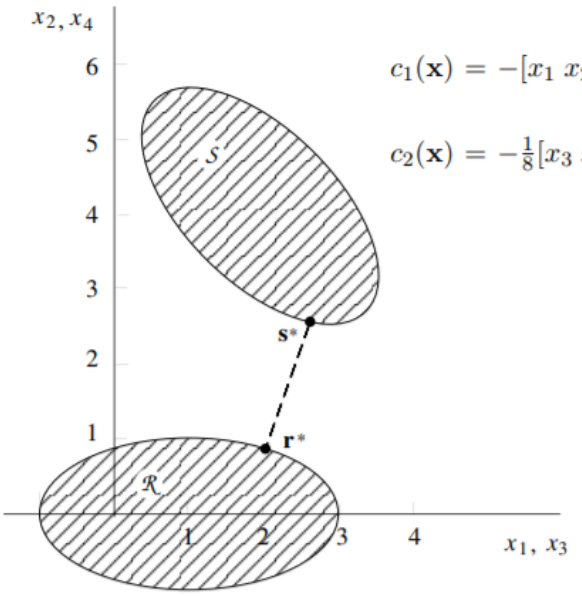
Estime a menor distância entre as duas elipses da Figura 1 abaixo, descritas pelas funções $c_1(x)$ e $c_1(x)$.

Como deseja-se estimar a menor distância entre as elipses, podemos definir a seguinte função a ser minimizada:

$$\text{minimize} \quad (x_1 - x_3)^2 + (x_2 - x_4)^2 \tag{8}$$

$$\text{sujeito a:} \quad c_i(\mathbf{x}) \geq 0 \quad i = 1{,}2 \tag{9}$$

Para resolvê-la precisamos primeiro eliminar as restrições de desigualdade apresentadas em (9). Para tal, seguimos o mesmo processo da Questão 1, aplicando o método da barreira logarítma, e obtemos o novo problema:



$$c_1(\mathbf{x}) = -[x_1\ x_2]\begin{bmatrix}\frac{1}{4} & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + [x_1\ x_2]\begin{bmatrix}\frac{1}{2}\\ 0\end{bmatrix} + \frac{3}{4} \geq 0$$

$$c_2(\mathbf{x}) = -\frac{1}{8}[x_3\ x_4]\begin{bmatrix}5 & 3\\ 3 & 5\end{bmatrix}\begin{bmatrix}x_3\\ x_4\end{bmatrix} + [x_3\ x_4]\begin{bmatrix}\frac{11}{2}\\ \frac{13}{2}\end{bmatrix} - \frac{35}{2} \geq 0$$

Figura 1: Elipses definidas na Questão 2 e suas respectivas equações.

$$\text{minimize} \quad t[(x_1 - x_3)^2 + (x_2 - x_4)^2] - \sum_{i=1}^{2} log(c_i(\mathbf{x})) \tag{10}$$

Para a execução dos algoritmos, foi escolhido o ponto inicial:

$$\mathbf{x}_0 = \begin{bmatrix}1 & 0.5 & 3 & 4\end{bmatrix}^T$$

que satisfaz todas as restrições do problema original.

No trecho de código abaixo temos os resultado do obtido por cada algoritmo quando executado para o problema da Questão 2. Os valores de t e μ utilizados para o método iterativo de cada algoritmo variaram entre $[1, 2.5]$ e $[1.2, 20]$, respectivamente, e foram decididos experimentalmente.

Listing 2: Resposta do script 'lista04.py' quando executado para a Questão 2.

```
python lista04.py -exe_num 2
Initial point: [1, 0.5, 3, 4]

Steepest Descent
x: [1. 0.5 3. 4. ], fx: 4.03113, num_iter: 2, calls: 2, time: 0.60511 ms

Conjugate Gradiente
x: [2.04402098 0.852939 2.54431789 2.48581532], fx: 1.70780, num_iter: 6315,
    calls: 85, time: 754.53782 ms
```

```
Quasi-Newton
x: [1.53417799 0.89993808 2.37823436 2.76089438], fx: 2.04343, num_iter: 7,
   calls: 6, time: 12.25996 ms

Newton
x: [1.9862544 0.76057329 2.63959484 2.57095101], fx: 1.92466, num_iter: 25,
   calls: 6, time: 83.08411 ms
```

Na Tabela 2 temos a comparação dos resultados obtidos para os quatro algoritmos quando executados para o problema em questão. Neste caso, o método do Gradiente Conjugado alcançou o menor valor final para a função, porém ao custo de muitas iterações. Por outro lado o método de Newton e Quasi-Newton alcançaram valores razoáveis com um tempo de execução muito menor.

Assim como na Questão 1, vemos que o método do Gradiente Descendente não conseguiu executar de forma satisfatória, ficando preso no ponto inicial. Variações nos parâmentros t e μ foram testadas, mas mesmo assim não foi possível alterar esta situação.

Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
Gradiente Desc.	[1.0, 0.5, 3.0, 4.0]	4.0311	2	0.6051
Gradiente Conj.	[2.0440, 0.8529, 2.5443, 2.4858]	1.7078	6315	754.5378
Quasi-Newton	[1.5342, 0.8999, 2.3782, 2.7609]	2.0434	160	12.2599
Newton	[1.9862, 0.7605, 2.6396, 2.5709]	1.9247	25	83.0841

Tabela 2: Comparação dos resultados obtidos para cada algoritmo quando executados para o problema da Questão 2.

3 Questão 3

Resolva o problema:

$$\text{minimize} \quad \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 4 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix} \mathbf{x} + \mathbf{x}^T \begin{bmatrix} -8 \\ -6 \\ -6 \end{bmatrix} \tag{11}$$

$$\text{sujeito a:} \quad x_1 + x_2 + x_3 = 3 \tag{12}$$

$$\mathbf{x} \geq 0 \tag{13}$$

Para resolvê-lo precisamos primeiro eliminar a restrição de igualdade (12). Definindo a matriz $\mathbf{A}$ e o vetor $\mathbf{b}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 3 \end{bmatrix}$$

podemos seguir o mesmo processo da Questão 1 e reescrevermos a variável $\mathbf{x}$ em função de uma nova variável $\mathbf{z}$ da forma:

$$\mathbf{x} = \mathbf{F}\mathbf{z} + \bar{\mathbf{x}}$$

sendo $\mathbf{F}$ a matriz espaço-nulo de $\mathbf{A}$ e $\bar{\mathbf{x}}$ uma solução particular para o problema original - que atende a restrição de igualdade. Em seguida é aplicado o método da barreira logarítma para eliminarmos as restrições de desigualdade (13). Assim, obtemos o problema:

$$\text{minimize} \quad t\Big[\frac{1}{2}(\mathbf{Fz}+\bar{\mathbf{x}})^T\mathbf{C}(\mathbf{Fz}+\bar{\mathbf{x}})+(\mathbf{Fz}+\bar{\mathbf{x}})^T\mathbf{d}\Big]-\sum_{i=1}^{3} log(\mathbf{F(i,\ :)z}+\bar{\mathbf{x}}(i)) \tag{14}$$

$$\mathbf{C}=\begin{bmatrix}4 & 0 & 0\\ 0 & 1 & -1\\ 0 & -1 & 1\end{bmatrix}, \quad \mathbf{d}=\begin{bmatrix}-8\\ -6\\ -6\end{bmatrix}$$

Analisando o problema original e suas restrições foi escolhido o ponto inicial:

$$\mathbf{x}_0=\begin{bmatrix}1 & 1 & 1\end{bmatrix}^T$$

e para nosso novo problema (14), definimos o ponto inicial $\bar{\mathbf{z}}$ de forma trivial como sendo:

$$\bar{\mathbf{z}}=\begin{bmatrix}0 & 0\end{bmatrix}^T$$

No trecho de código abaixo temos os resultado do obtido por cada algoritmo quando executado para o problema da Questão 3. Foram utilizados os valores de $t=1$ e $\mu=10$ para o método iterativo de cada algoritmo. Estes valores foram decididos experimentalmente.

Listing 3: Resposta do script 'lista04.py' quando executado para a Questão 3.

```
python lista04.py -exe_num 3
Initial point: [0, 0]

Steepest Descent
x: [1. 1. 1.], fx: -18.00000, num_iter: 2, calls: 2, time: 0.87094 ms

Conjugate Gradiente
x: [0.66307492 1.16846254 1.16846254], fx: -18.44681, num_iter: 6, calls: 2,
    time: 3.93009 ms

Quasi-Newton
x: [0.66307492 1.16846254 1.16846254], fx: -18.44681, num_iter: 4, calls: 2,
    time: 4.42791 ms

Newton
x: [0.66307494 1.16846253 1.16846253], fx: -18.44681, num_iter: 1, calls: 2,
    time: 27.47202 ms
```

Na Tabela 3 temos a comparação dos resultados obtidos para os quatro algoritmos quando executados para o problema em questão. Vemos que os métodos do Gradiente Descendente, Quasi-Newton e Newton foram muito eficientes, aproximando-se do mínimo com um número bastante baixo de iterações. Mais uma vez, o método do Gradiente Descendente não conseguiu executar de forma satisfatória, ficando preso no ponto inicial.

Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
Gradiente Desc.	[1.0, 1.0, 1.0]	-18.0000	2	0.8709
Gradiente Conj.	[0.6631, 1.1685, 1.1685]	-18.4468	6	3.9301
Quasi-Newton	[0.6631, 1.1685, 1.1685]	-18.4468	4	4.4279
Newton	[0.6631, 1.1685, 1.1685]	-18.4468	1	27.4720

Tabela 3: Comparação dos resultados obtidos para cada algoritmo quando executados para o problema da Questão 3.

4 Questão 4

Resolva o problema:

$$\text{minimize} \quad \mathbf{c}^T\mathbf{x} \tag{15}$$

$$\text{sujeito a:} \quad \mathbf{F}_0 + \sum_{i=1}^{4} x_i\mathbf{F}_i \succeq 0 \tag{16}$$

$$\mathbf{c} = \begin{bmatrix} 1 & 0 & 2 & -1 \end{bmatrix}^T$$

com as matrizes $\mathbf{F}_i$, $i = 0, 1, ..4$ definidas.

Para resolver este problema precisamos primeiro eliminar a restrição de desigualdade (16). Porém, diferente das Questões anteriores, temos neste caso uma desigualdade generalizada no cone semidefinido positivo. Logo, a barreira logarítimica ganha um termo que considera o determinante e nosso problema se torna:

$$\text{minimize} \quad t\mathbf{c}^T\mathbf{x} - log\big(det(\mathbf{F}_0 + \sum_{i=1}^{4} x_i\mathbf{F}_i)\big) \tag{17}$$

Ao avaliarmos a função precisamos sempre verificar se $-det(f(x)) \leq 0$. Caso essa condição não seja satisfeita, um erro será gerado na função log, pois estaremos fornecendo uma entrada fora de seu domínio. Porém, mesmo aplicando esta restrição foi observado que durante a execução dos agoritmos os mesmos sempre acabavam divergindo em algum ponto. Isto ocorre pois os algoritmos podem trocar dois autovalores em um mesmo passo, fazendo com que a restrição pareça que foi respeitada quando na verdade ela foi violada.

Para evitar que isto aconteça a restrição $-det(f(x)) \leq 0$ foi substituída por $-eig(f(x)) \leq 0$, sendo $eig(.)$ o menor autovalor da matrix $f(x)$. Essa alteração garante que todos os pontos encontrados respeitem a restrição original desejada.

O ponto inicial necessário a execução dos algoritmos foi obtido iterando-se sobre vetores gerados de forma aleatória até que um deles satisfizesse as restrições do problema. Para os resultados analisados abaixo foi-se utilizado o ponto:

$$\mathbf{c} = \begin{bmatrix} 2.7169 & -0.6894 & -0.4248 & 0.1859 \end{bmatrix}^T$$

No trecho de código abaixo temos os resultado do obtido por cada algoritmo quando executado para o problema da Questão 4. Foram utilizados o valores $t = 1$ e μ variando

entre $[1.9, 20]$ para o método iterativo de cada algoritmo. Estes valores foram decididos experimentalmente.

Listing 4: Resposta do script 'lista04.py' quando executado para a Questão 4.

```
Initial point: [ 2.71694478 -0.68939006 -0.42482106 0.18599084]

Steepest Descent
x: [ 2.71694478 -0.68939006 -0.42482106 0.18599084], fx: 1.68131, num_iter:
   2, calls: 2, time: 1.36089 ms

Conjugate Gradiente
x: [ 2.70999644 -0.68652949 -0.45020142 0.1926772 ], fx: 1.61692, num_iter:
   3508, calls: 2, time: 3301.08595 ms

Quasi-Newton
x: [ 1.6031711 -0.7533002 -0.34223774 0.54333698], fx: 0.37536, num_iter:
   87, calls: 25, time: 156.25095 ms

Newton
x: [ 1.6031711 -0.7533002 -0.34223774 0.54333698], fx: 0.37536, num_iter: 2,
    calls: 2, time: 19.50908 ms
```

Na Tabela 4 temos a comparação dos resultados obtidos para os quatro algoritmos quando executados para o problema em questão. Vemos que os métodos de Quasi-Newton e Newton foram os que obtiveram o melhor desempenho, alcançando um valor ótimo com um custo computacional total bem menor. É importante ressaltar que para este caso o método do Gradiente Descendente conseguiu executar e minimizar a função, tendo um desempenho superior ao método do Gradiente Conjugado, por exemplo.

Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
Gradiente Desc.	[2.7169, -0.6894, -0.4248, 0.1860]	1.6813	2	1.3609
Gradiente Conj.	[2.7100, -0.6865, -0.4502, 0.1927]	1.6169	3508	3301.0859
Quasi-Newton	[1.6032, -0.7533, -0.3422, 0.5433]	0.37536	87	156.2509
Newton	[1.6032, -0.7533, -0.3422, 0.5433]	0.37536	2	19.5091

Tabela 4: Comparação dos resultados obtidos para cada algoritmo quando executados para o problema da Questão 4.

5 Questão 5

Resolva o problema:

$$\begin{aligned} \text{minimize} \quad f(x) =& 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + 90(x_3^2 - x_4)^2 && (18)\\ &+ (x_3 - 1)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2)) && (19)\\ &+ 19.8(x_2 - 1)(x_4 - 1) && (20)\\ \text{sujeito a:} \quad & -10 \leq x_i \leq 10 \quad \text{para} i = 1, 2, 3, 4 && (21) \end{aligned}$$

Para resolver esse problema precisamos primeiro eliminar as restrições de desigualdade expostas em (21). Assim como nas questões anteriores, é aplicado o método da barreira logarítma, obtendo o novo problema:

$$\text{minimize} \quad f(x) - \sum_{i=1}^{4} log(x_i - 10) - \sum_{i=1}^{4} log(-x_i - 10) \tag{22}$$

Analisando o problema original e suas restrições foi escolhido o ponto inicial:

$$\mathbf{x}_0 = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix}^T$$

para executarmos os algoritmos em nosso novo problema (22).

No trecho de código abaixo temos os resultados do obtidos por cada algoritmo quando executado para o problema da Questão 5. Foram utilizados o valores $t = 1$ e $\mu = 10$ para o método iterativo de cada algoritmo, com excessão do Gradiente Descendente, para o qual foi utilizado $t = 0.001$. Estes valores foram decididos experimentalmente.

Listing 5: Resposta do script 'lista04.py' quando executado para a Questão 5.

```
python lista04.py -exe_num 5
Initial point: [2. 2. 2. 2.]

Steepest Descent
x: [1.00002029 1.00004198 0.99996982 0.99994003], fx: 0.00000, num_iter:
   626, calls: 11, time: 82.14307 ms

Conjugate Gradiente
x: [0.99996013 0.9999251 0.99996 0.9999254 ], fx: 0.00000, num_iter: 12,
   calls: 2, time: 1.77312 ms

Quasi-Newton
x: [0.99996012 0.9999251 0.99996001 0.99992539], fx: 0.00000, num_iter: 6,
   calls: 2, time: 1.88684 ms

Newton
x: [0.99960144 0.99925156 0.9996 0.99925406], fx: 0.00002, num_iter: 3,
   calls: 2, time: 45.85505 ms
```

Na Tabela 5 temos a comparação dos resultados obtidos para os quatro algoritmos quando executados para o problema em questão. Todos os métodos alcançaram o ponto ótimo, sendo o Gradiente Conjugado e o Quasi-Newton aqueles com melhor desempenho. Novamente, vale resaltar que o método do Gradiente Descendente executou de forma satisfatória, apesar de apresentar o maior custo computacional total dentre os quatro métodos para neste caso.

Método	x_{min}	$f(x_{min})$	# iter	Tempo total (ms)
Gradiente Desc.	[1.0000, 1.0000, 1.0000, 0.9999]	0.0000	626	82.1431
Gradiente Conj.	[1.0000, 0.9999, 1.0000, 0.9999]	0.00000	12	1.7731
Quasi-Newton	[1.0000, 0.9999, 1.0000, 0.9999]	0.00000	6	1.8868
Newton	[0.9996, 0.9993, 0.9996, 0.9993]	0.00000	3	45.8551

Tabela 5: Comparação dos resultados obtidos para cada algoritmo quando executados para o problema da Questão 5.

6 Questão 6

Refaça detalhadamente os os exemplos 16.1 e 16.2 do *Antoniou* (pp. 536 à 540) utilizando pontos interiores e:

(a) Gradiente descendente;

(b) Gradiente conjugado;

(c) Quase-Newton;

(d) Newton.

6.1 Exemplo 16.1

Queremos resolver o problema:

$$\text{minimize} \quad e(x) = \mathbf{x}^T\mathbf{Q}_l\mathbf{x} - 2\mathbf{b}_l\mathbf{x} + k \tag{23}$$

$$\text{sujeito a:} \quad \begin{bmatrix}\mathbf{A}_p \\ \mathbf{A}_a\end{bmatrix}\mathbf{x} \leq \begin{bmatrix}\mathbf{b}_p \\ \mathbf{b}_a\end{bmatrix} \tag{24}$$

Para tal, precisamos primeiro eliminar as restrições de desigualdade apresentadas em (24). Assim como nas questões anteriores, é aplicado o método da barreira logaritma, gerando o novo problema:

$$\text{minimize} \quad t\Big(\mathbf{x}^T\mathbf{Q}_l\mathbf{x} - 2\mathbf{b}_l\mathbf{x} + k\Big) - \sum_i log(-a_i^T\mathbf{x} + b) \tag{25}$$

Em seguida precisamos calcular os valores de $\mathbf{A}_a, \mathbf{A}_p, \mathbf{b}_a, \mathbf{b}_p, \mathbf{Q}_l, \mathbf{b}_l$ utilizando as fórmulas dos capítulos 9 e 16. A partir desse ponto podemos aplicar os algoritmos pedidos definindo um ponto inicial e as funções de gradiente e Hessiana necessárias a eles.