Optimization and Algorithms
Project report

Group 39
Francisco Nogueira 90071, Miguel Figueira 90144,
Alexandre Fonseca 90210 and Tiago Fonseca 98521

1 Transferring a robot

Task 1 - Norma $l_2{}^2$

Abaixo encontram-se as figuras referentes a cada valor de λ, utilizando a norma euclidiana ao quadrado, $l_2{}^2$. O gráfico da esquerda refere-se à posição ótima do robot, enquanto que o da direita se refere ao sinal de controlo $u(t)$ gerado. A componente $u_1(t)$ está representada a azul, enquanto que $u_2(t)$ está a laranja.

DEPOIS METER A PAGINAÇÂO COMO DEVE SER (FAZER ISTO SO NO FINAL)

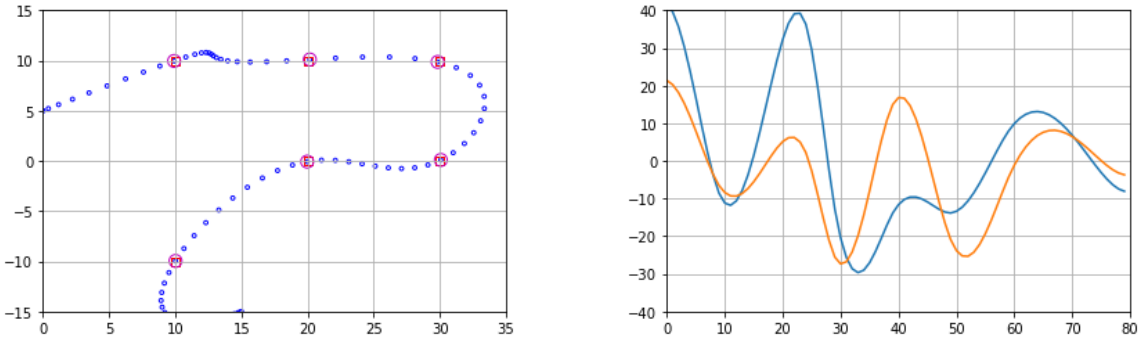

Figura 1: Gráficos para $\lambda = 0.001$, norma $l_2{}^2$

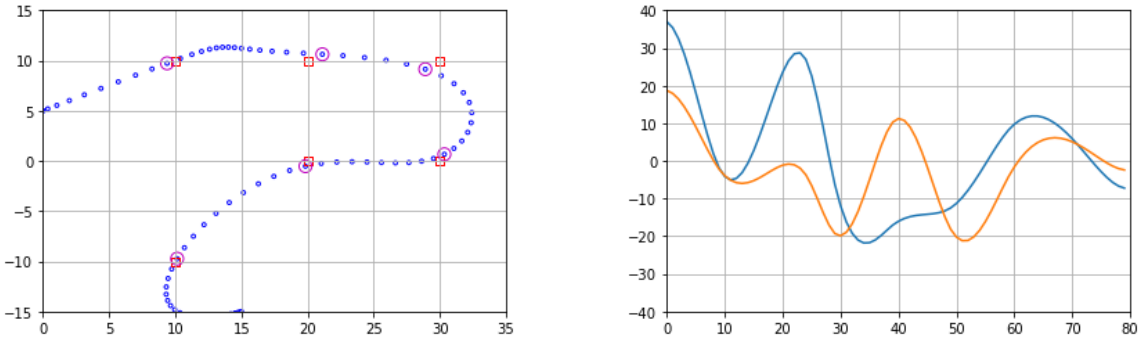

Figura 2: Gráficos para $\lambda = 0.01$, norma ${l_2}^2$

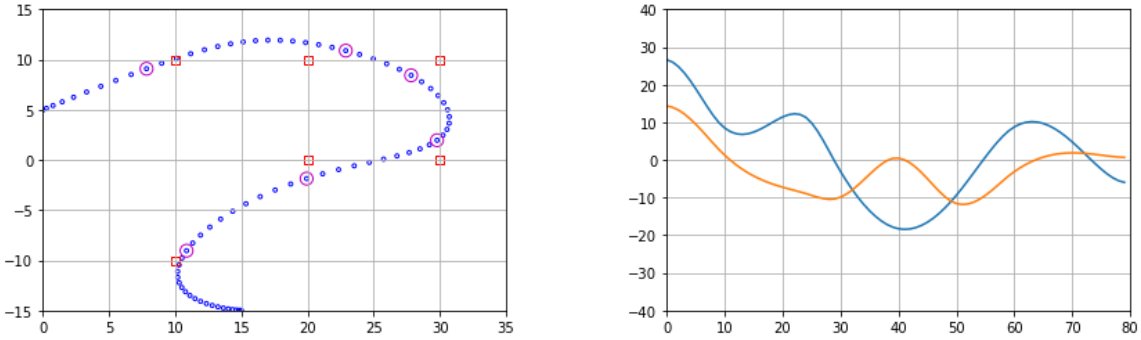

Figura 3: Gráficos para $\lambda = 0.1$, norma ${l_2}^2$

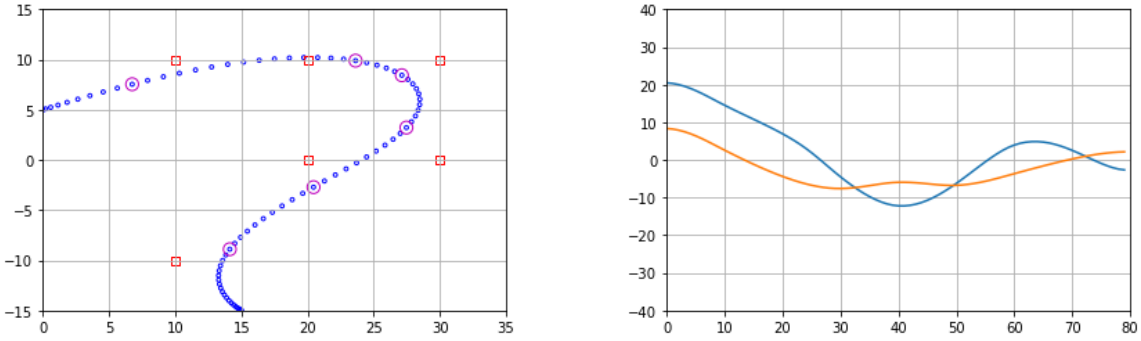

Figura 4: Gráficos para $\lambda = 1$, norma ${l_2}^2$

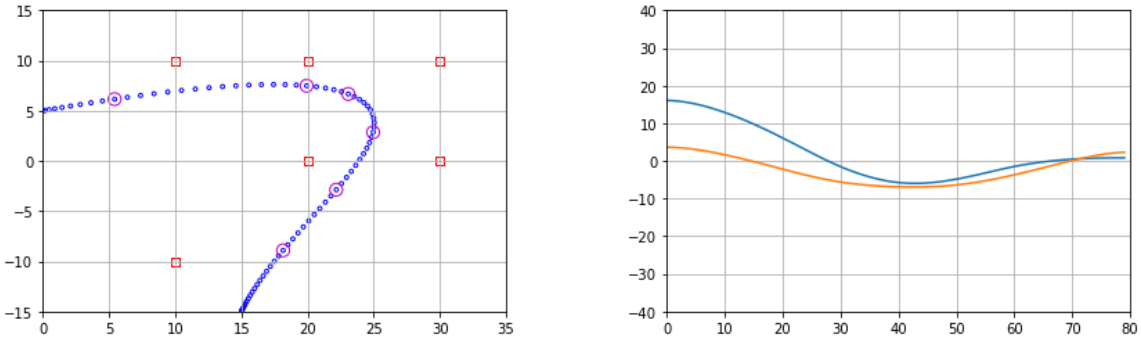

Figura 5: Gráficos para $\lambda = 10$, norma $l_2{}^2$

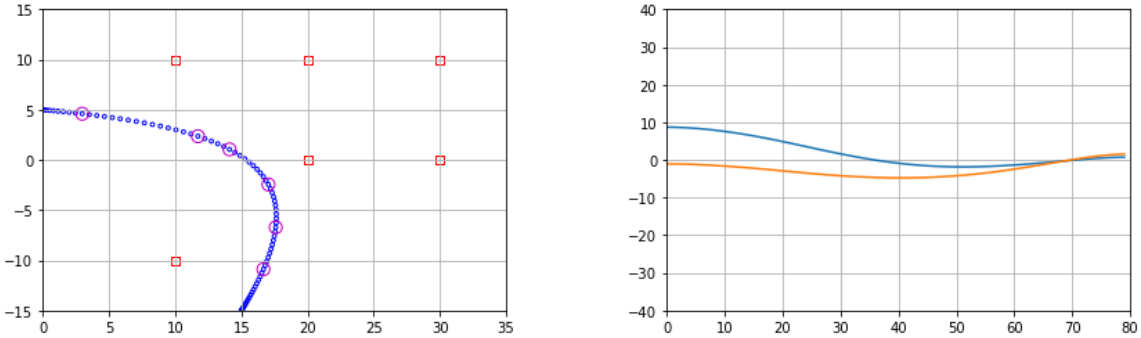

Figura 6: Gráficos para $\lambda = 100$, norma $l_2{}^2$

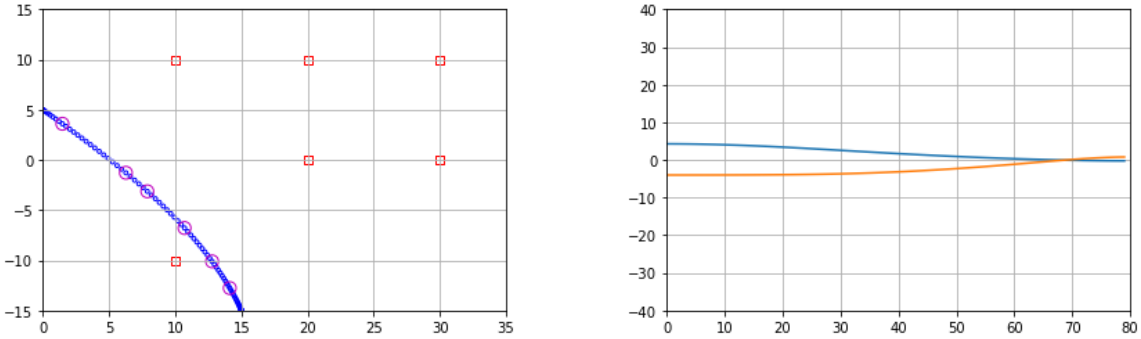

Figura 7: Gráficos para $\lambda = 1000$, norma $l_2{}^2$

Finalmente, o desvio médio para cada um dos valores de λ e o número de alterações no sinal $u(t)$ estão representados na tabela seguinte.

λ	10^{-3}	10^{-2}	10^{-1}	1	10^{1}	10^{2}	10^{3}
Desvio médio	0.1257	0.8242	2.1958	3.6826	5.6317	10.9042	15.3304
Mudanças de u(t)	79	79	79	79	79	79	79

Tabela 1: Desvio médio e mudanças de u(t), norma $l_2{}^2$

Task 2 - Norma l_2

Abaixo encontram-se as figuras referentes a cada valor de λ, utilizando apenas a norma euclidiana, l_2. Tanto os gráficos como as componentes de $u(t)$ se encontram representadas como referido na secção Task 1.

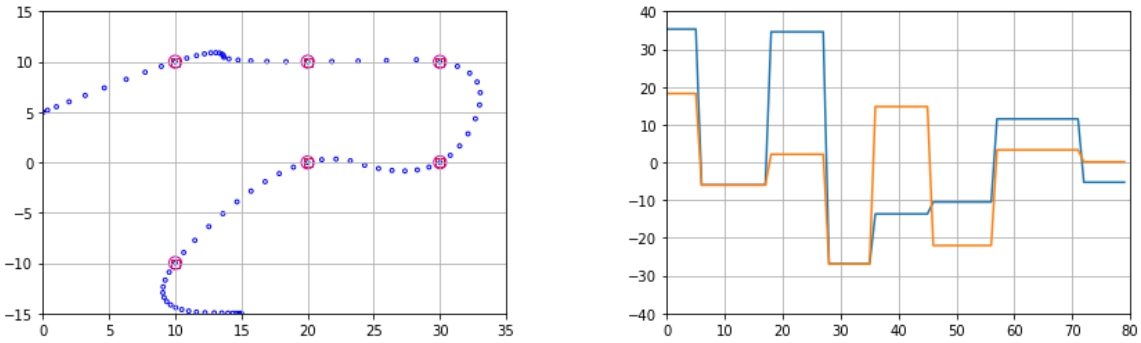

Figura 8: Gráficos para $\lambda = 0.001$, norma l_2

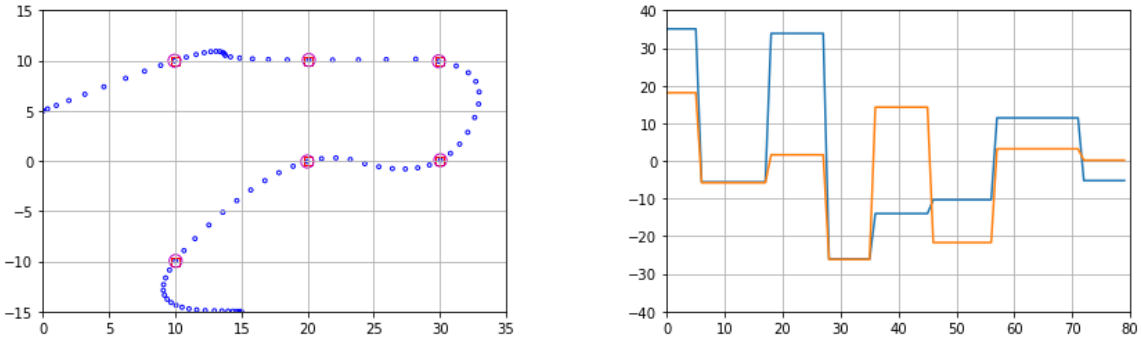

Figura 9: Gráficos para $\lambda = 0.01$, norma l_2

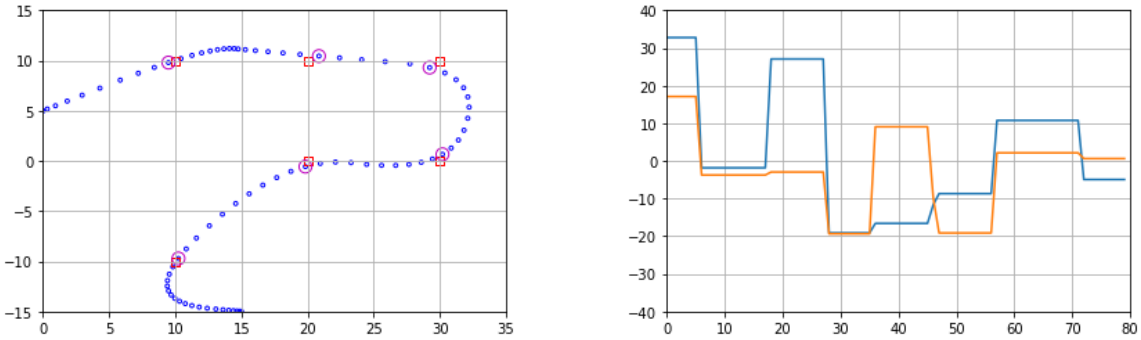

Figura 10: Gráficos para $\lambda = 0.1$, norma l_2

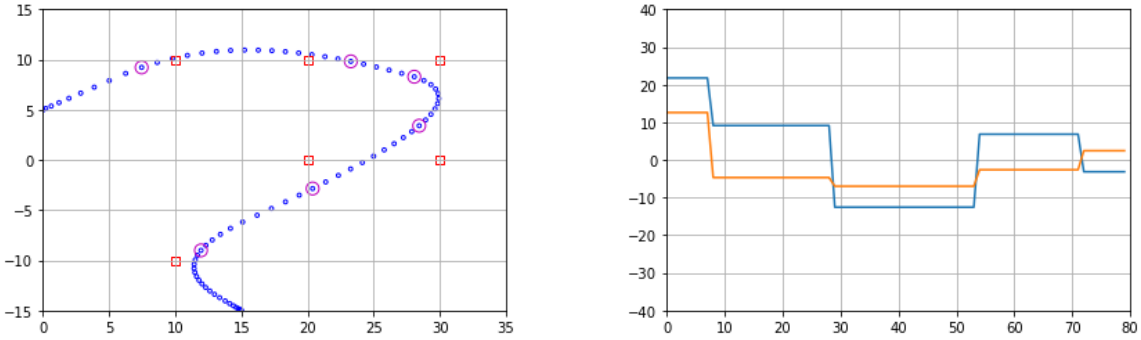

Figura 11: Gráficos para $\lambda = 1$, norma l_2

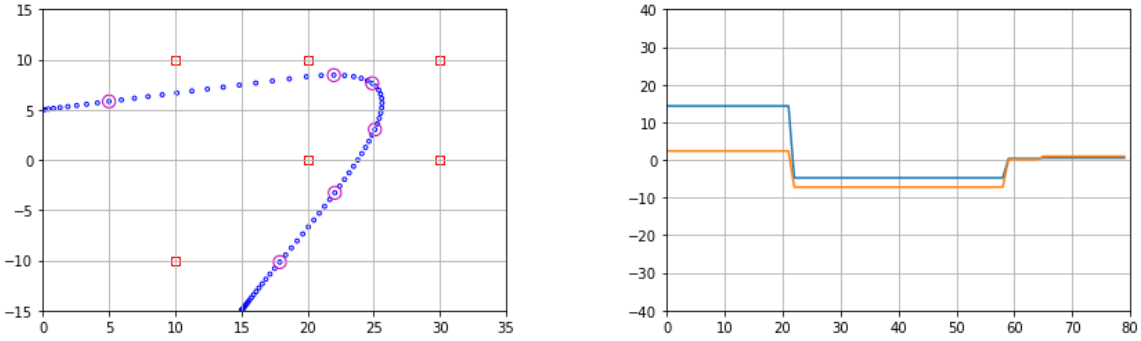

Figura 12: Gráficos para $\lambda = 10$, norma l_2

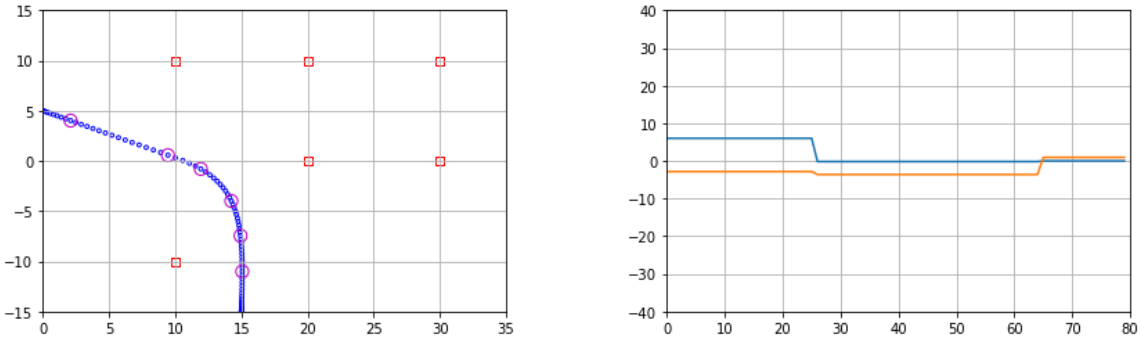

Figura 13: Gráficos para $\lambda = 100$, norma l_2

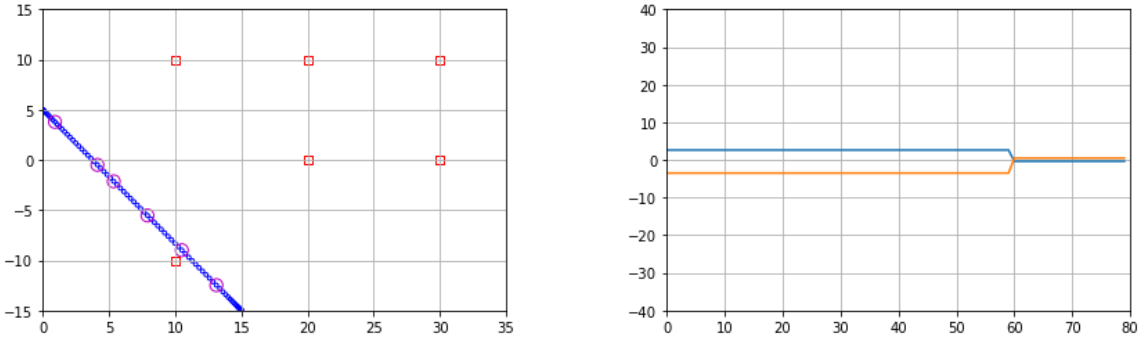

Figura 14: Gráficos para $\lambda = 1000$, norma l_2

Mais uma vez, as informações sobre o desvio médio e o número de mudanças do sinal $u(t)$ estão reunidas na tabela que se segue.

λ	10^{-3}	10^{-2}	10^{-1}	1	10^{1}	10^{2}	10^{3}
Desvio médio	0.0075	0.0747	0.7021	2.8876	5.3689	12.5914	16.2266
Mudanças de u(t)	7	7	8	4	3	2	1

Tabela 2: Desvio médio e mudanças de u(t), norma l_2

Task 3 - Norma l_1

Abaixo encontram-se as figuras referentes a cada valor de λ, utilizando agora a norma l_1.

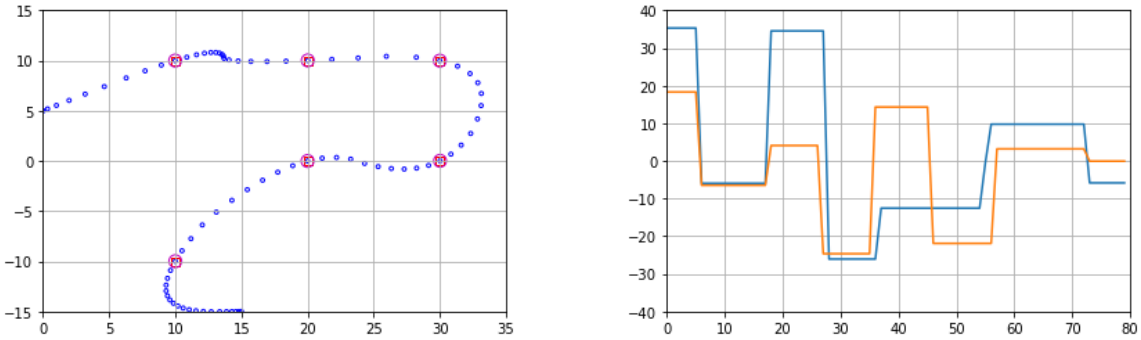

Figura 15: Gráficos para $\lambda = 0.001$, norma l_1

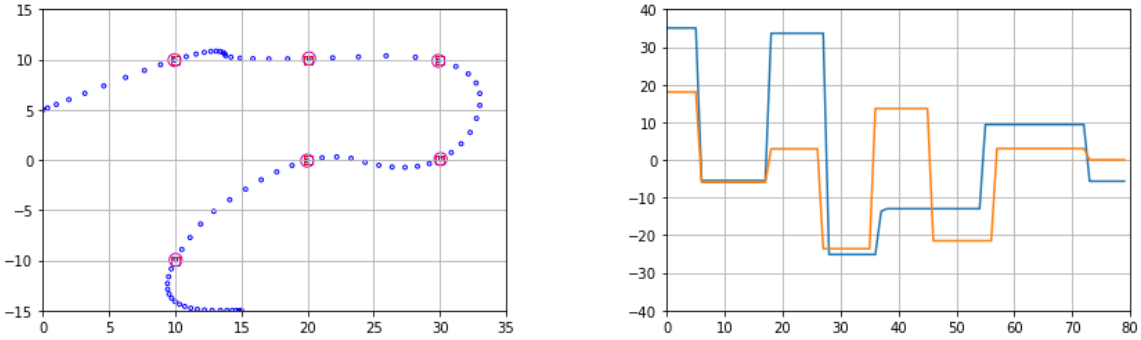

Figura 16: Gráficos para $\lambda = 0.01$, norma l_1

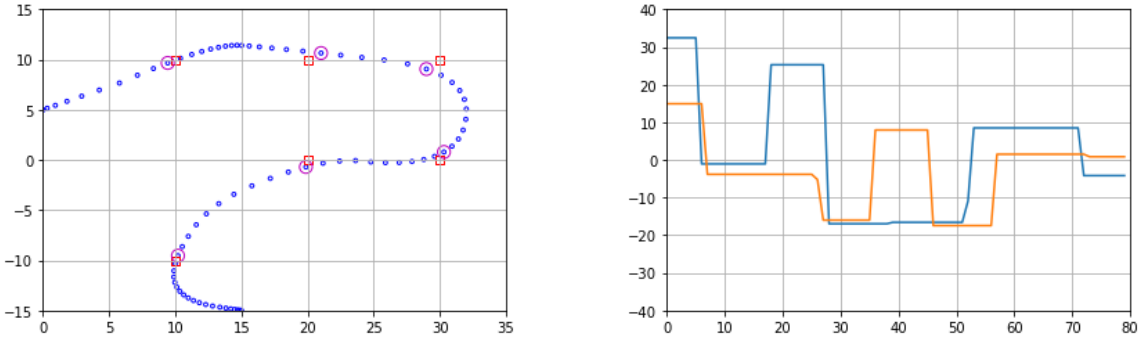

Figura 17: Gráficos para $\lambda = 0.1$, norma l_1

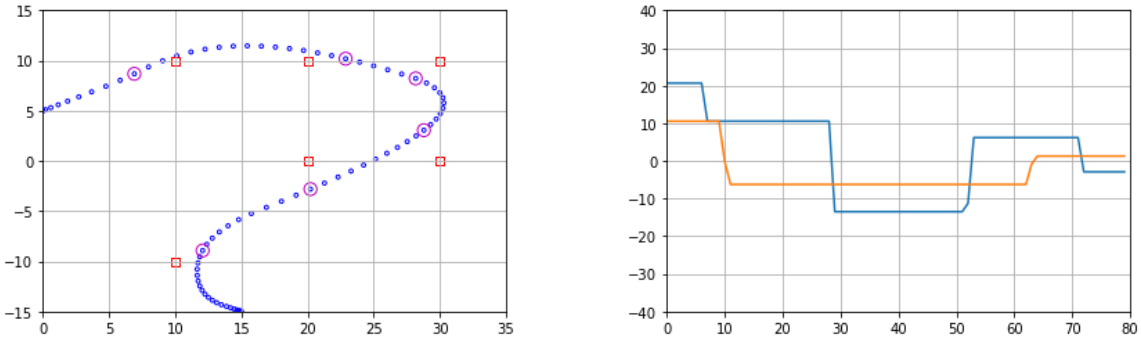

Figura 18: Gráficos para $\lambda = 1$, norma l_1

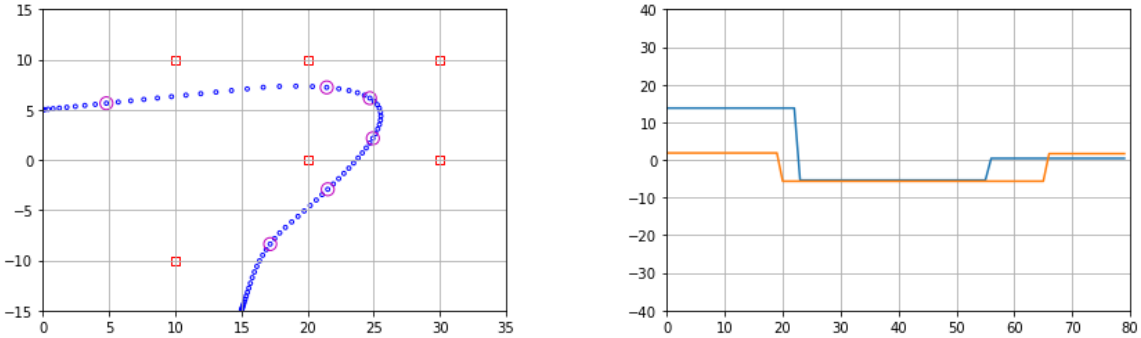

Figura 19: Gráficos para $\lambda = 10$, norma l_1

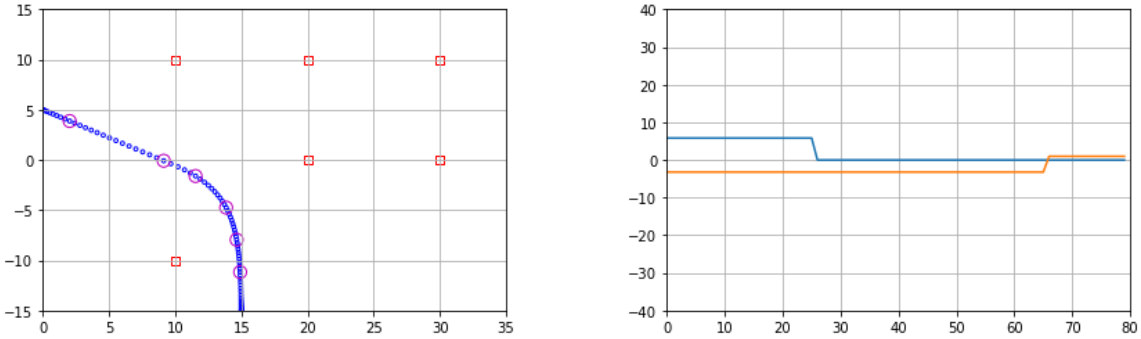

Figura 20: Gráficos para $\lambda = 100$, norma l_1

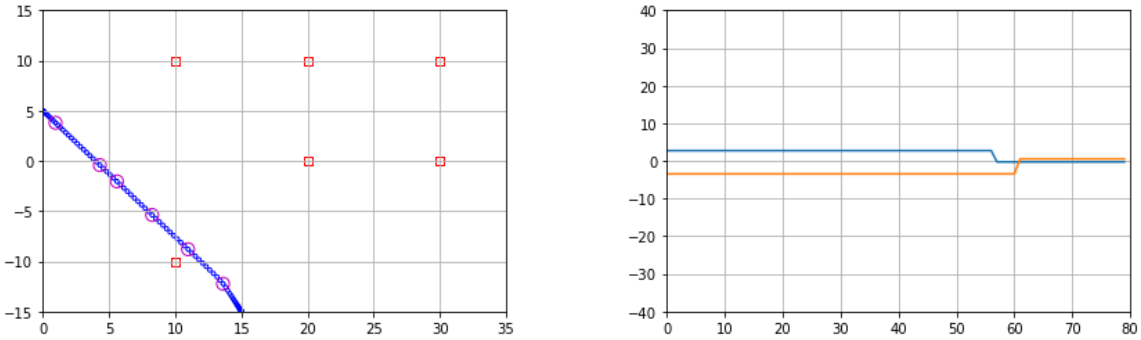

Figura 21: Gráficos para $\lambda = 1000$, norma l_1

Tal como nas secções anteriores, a tabela apresenta o desvio médio e o número de mudanças de $u(t)$.

λ	10^{-3}	10^{-2}	10^{-1}	1	10^{1}	10^{2}	10^{3}
Desvio médio	0.0107	0.1055	0.8863	2.8732	5.4312	13.0273	16.0463
Mudanças de u(t)	11	11	14	9	4	2	2

Tabela 3: Desvio médio e mudanças de u(t), norma l_1

Task 4 - Comentários sobre $u(t)$

Para a norma ${l_2}^2$, ambas as componentes de $u(t)$ se assemelham a uma sinusóide. Por outro lado, para l_2 e l_1, o sinal $u(t)$ é formado por escalões.

A diferença entre a forma destes dois tipos de sinal, deve-se ao facto da norma ${l_2}^2$ penalizar muito mais desvios de $u(t)$ em relação ao instante de tempo anterior. Como tal, ${l_2}^2$ nunca poderia induzir um sinal de controlo formado por escalões, que em certo instante variasse muito. Assim, o sinal de controlo para a norma ${l_2}^2$ tenta variar pouco de instante em instante, formando o referido aspeto sinusoidal.

Com o aumento de λ a amplitude dos escalões diminui para l_2 e l_1 e a sinusóide induzida por ${l_2}^2$ é "amortecida". A diminuição de amplitude de todos os sinais $u(t)$ é consequência do facto da função de custo a minimizar,

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 , \tag{1}$$

aumentar com λ. Se o valor de λ aumentar, as mudanças de $u(t)$ são mais custosas. Como tal, a solução procura que o sinal de controlo se mantenha quase constante ao longo do tempo, ou que varie muito pouco.

A evolução do número de mudanças de $u(t)$ é consequência das afirmações do parágrafo anterior: com o aumento de λ, o sinal de controlo gerado apresenta menos mudanças para l_2

e l_1. É assim também explicado que, para ${l_2}^2$, $u(t)$ elevado pareça quase uma recta (figura (7)), por exemplo. No entanto, como para esta norma as componentes do sinal têm sempre um ligeiríssimo declive, o número de mudanças de $u(t)$ é sempre o máximo, 79.

É de notar que a norma l_2 provoca menos mudanças em $u(t)$ que l_1. Tal pode dever-se ao facto de os escalões, em l_2, serem síncronos em ambas as componentes, ao contrário do que acontece em l_1. Ora, esta observação advém do facto de l_1 penalizar menos as mudanças de $u(t)$. Assim, l_1 "tolera"mais mudanças do sinal de controlo.

Por fim, pode observar-se que para λ pequeno, os sinais gerados para a as diferentes normas têm picos e vales ao mesmo tempo, apesar de formas diferentes. Tal demonstra que, se não for dada muita relevância ao desejo 4 do problema, o sinal de controlo gerado seria semelhante para todas as normas.

2 Locating a moving target

Task 5 - Formulação do problema

Para formular esta tarefa como um problema de otimização, são definidos 3 parâmetros: condição, função de custo e variáveis.

O enunciado estabelece que, durante o movimento do robot, não se conhece a sua posição exata, mas apenas que passou a uma distância menor que R_k de pontos críticos em determinados instantes. Por outras palavras, a condição que é necessário satisfazer é que a distância entre o robot e o ponto crítico c_k, no instante t_k, seja inferior a R_k, em que c_k são as coordenadas do ponto crítico e R_k o raio do círculo centrado em c_k.

Por outro lado, o objetivo do problema é determinar a posição inicial, p_0, e a velocidade v, tal que a posição final do robot seja a mais próxima possível de um ponto estabelecido, x^*. Como tal, as variáveis do problema são p_0 e v e será definida uma função de custo que representa a distância da posição final do robot a x^*, no instante de tempo final definido pelo enunciado, t^*.

Finalmente, o problema de otimização que deve ser resolvido utilizando a biblioteca *cvx* do python consiste em minimizar a distância da posição final a x^*.

Em suma, o problema pode ser definido como:

$$\begin{aligned} \underset{p_0,v}{\text{minimize}} \quad & \left\| x^* - (p_0 + v \cdot t^*) \right\| \\ \text{subject to} \quad & \|c_k - (p_0 + v \cdot t_k)\| \leq R_k \text{ for } 0 \leq k \leq 5 \end{aligned} \tag{2}$$

A solução gerada é apresentada na Figura 22, onde é obtida uma distância mínima de 3.4651 entre a posição final do robot e x^*.

Task 6 - Rectângulo que engloba todas as possíveis posições finais

Nesta tarefa pretende-se descobrir a menor área que abrange todas as possíveis posições finais para um robot que se mova segundo a regra referida na Task 5. Como tal, o movimento do

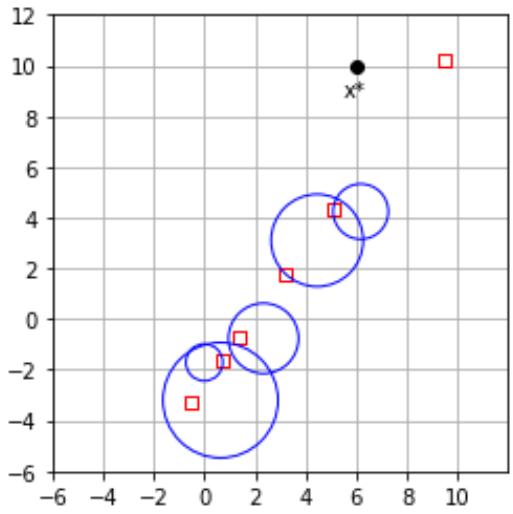

Figura 22: Posições do robot no decorrer do movimento (rectângulos a vermelho)

robot respeita a condição de que a sua distância ao ponto crítico c_k deve ser menor que R_k no instante t_k

O rectângulo que engloba todas as posições finais possíveis para o robot é delimitado por um segmento de reta à esquerda, à direita, em cima e em baixo, claro. Estes extremos serão denominados x_{min} e x_{max}, para os limites à esquerda e à direita, e y_{min} e y_{max}, para os limites acima e abaixo, respetivamente.

São definidas duas funções de custo:

$$cost1 = p_{0x} + v_x \cdot t^*, \text{ referente à abcissa final do robot} \tag{3}$$

$$cost2 = p_{0y} + v_y \cdot t^*, \text{ referente à ordenada final do robot,} \tag{4}$$

em que p_0 e v são variáveis, tal como na tarefa anterior.

Ora, de forma a determinar as variáveis x_{min}, x_{max}, y_{min} e y_{max}, basta utilizar o comando $cvx.Problem$ para as 4 situações que definem as variáveis mencionadas.

Assim, são computados 4 problemas:

$$\begin{aligned}
&\underset{p0\mathrm{x},v\mathrm{x}}{\text{minimize}} && p_{0x} + v_x \cdot t^* \\
&\underset{p0\mathrm{x},v\mathrm{x}}{\text{maximize}} && p_{0x} + v_x \cdot t^* \\
&\underset{p0\mathrm{y},v\mathrm{y}}{\text{minimize}} && p_{0y} + v_y \cdot t^* \\
&\underset{p0\mathrm{y},v\mathrm{y}}{\text{maximize}} && p_{0y} + v_y \cdot t^* \\
&\text{subject to} && \|c_k - (p_0 + v \cdot t_k)\| \leq R_k \text{ for } 0 \leq k \leq 5
\end{aligned} \tag{5}$$

Repare-se que as duas primeiras expressões se referem à abcissa, logo a $cost1$ (eq. 3) e as seguintes duas à ordenada, logo a $cost2$ (equação 4). Minimizar $cost1$ equivale a descobrir o

limite à esquerda, x_{min}, maximizar equivale a descobrir o limite à direita, x_{max}. O mesmo se aplica para $cost2$ e os limites inferior e superior, y_{min} e y_{max}.

Finalmente, após descobrir os extremos, o rectângulo que estes delimitam inclui todas as possíveis posições finais para um robot que se mova respeitando a condição supra mencionada.

A solução gerada é apresentada na Figura 22.

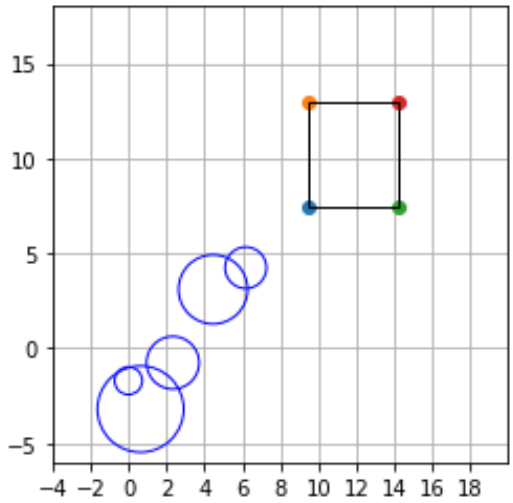

Figura 23: Posições Finais possíveis para o robot (rectângulo a preto)

3 Introduction to logistic regression

Task 1 - Função convexa

De forma a verificar se a expressão 6 é convexa, é necessário decompô-la de forma a que se possam obter funções que sejam também elas convexas. Esta função é formada pela multiplicação de uma constante positiva $(1/K)$ por um somatório de uma função $g(x)$ dada por 7. Deste modo, basta provar que a função $g(x)$ é convexa, pois é sabido que a soma de n funções convexas resulta numa função convexa, e que a multiplicação de uma função convexa por uma constante positiva resulta também numa função convexa.

$$f(x) = \frac{1}{K}\sum_{k=1}^{K}(log(1 + exp(s^T x_t - r)) - y_t(s^T x_t - r)). \tag{6}$$

Assim sendo, é necessário vericar se a função $g(x)$ é convexa. Sendo 7 a soma de duas funções, é necessário mais uma vez verificar se ambas são convexas de que forma a provar a convexidade de $g(x)$.

$$g(x) = log(1 + exp(s^T x - r)) - y_t(s^T x - r). \tag{7}$$

A função $h(x)$ dada em 8 é a multiplicação de uma constante por uma função afim, pelo que se pode garantir a sua convexidade, visto que uma função afim é uma função convexa elementar e mantém essa característica qualquer que seja a constante pela qual se multiplique.

$$h(x) = y(s^T x - r). \tag{8}$$

Para o caso da função $l(x)$ dada em 9, de forma a provar a sua convexidade, é necessário calcular $\nabla^2 l(x)$. Para tal admite-se $s^T x - r = z$. O cálculo de $\nabla^2 l(x)$ é dado pela expressão 10.

$$l(x) = log(1 + exp(s^T x - r))), \tag{9}$$

$$\nabla^2 log(1 + exp(s^T x - r)) = \Big(\frac{e^z}{1 + e^z}\Big)' = \frac{e^{2z} - e^{2z}}{(1 + e^z)^2} = 0. \tag{10}$$

Como $\nabla^2 l(x)$ é continua e $\nabla^2 l \succeq 0$, verifica-se que a função $l(x)$ é convexa.

Assim, sabendo que $l(x)$ e $h(x)$ são funções convexas, garante-se que $g(x)$ é também convexa. Por fim, com a convexidade de $g(x)$ garatinda, é possível provar como originalmente foi proposto, a convexidade da função $f(x)$.

4 Gradient method

Task 2 - Resolução do problema - data1.mat

O método do gradiente é um algoritmo de otimização sem constraints e que, orientado pela direção negativa do gradiente da função no ponto inicial de iteração (s_0 = (-1,-1)) procura encontrar o valor mínimo da função de uma maneira iterativa. A expressão /refeq:2g1 representa a evolução dos objetos da função ao longo das várias iterações do algoritmo, em que α_k é o *step size* respetivo de cada uma delas. A expressão /refeq:2g2 permite obter o valor d_k.

$$x_{k+1} = x_k + \alpha_k d_k \; para \; k = 0, 1, 2, ..., \tag{11}$$

$$d_k = -\nabla f(x_k), \tag{12}$$

$$\alpha_k > 0. \tag{13}$$

Como referido na secção Task 1, a função $f(x)$ é convexa e por isso, ao utilizar-se o método do gradiente é possível chegar à solução do problema de otimização dado. Aplicando este método, obteve-se os valores $r = 4.8815$ e $s = (1.3495, 1.0540)$.

Abaixo encontram-se as figuras referentes à dispersão dos dados com a linha pedida e à norma do gradiente ao longo das iterações do método do gradiente.

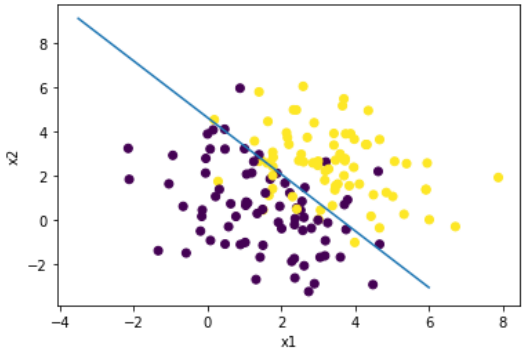

Figura 24: Gráfico de dispersão dos dados *data1.mat* com a linha $\{x \in \ R^2 : s^T x = r\}$.

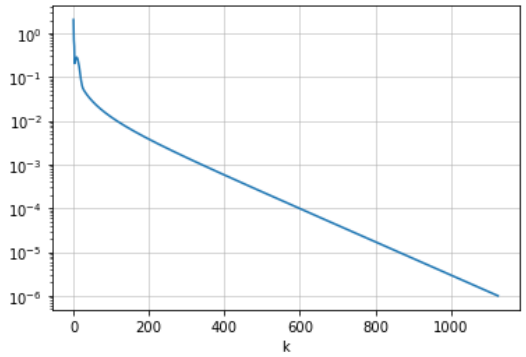

Figura 25: Norma do gradiente ao longo das iterações do método do gradiente com os dados *data1.mat*.

Task 3 - Resolução do problema - data2.mat

Aplicando o método do gradiente para os dados *data2.mat*, obtêm-se os valores $r = 4.5553$ e $s = (0.7402, 2.3577)$.

Abaixo encontram-se as figuras referentes à dispersão dos dados com a linha pedida e à norma do gradiente ao longo das iterações do método do gradiente.

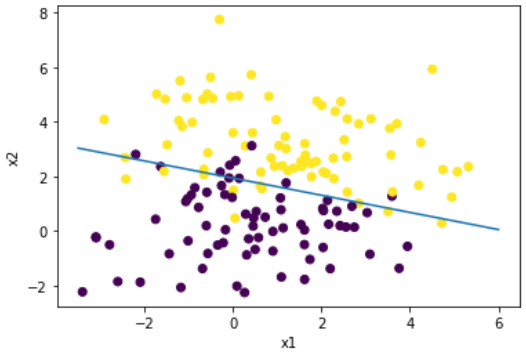

Figura 26: Gráfico de dispersão dos dados *data2.mat* com a linha $\{x \in R^2 : s^T x = r\}$.

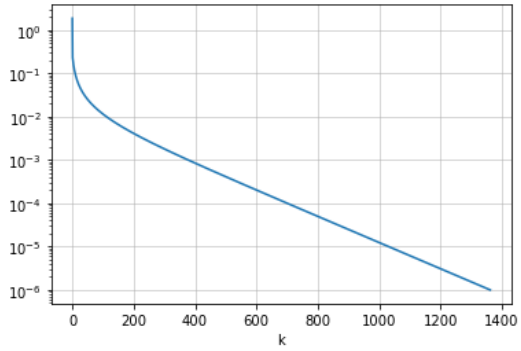

Figura 27: Norma do gradiente ao longo das iterações do método do gradiente com os dados *data2.mat*.

Task 4 - Resolução do problema - data3.mat e data4.mat

Aplicando o método do gradiente para os dados *data3.mat*, obtêm-se os valores $r = 4.7984$ e $s =$ (-1.3082, 1.4078, 0.8049, -1.0024, 0.5548, -0.5489, -1.1997, 0.0792, -1.8279, -0.1484, 1.9241, -0.3586, -0.29 , 0.1925, 1.0614, 0.2107, -0.0929, 1.0476, -1.1248, -1.3311, 0.7661, -0.2729, -0.5349, 0.9996, -0.4191, -0.3133, 0.4075, -0.1965, -0.7379, -0.9814).

Abaixo encontra-se as figura referentea à norma do gradiente ao longo das iterações do método do gradiente.

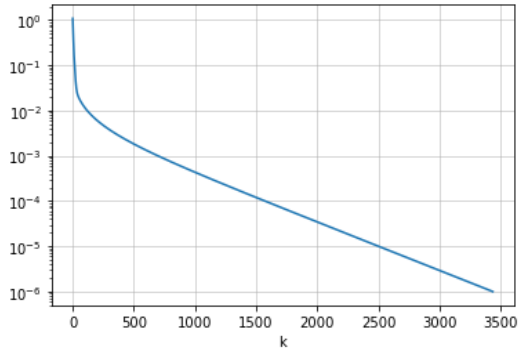

Figura 28: Norma do gradiente ao longo das iterações do método do gradiente com os dados *data3.mat*.

Aplicando o método do gradiente para os dados *data4.mat*, obtêm-se os valores $r = 7.6701$ e $s =$ (0.1098, -0.6423, 0.1019, 1.2428, -1.6431, 1.0244, 0.0512, 0.8271, 0.3136, 0.7449, -0.5858, 0.6267, 1.3611, 0.1534, 2.3234, -0.084 , -0.9489, 2.4699, -0.8678, -1.6516, 0.646 , -0.4779, 1.6397, 0.9034, -1.2293, -0.7587, -0.4887, 1.0306, 0.0888, -1.0917, -1.2717, -2.0333, -0.2505, -0.3518, -0.3486, -2.561 , -0.3132, -0.4902, 0.7258, 0.5774, -1.0528, 0.64 , 0.3759, -0.1547, 0.0298, 0.9547, -0.2863, 0.6364, 0.7859, 0.7584, 0.288 , 0.1648, 0.6776, 2.055 , 1.0996, 0.5261, -0.577 , 1.1454, -0.5617, 0.0065, 0.4768, -2.3677, -1.1561, -2.6619, 0.0622, 0.1037, -0.6237, 0.1913, 0.6672, -1.0493, -0.324 , -0.3207, -1.0904, -0.8293, -0.3104, -0.4879, -0.106 ,

-0.1646, 2.2683, -1.238 , -0.8575, -2.4781, -0.4158, 0.166 , 0.7931, 0.3685, -0.0524, -0.9997, -0.5732, 0.3971, 1.1911, 1.8318, -1.7287, 0.2329, -1.1921, 1.6558, 0.4612, -0.6431, 0.8295, 0.2975).

Abaixo encontra-se as figura referente à norma do gradiente ao longo das iterações do método do gradiente.

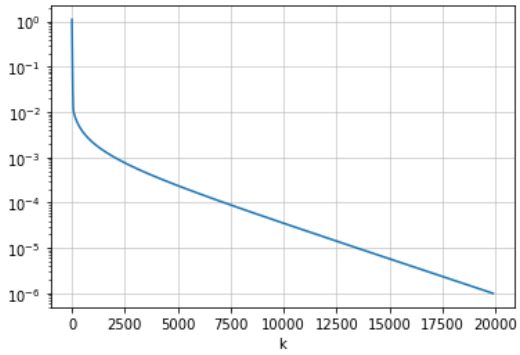

Figura 29: Norma do gradiente ao longo das iterações do método do gradiente com os dados *data4.mat*.

Para os conjuntos de dados *data3.mat* e *data4.mat* não é humanamente possível representar os seus gráficos de dispersão visto que estes conjuntos pertencem a espaços de dimensões superiores a 3.

5 Newton method

Task 5 - Demonstracões

Parte (a)

Para resolver esta demonstração, é trivial conhecer a definição da derivada de uma função composta dada pela expressão 14.

$$y' = (f \circ g)'(x) = f'[g(x_0)]g'(x_0). \tag{14}$$

Assim sendo, é possível provar a igualdade

$$\nabla(\phi(a_k^T x)) = a_k \dot{\phi}(a_k^T x), \tag{15}$$

através da qual é possível calcular a expressão pedida como

$$\nabla p(x) = \nabla\Big(\sum_{k=1}^{K} \phi(a_k^T x)\Big) = \sum_{k=1}^{K} a_k \dot{\phi}(a_k^T x). \tag{16}$$

A expressão 16 é o mesmo que ter

$$\nabla p(x) = Av, \tag{17}$$

em que $A = \begin{bmatrix} a_1 & \cdots & a_k \end{bmatrix}$ e

$$v = \begin{bmatrix} \dot{\phi}(a_1^T x) \\ \vdots \\ \dot{\phi}(a_1^T x) \end{bmatrix}. \tag{18}$$

Parte (b)

A partir da demonstração da *Parte (a)* tem-se

$$\nabla^2 p(x) = \nabla(Av) = \nabla\Big(\sum_{k=1}^{K}[a_k \dot{\phi}(a_k^T x)]\Big) = \sum_{k=1}^{K} \nabla[a_k \dot{\phi}(a_k^T x)] = \sum_{k=1}^{K} a_k^2 \ddot{\phi}(a_k^T x). \tag{19}$$

Sabendo que

$$ADA^T = \begin{bmatrix} a_1 & \cdots & a_k \end{bmatrix} \begin{bmatrix} \ddot{\phi}(a_1^T x) & & \\ & \ddots & \\ & & \ddot{\phi}(a_k^T x) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix} = a_1^2 \ddot{\phi}(a_1^T x) + \cdots + a_k^2 \ddot{\phi}(a_k^T x), \tag{20}$$

e que

$$\sum_{k=1}^{K} a_k^2 \ddot{\phi}(a_k^T x) = a_1^2 \ddot{\phi}(a_1^T x) + \cdots + a_k^2 \ddot{\phi}(a_k^T x), \tag{21}$$

é possível concluir que

$$\nabla^2 p(x) = ADA^T. \tag{22}$$

Task 6 - Resolução do problema utilizando o método de Newton

Aplicando o método de Newton para os dados *data1.mat*, obtêm-se os valores para $r = 4.8817$ e $s = (1,3496; 1,0540)$.

Abaixo encontram-se as figuras referentes à norma do gradiente da função custo ao longo das iterações e ao valor do α_k determinado pelo backtracking subroutine.

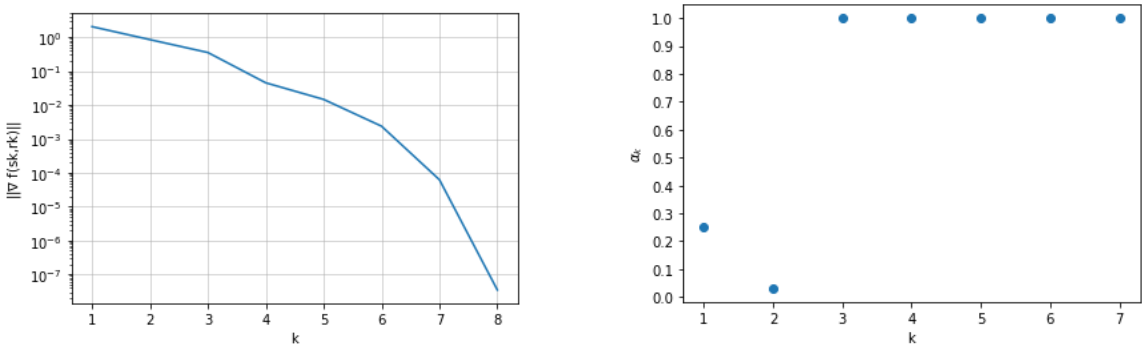

Figura 30: Norma do gradiente da função de custo ao longo das iterações para os dados *data1.mat*, o valor do α_k determinado pelo backtracking subroutine para os dados *data1.mat*

Aplicando o método de Newton para os dados *data2.mat*, obtêm-se os valores $r = 4.5554$ e $s = (0.7402, 2.3577)$.

Abaixo encontram-se as figuras referentes à norma do gradiente da função custo ao longo das iterações e ao valor do α_k determinado pelo backtracking subroutine.

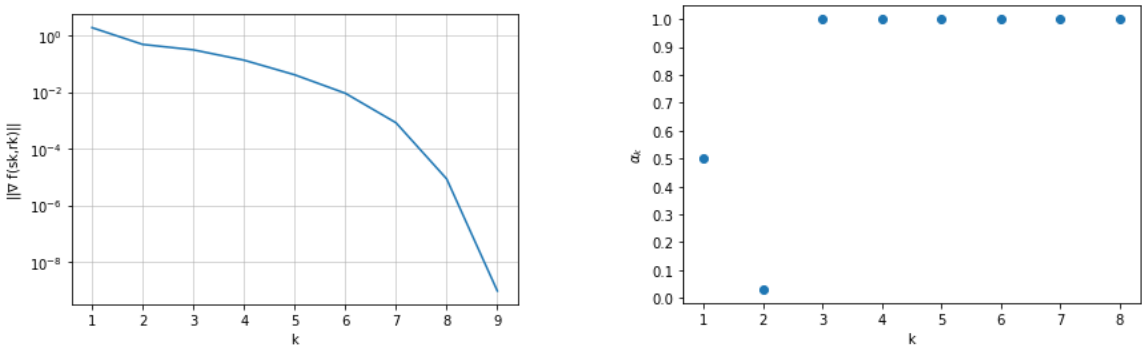

Figura 31: Norma do gradiente da função de custo ao longo das iterações para os dados *data2.mat*, o valor do α_k determinado pelo backtracking subroutine para os dados *data2.mat*

Aplicando o método de Newton para os dados *data3.mat*, obtêm-se os valores $r = 4.7987$ e $s =$ (-1.3083, 1.4079, 0.8049, -1.0025, 0.5548, -0.5489, -1.1998, 0.0792, -1.828 , -0.1484, 1.9242, -0.3586, -0.29 , 0.1925, 1.0615, 0.2107, -0.0929, 1.0477, -1.1249, -1.3311, 0.7662, -0.2729, -0.5349, 0.9996, -0.4192, -0.3133, 0.4075, -0.1965, -0.738 , -0.9815).

Aplicando o método de Newton para os dados *data4.mat*, obtêm-se os valores $r = 7.6718$ e $s =$ (0.1099, -0.6424, 0.1019, 1.2431, -1.6434, 1.0247, 0.0513, 0.8273, 0.3136, 0.7451, -0.5859, 0.6269, 1.3614, 0.1534, 2.3239, -0.084 , -0.9491, 2.4704, -0.868 , -1.652 , 0.6462, -0.478 , 1.6401, 0.9036, -1.2296, -0.7589, -0.4888, 1.0308, 0.0888, -1.0919, -1.272 , -2.0337, -0.2506, -0.3519, -0.3487, -2.5616, -0.3133, -0.4903, 0.7259, 0.5775, -1.0531, 0.6401, 0.376 , -0.1548, 0.0298, 0.9549, -0.2863, 0.6365, 0.786 , 0.7586, 0.2881, 0.1649, 0.6777, 2.0555, 1.0998, 0.5262,

-0.5771, 1.1456, -0.5618, 0.0065, 0.4769, -2.3682, -1.1564, -2.6624, 0.0622, 0.1037, -0.6238, 0.1913, 0.6674, -1.0495, -0.3241, -0.3208, -1.0906, -0.8295, -0.3105, -0.488 , -0.106 , -0.1646, 2.2688, -1.2383, -0.8577, -2.4786, -0.4159, 0.166 , 0.7933, 0.3686, -0.0524, -0.9999, -0.5733, 0.3972, 1.1913, 1.8322, -1.7291, 0.233 , -1.1923, 1.6562, 0.4613, -0.6432, 0.8297, 0.2976).

Task 7 - Comentário dos resultados obtidos

Como analisado nas Tasks anteriores, pode verificar-se as diferenças entre o método do gradiente e o método de Newton.

Para todos os dados, o método de gradiente apresentou valores ligeiramente inferiores (sendo esta diferença quase insignificativa). A diferença significativa entre os dois métodos é o tempo que demora a devolver a solução. Embora o método de Newton precise de maior memória e mais tempo a calcular cada iteração, visto que precisa de fazer menos iterações (devido ao d_k) , acaba por ser mais rápido a calcular a solução do que o método do gradiente. Nos dados *data4.mat* o método de Newton demorou cerca de 8 segundos a devolver a solução enquanto o método do gradiente demorou alguns minutos. Ambos os métodos procuram encontrar o mínimo da função de uma forma iterativa, todavia, apresentam formulas diferentes. O método de Newton segue a seguinte iteração:

$$x_{k+1} = x_k + \alpha_k d_k \; para \; k = 0, 1, 2, ..., \tag{23}$$

$$d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k), \tag{24}$$

$$\alpha_k > 0. \tag{25}$$

Assim sendo, pode observar-se que a diferença no modo iterativo dos métodos consiste no calculo do d_k justificando assim a diferença significativa no tempo de desempenho dos métodos.

O d_k do método do gradiente acaba por ser uma aproximação linear de primeira ordem enquanto que o d_k do método de Newton acaba por ser uma aproximação de segunda ordem (matriz Hessiana) e por isso garante uma descida do modelo do gradiente mais rapida.

6 Dimensionality reduction with Multidimensional Scaling

Task 1 - Matriz de distâncias de *data_opt.csv*

Para se obter a matriz de distâncias (D) do dataset *data_opt.csv*, é necessário fazê-lo computacionalmente devido à dimensão do dataset. Neste projeto a linguagem de programação utilizada é o Python, pelo que para o cálculo da matriz D são necessários comandos bastante simples.

Posto isto, e confirmados os valores $D_{23} = 5.8749$ e $D_{45} = 24.3769$ é possível calcular a maior distância entre dois quaisquer pontos do dataset $D_{max} = 83.0030$ como também o par de pontos $(32, 133)$ (índices em notação de Python) a que esta corresponde.

Abaixo, encontra-se código correspondente a este exercício (ficheiro "parte3.py"):

```
import numpy as np
import scipy.spatial as spatial
k = 2
#Load ficheiros
y_file = 'yinit{}.csv'.format(k)
data = np.genfromtxt('data_opt.csv', delimiter = ',')
y = np.genfromtxt(y_file, delimiter = ',')
y = np.reshape(y, (len(y), 1))#converter para ter dimens o N,1 em vez de N,

#Task 1
D = spatial.distance.cdist(data, data, 'euclidean')
size = len(D)

maximum_dist = np.amax(D)
maximum_dist_index = []
maximum_dist_index.append(np.argmax(D) // size)
maximum_dist_index.append(np.argmax(D) % size)
print('Distancia m xima =',maximum_dist,'em D(',maximum_dist_index[0],',',
      maximum_dist_index[1],')\n')
```

6.1 Levenberg-Marquardt (LM) method

Task 2 - Expressões de $f_{nm}(y)$, $\nabla f(y)$, $\nabla f_{nm}(y)$, A e b

Através do problema de otimização dado no enunciado é possível obter as expressões de $f_{nm}(y)$, $\nabla f(y)$ e $\nabla f_{nm}(y)$. Sendo $f(y)$ a função que se pretende minimizar, $f_{nm}(y)$ é a raíz da função a que se aplicam os dois somatórios em $f(y)$, ou seja, é dada por

$$f_{nm}(y) = ||y_m - y_n|| - D_{mn}, \tag{26}$$

em que y_m e y_n representam as posições de um determinado ponto, e D é a matriz de distâncias calculada da Task 1.

Para obter $\nabla f(y)$ é necessário calcular a derivada de $f(y)$. Este cálculo é possível através da expressão:

$$\nabla f(y) = \nabla\Big(\sum [f_{nm}(y)]^2\Big) = \sum \nabla f_{nm}^2(y), \tag{27}$$

que substituindo f_{nm} pela expressão 26 fica

$$\sum \nabla(||y_m - y_n|| - D_{mn})^2 = \sum \nabla(||y_m - y_n||^2 - 2D_{mn}||y_m - y_n|| + D_{mn}^2), \tag{28}$$

de onde se obtêm as expressões:

$$\nabla f_{nm}^2(y) = \begin{cases} 0, & \text{if } j \neq m, n \\ 2(y_m - y_n) - 2D_{mn}\frac{y_m - y_n}{||y_m - y_n||}, & \text{if } j = m \\ -2(y_m - y_n) + 2D_{mn}\frac{y_m - y_n}{||y_m - y_n||}, & \text{if } j = n \end{cases}. \tag{29}$$

Juntando a equação 27 com as equações dadas em 29 é possível obter a expressão de $\nabla f(y)$.

A expressão de $\nabla f_{nm}(y)$ obtém-se fazendo a derivada da expressão 26. Tem-se portanto

$$\nabla f_{nm}(y) = \nabla(||y_m - y_n|| - D_{mn}) = \begin{bmatrix} \frac{\partial f_{mn}}{\partial y_1} & \cdots & \frac{\partial f_{mn}}{\partial y_j} & \cdots & \frac{\partial f_{mn}}{\partial y_N} \end{bmatrix}^T, \tag{30}$$

que resulta no seguinte conjunto de equações:

$$\nabla f_{nm}(y) = \begin{cases} 0, & \text{if } j \neq m, n \\ \frac{y_m - y_n}{||y_m - y_n||}, & \text{if } j = m \\ \frac{y_n - y_m}{||y_m - y_n||}, & \text{if } j = n \end{cases}. \tag{31}$$

A matriz A e o vetor b definidos no slide 91 do conjunto de slides "Part 2: unconstrained optimization" são neste caso compostos da seguinte forma:

$$A = \begin{bmatrix} \nabla f_{12}(y_k)^T \\ \nabla f_{13}(y_k)^T \\ \vdots \\ \nabla f_{1n}(y_k)^T \\ \nabla f_{23}(y_k)^T \\ \vdots \\ \nabla f_{mn}(y_k)^T \\ \sqrt{\lambda_k} I \end{bmatrix}, b = \begin{bmatrix} \nabla f_{12}(y_k)^T y_k - f_{12}(y_k) \\ \nabla f_{13}(y_k)^T y_k - f_{13}(y_k) \\ \vdots \\ \nabla f_{1n}(y_k)^T y_k - f_{1n}(y_k) \\ \nabla f_{23}(y_k)^T y_k - f_{23}(y_k) \\ \vdots \\ \nabla f_{mn}(y_k)^T y_k - f_{mn}(y_k) \\ \sqrt{\lambda_k} y_k \end{bmatrix}. \tag{32}$$

Task 3 - Aplicação do método LM

A implementação do algoritmo de Levenberg-Marquardt realizada encontra-se no troço de código abaixo, assim como o código utilizado para visualizar os gráficos pedidos (ficheiro "parte3.py"):

```
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 26 15:15:49 2020

@author: Alexandre, Francisco, Miguel, Tiago
"""

import matplotlib.pyplot as plt
import numpy as np
import scipy.spatial as spatial
import matplotlib.ticker as ticker
import time
import datetime

start = time.time()
#Mudar valor de k dependendo do ficheiro de dados a usar
k = 2
#Load ficheiros
y_file = 'yinit{}.csv'.format(k)
data = np.genfromtxt('data_opt.csv', delimiter = ',')
y = np.genfromtxt(y_file, delimiter = ',')
y = np.reshape(y, (len(y), 1))#converter para ter dimens o N,1 em vez de N,

lambda_0 = 1

#Task 3
grad_f_history = []
cost_history = []
b = []
epsilon = k*1e-2
lambda_k = lambda_0
iterations = 0

while True:
    cost = 0
    b = []
    A = []
    list_grad_fnm = []
    list_grad_fnm_square = []

    for m in range(0, k*size-1, k):
        for n in range(m + k, k*size-1, k):
            partial_grad = np.zeros(k*size)
            grad_fnm = np.zeros(k*size)
            norm = []
            for i in range(k):
                #Componentes da norma
                norm.append(y[m+i]-y[n+i])
            norm = np.linalg.norm(norm)
            #Valor da  f u n   o de custo
            cost += (norm - D[int(m/k), int(n/k)])**2
```

```
            for i in range(k):
                partial_grad[m+i] = (y[m+i]-y[n+i])/norm
                partial_grad[n+i] = -partial_grad[m+i]
                grad_fnm[m+i] = 2*(y[m+i] - y[n+i]) - \
                2*D[int(m/k),int(n/k)]*partial_grad[m+i]

                grad_fnm[n+i] = -2*(y[m+i] - y[n+i]) - \
                2*D[int(m/k),int(n/k)]*partial_grad[n+i]

            list_grad_fnm.append(partial_grad)
            list_grad_fnm_square.append(grad_fnm)
            b = np.append(b, partial_grad@y - (norm - D[int(m/k),int(n/k)]))

    b = np.reshape(np.array(b), (len(b), 1))
    b = np.vstack((b, np.sqrt(lambda_k)*y))
    A = np.vstack((list_grad_fnm, np.sqrt(lambda_k)*np.identity(size*k)))
    ls = np.linalg.lstsq(A, b)

    cost_ls = 0
    for m in range(0, k*size-1, k):
        for n in range(m + k, k*size-1, k):
            norm = []
            for i in range(k):
                norm.append(ls[0][m+i]-ls[0][n+i])
            norm = np.linalg.norm(norm)
            cost_ls += (norm - D[int(m/k), int(n/k)])**2

    grad_f = np.linalg.norm(sum(list_grad_fnm_square))
    grad_f_history.append(grad_f)

    if grad_f < epsilon:
        break

    if cost_ls < cost:
        y = ls[0]
        lambda_k *= 0.7
        cost_history.append(cost_ls)
    else:
        lambda_k *= 2
    iterations += 1


end = time.time()
print('Tempo (LM) =', datetime.timedelta(seconds = end - start))
#Scatter dos dados para k=2
"""
plt.figure()
plt.grid()
plt.scatter(y[0:len(y):k],y[k-1:len(y):k], s=8)
plt.xlabel('y1')
plt.ylabel('y2')
```

```
plt.title('Dados com dimens o reduzida para k=2')
plt.show()
"""
#F u n   o de custo ao longo do algoritmo

plt.figure()
plt.grid()
plt.plot(range(len(cost_history)),cost_history, marker='o', markersize=4)
plt.yscale('log')
plt.xlabel('I t e r a   o')
plt.ylabel('Custo')
plt.xticks(range(0,iterations,4))
plt.title('F u n   o de custo ao longo das i t e r a   e s do algoritmo')
plt.show()

#M d u l o do gradiente ao longo do algoritmo

plt.figure()
plt.grid()
plt.plot(range(len(grad_f_history)),grad_f_history[0:len(grad_f_history)],
         marker='o', markersize=4)
plt.yscale('log')
plt.xlabel('I t e r a   o')
plt.ylabel('||'r'$\nabla$f(y)||')
plt.xticks(range(0,iterations+1,4))
plt.title('M d u l o do gradiente da f u n   o de custo ao\nlongo das\
 i t e r a   e s do algoritmo')
plt.show()

#Scatter dos dados para k=3
"""
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plt.grid()
ax.scatter(y[0:len(y):k],y[1:len(y):k],y[k-1:len(y):k], s=8)
ax.view_init(elev=30, azim=270)
plt.xlabel('y1')
plt.ylabel('y2')
ax.set_zlabel('y3')
plt.title('Dados com dimens o reduzida para k=3')
plt.savefig('datak3.png', dpi=300)
plt.show()
"""
```

Os gráficos obtidos para a redução dimensional dos dados para $k = 3$ encontram-se nas figuras.

Figura 32: Representação gráfica dos dados no fim do algoritmo para k=3

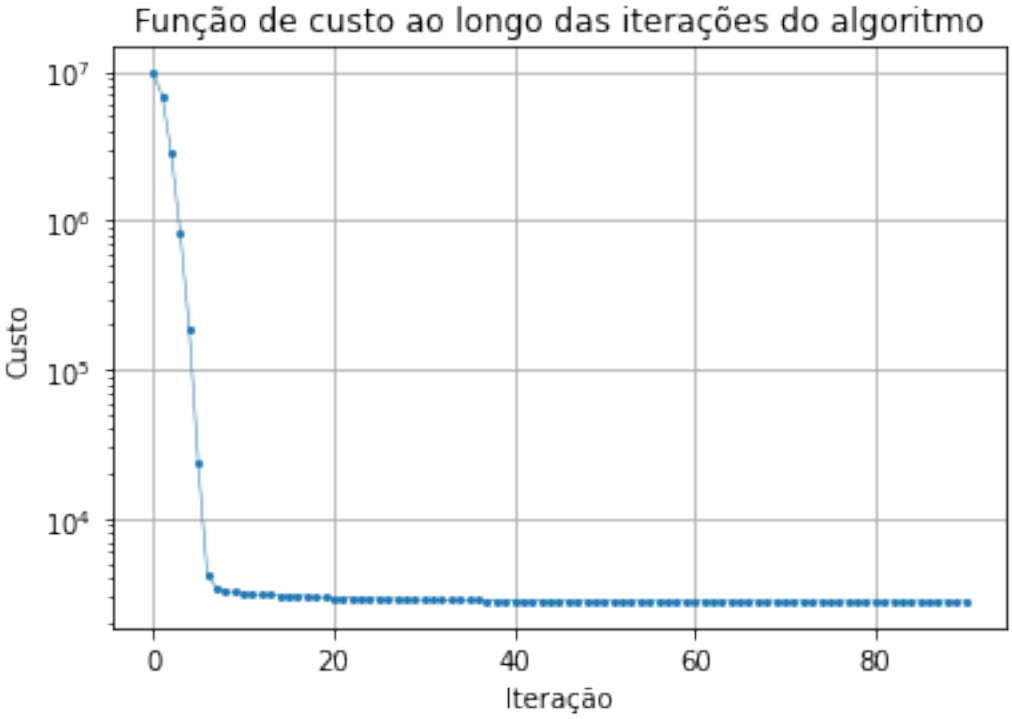

Figura 33: Valor da função de custo ao longo das iterações do algoritmo para k=3

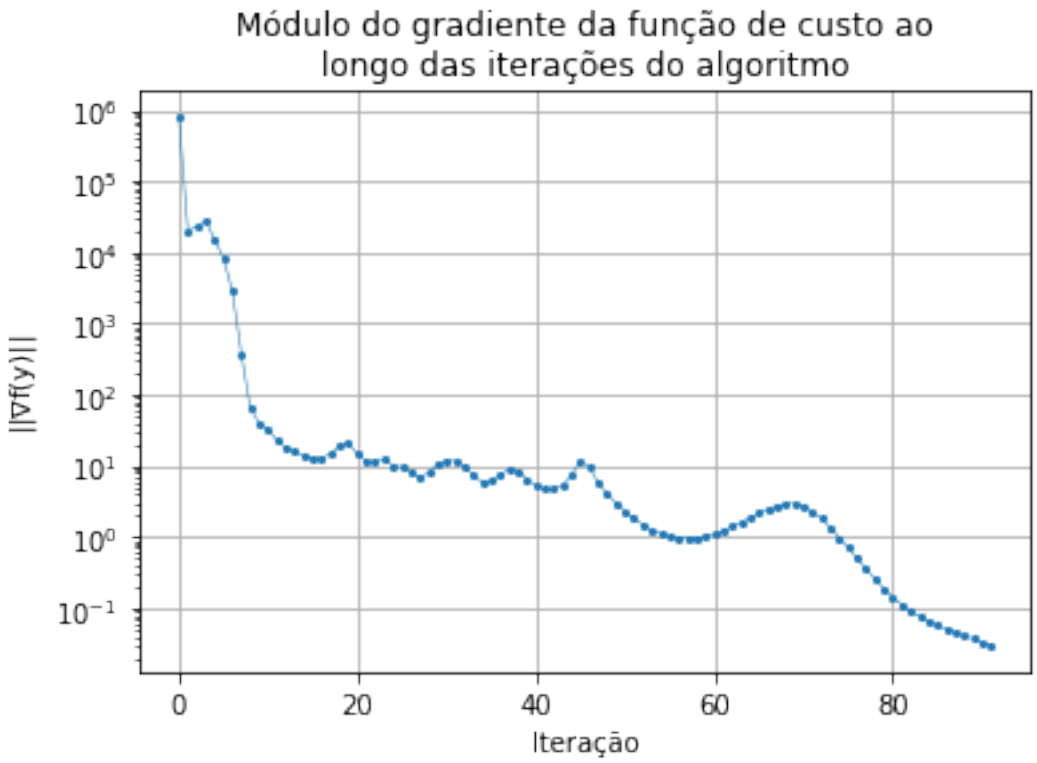

Figura 34: Módulo do gradiente da função de custo ao longo das iterações do algoritmo para k=3

Task 4 - Resolução do problema (1) através do método LM