Optimização e Algoritmos
Relatório do projeto

Groupo 27
Leandro Almeida 84112, Vasco Candeias 84196 e Pedro Moreira 85228

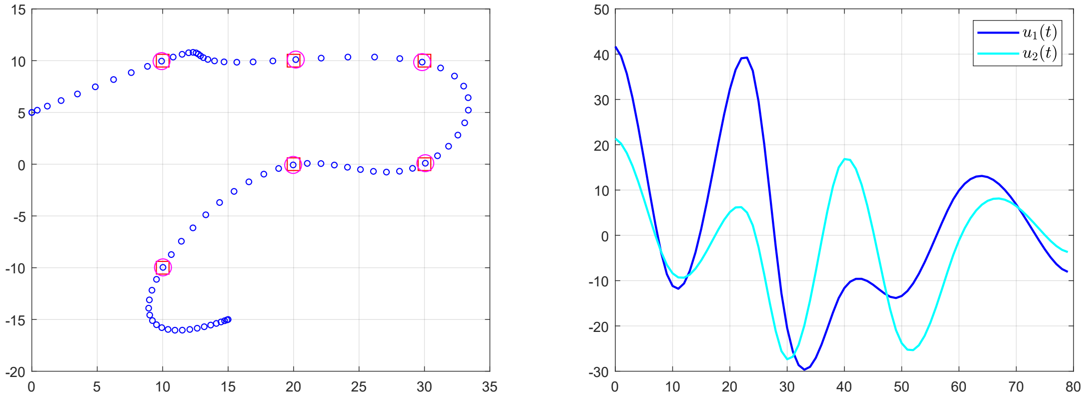
10 de Outubro de 2019

Parte I

1 Task 1

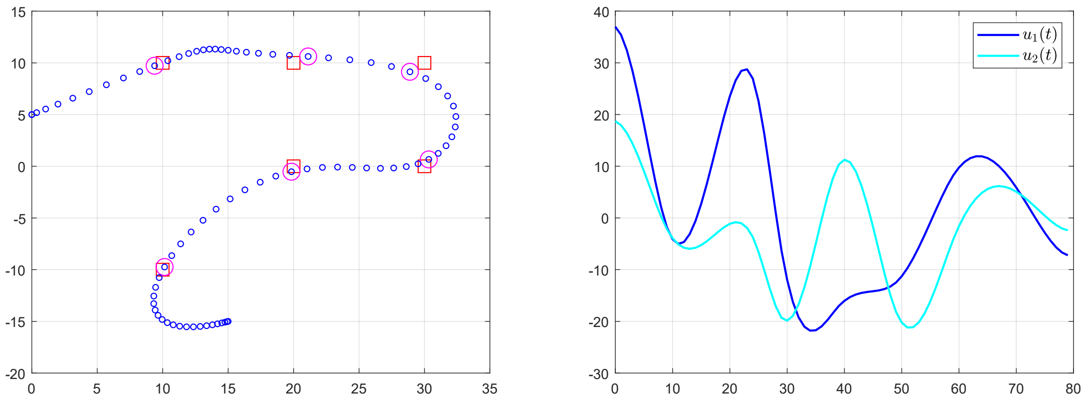
Pretende-se que o robô, no seu trajeto, passe o mais perto possível dos *waypoints* e que comece e acabe exatamente em pontos específicos, $x(0)$ e $x(T)$, com o menor número possível de variações do seu sinal de controlo e nunca excedendo a magnitude máxima $U_{max} = 100$. Convertendo estes desejos num problema de minimização, analisa-se a utilização de três diferentes regularizadores e, para cada um deles, para diversos valores de λ. Esta variável, que aparece a multiplicar pelo termo que representa as variações do sinal de controlo, define qual dos dois últimos desejos terá maior peso na função de custo.

As posições ótimas do robô e o sinal de controlo encontrado para satisfazer os quatro desejos, utilizando o regularizador ℓ_2^2, estão expressos nas figuras 1 – 7.



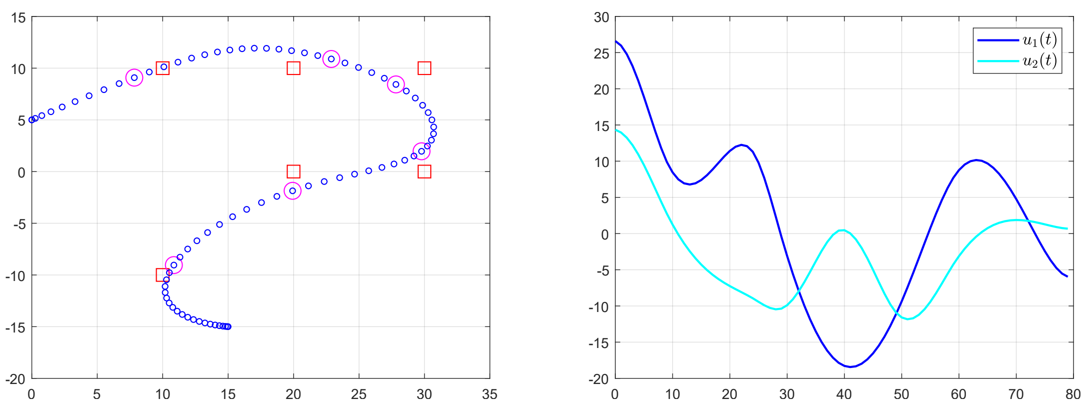
(a) Trajeto do robô (b) Sinal de controlo

Figura 1: Resultados para $\lambda = 10^{-3}$ com o regularizador ℓ_2^2.



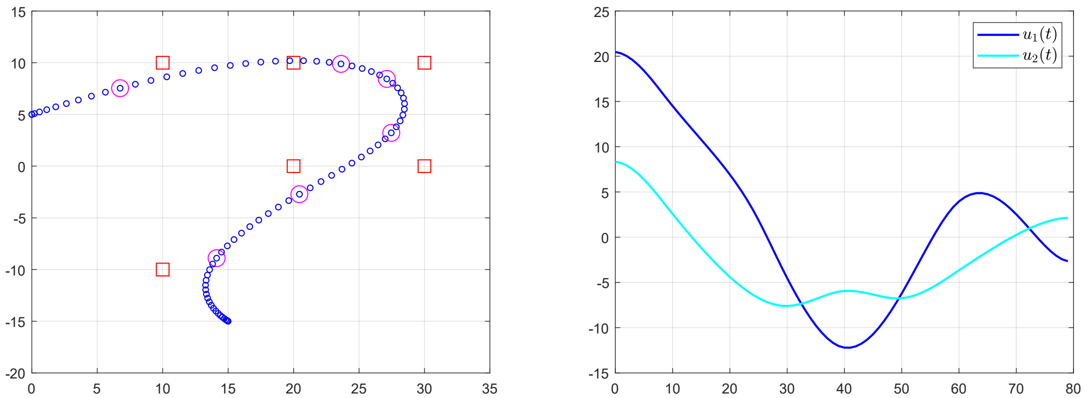
(a) Trajeto do robô (b) Sinal de controlo

Figura 2: Resultados para $\lambda = 10^{-2}$ com o regularizador ℓ_2^2.



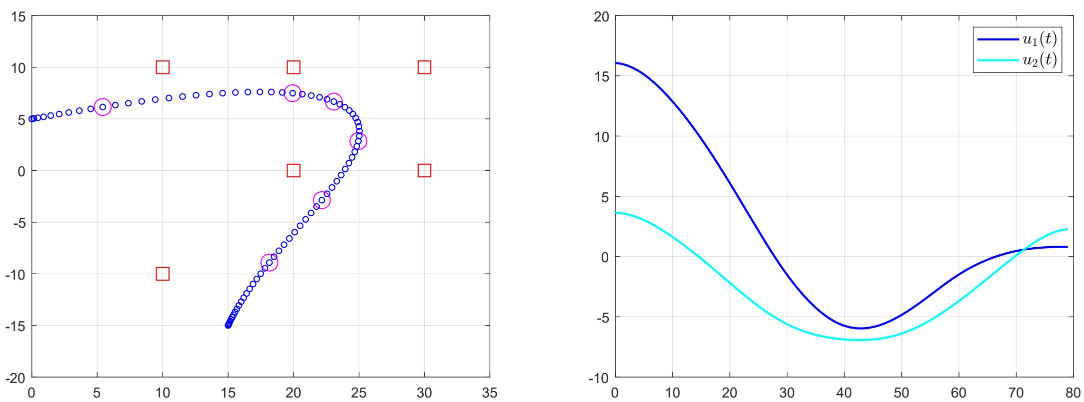
(a) Trajeto do robô (b) Sinal de controlo

Figura 3: Resultados para $\lambda = 10^{-1}$ com o regularizador ℓ_2^2.



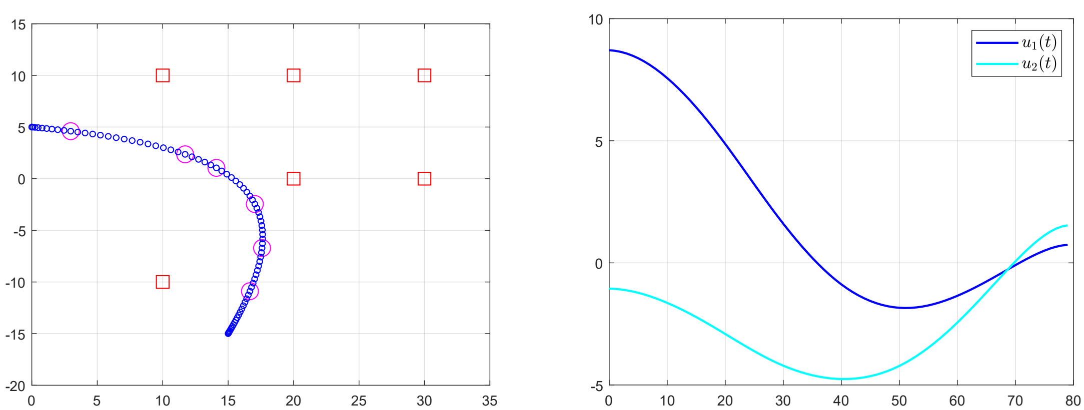
(a) Trajeto do robô (b) Sinal de controlo

Figura 4: Resultados para $\lambda = 1$ com o regularizador ℓ_2^2.



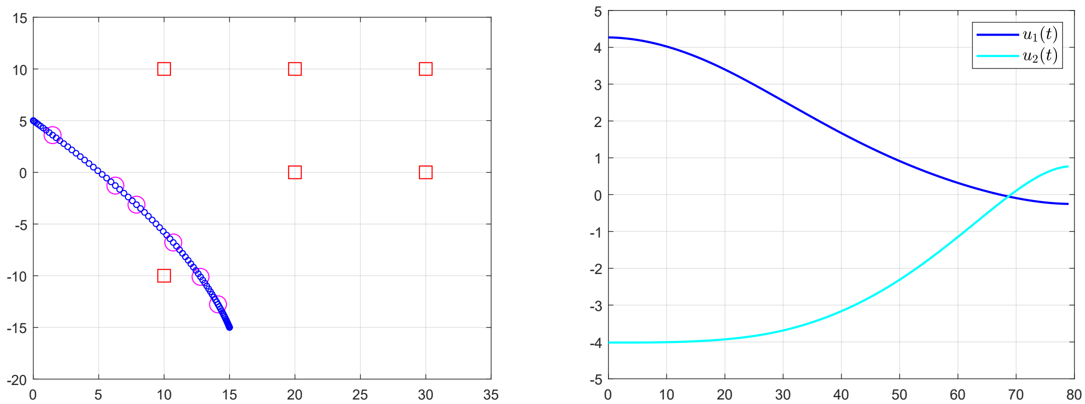
(a) Trajeto do robô (b) Sinal de controlo

Figura 5: Resultados para $\lambda = 10$ com o regularizador ℓ_2^2.



(a) Trajeto do robô (b) Sinal de controlo

Figura 6: Resultados para $\lambda = 10^2$ com o regularizador ℓ_2^2.



(a) Trajeto do robô

(b) Sinal de controlo

Figura 7: Resultados para $\lambda = 10^3$ com o regularizador ℓ_2^2.

Os parâmetros obtidos para o número de comutações no sinal de controlo e o desvio médio aos *waypoints* para cada λ encontram-se na Tabela 1.

λ	Comutações	Desvio médio
10^{-3}	79	0.1257
10^{-2}	79	0.8242
10^{-1}	79	2.1958
1	79	3.6829
10	79	5.6317
10^2	79	10.9042
10^3	79	15.3304

Tabela 1: Número de comutações do sinal de controlo de $t = 0$ a $t = T - 1$ e desvio médio dos *waypoints* nos instantes τ_k, em função de λ. Caso com o regularizador ℓ_2^2.

Pode ser visto em `part1task1.m` o código utilizado para obter os resultados apresentados.

Código 1: `part1task1.m`

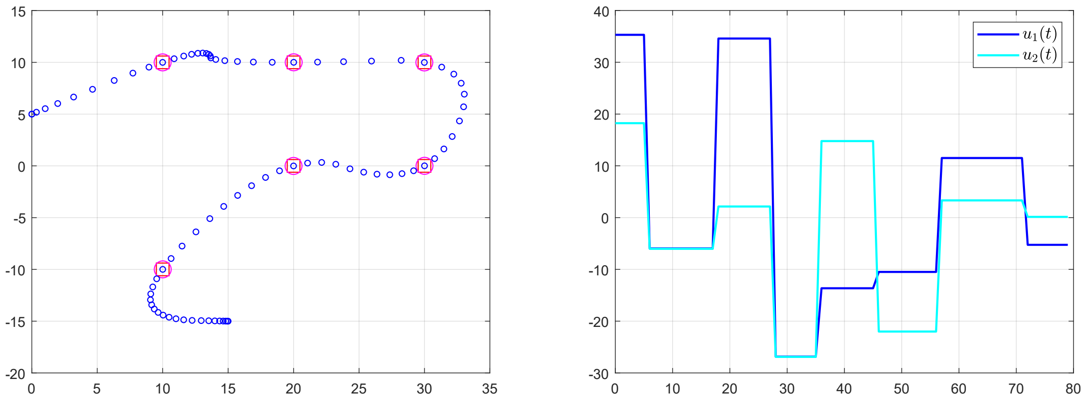
```
%[Part 1 - Task1]
%script that uses cvx tool to solve an optimization
%problem with the l_2^2 regularizor in the cost function

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing
```

```matlab

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 100;
T = 81;
K = 6;

for i = -1%-3:3
    lambda = 10^i;
    cvx_begin quiet
        variables u(2,T-1) x(4,T) %state and control signal are the unknowns
        t = 1:T-1;
        cost = 0;

        for j = 2:T-1
            cost=cost+square_pos(norm(u(:,j)-u(:,j-1),2));
        end
        cost=cost*lambda;
        for j = 1:K
            cost = cost + square_pos(norm(E*x(:,time_wk(j))-wk(:,j), 2));
        end

        minimize(cost);
        %constraints
        subject to
            x(:,1) == x_initial
            x(:,T) == x_final
            for j = 1:T-1
                norm(u(:,j),2) <= Umax
            end
            x(:,t+1) == A*x(:,t) + B*u(:,t)
    cvx_end

    %plot robot positions
    figure;
    plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
    hold on;
    plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
    plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
    grid on;
    axis([0 35 -20 15]);

    %plot control signal
    figure;
    plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
    hold on;
    plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
```

```
    grid on;
    leg = legend('$u_1(t)$', '$u_2(t)$');
    set(leg, 'Interpreter','latex', 'FontSize', 12);

    %point (c)
    control_signal_changes(i+4) = 0;
    for j = 2:T-1
        if norm(u(:,j)-u(:,j-1),2) > 10^(-6)
            control_signal_changes(i+4) = control_signal_changes(i+4) + 1;
        end
    end

    %point (d)
    mean_dev(i+4) = 0;
    for j = 1:K
        mean_dev(i+4) = mean_dev(i+4) +(1/K) *norm(E*x(:,time_wk(j))-wk(:,j)) ;
    end
end
```

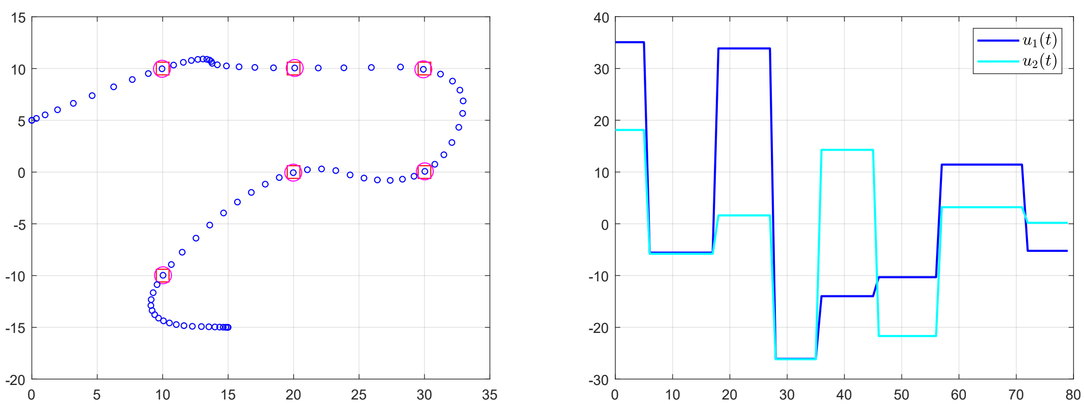
2 Task 2

Analogamente ao apresentado na *Task 1*, as posições ótimas do robô e o sinal de controlo encontrado para satisfazer os quatro desejos, utilizando o regularizador ℓ_2, estão expressos nas figuras 8 – 14.



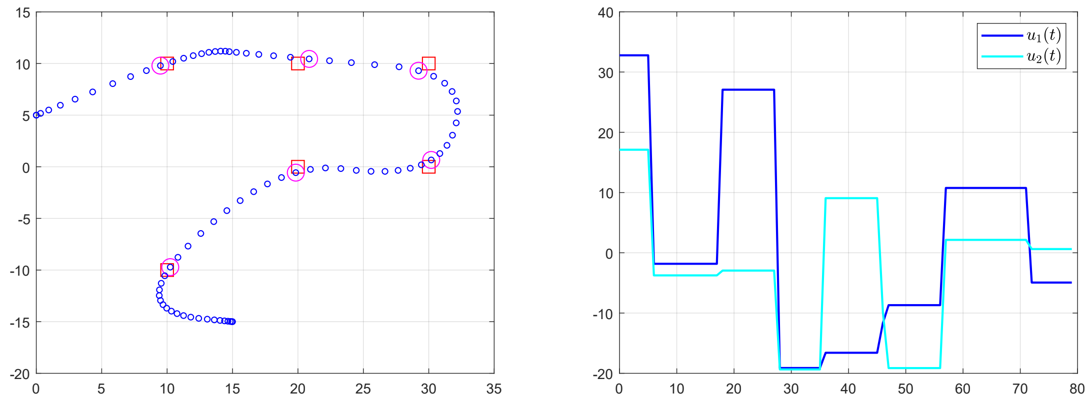
(a) Trajeto do robô (b) Sinal de controlo

Figura 8: Resultados para $\lambda = 10^{-3}$ com o regularizador ℓ_2.



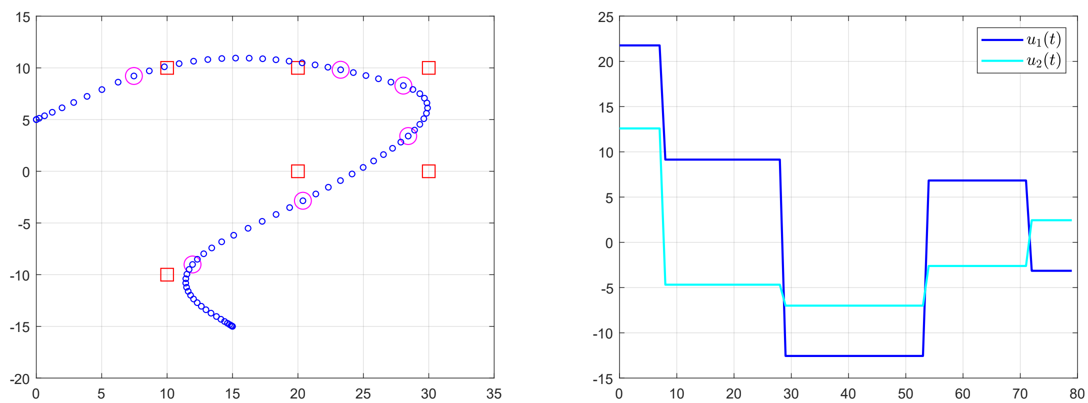
(a) Trajeto do robô (b) Sinal de controlo

Figura 9: Resultados para $\lambda = 10^{-2}$ com o regularizador ℓ_2.



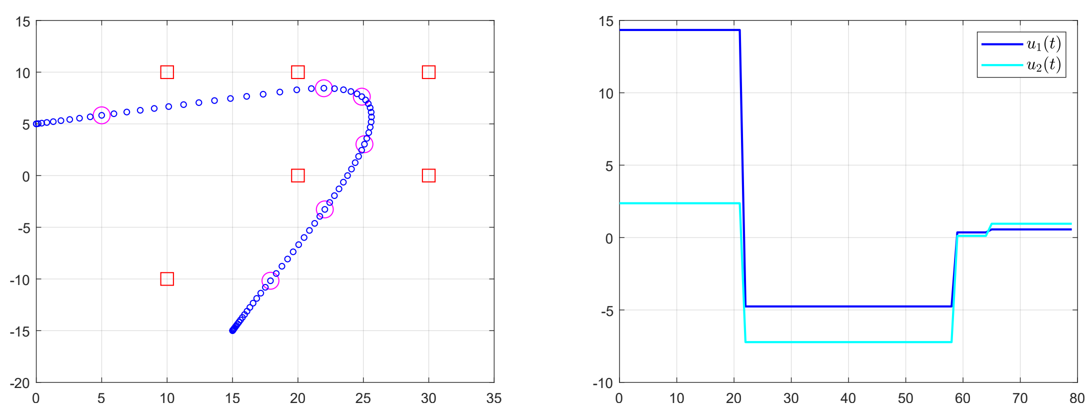
(a) Trajeto do robô (b) Sinal de controlo

Figura 10: Resultados para $\lambda = 10^{-1}$ com o regularizador ℓ_2.



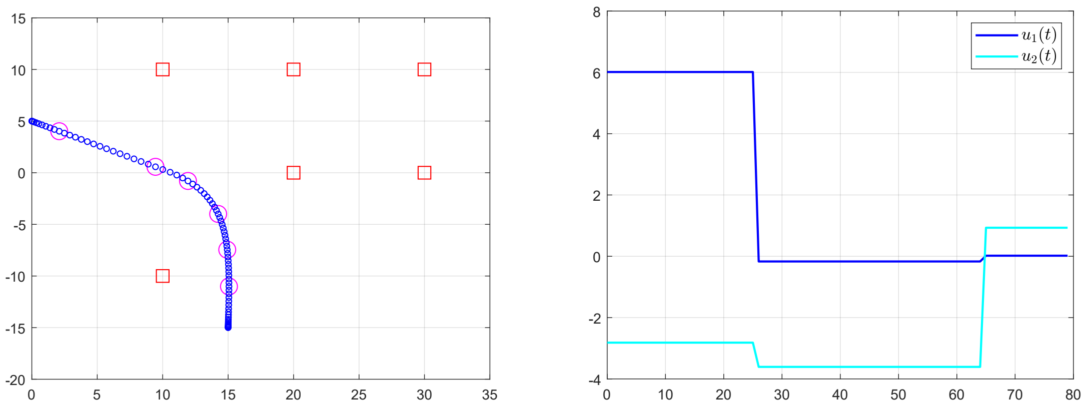
(a) Trajeto do robô (b) Sinal de controlo

Figura 11: Resultados para $\lambda = 1$ com o regularizador ℓ_2.



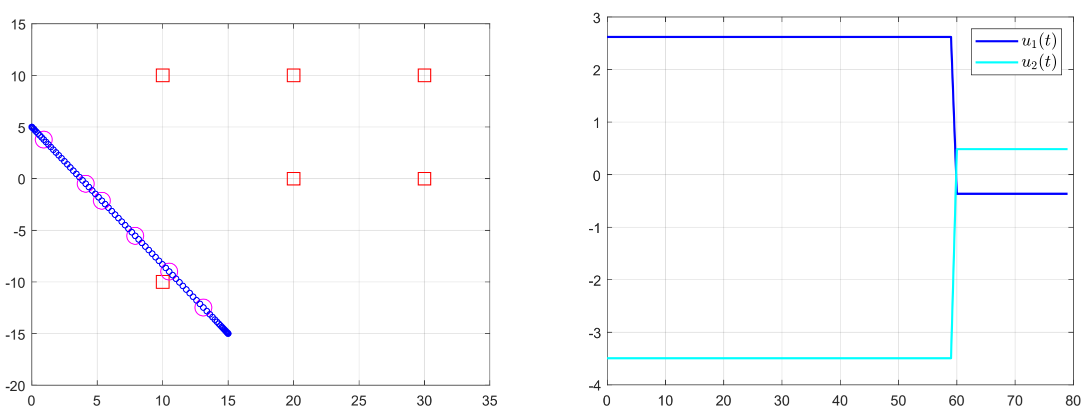
(a) Trajeto do robô (b) Sinal de controlo

Figura 12: Resultados para $\lambda = 10$ com o regularizador ℓ_2.



(a) Trajeto do robô (b) Sinal de controlo

Figura 13: Resultados para $\lambda = 10^2$ com o regularizador ℓ_2.



(a) Trajeto do robô (b) Sinal de controlo

Figura 14: Resultados para $\lambda = 10^3$ com o regularizador ℓ_2.

Os parâmetros obtidos para o número de comutações no sinal de controlo e o desvio médio aos *waypoints* para cada λ encontram-se na Tabela 2.

λ	Comutações	Desvio médio
10^{-3}	9	0.0075
10^{-2}	8	0.0747
10^{-1}	11	0.7021
1	5	2.8877
10	4	5.3689
10^2	4	12.5914
10^3	2	16.2266

Tabela 2: Número de comutações do sinal de controlo de $t = 0$ a $t = T - 1$ e desvio médio dos *waypoints* nos instantes τ_k, em função do λ. Caso com o regularizador ℓ_2.

O código utilizado para obter estes resultados está expresso em `part1task2.m`.

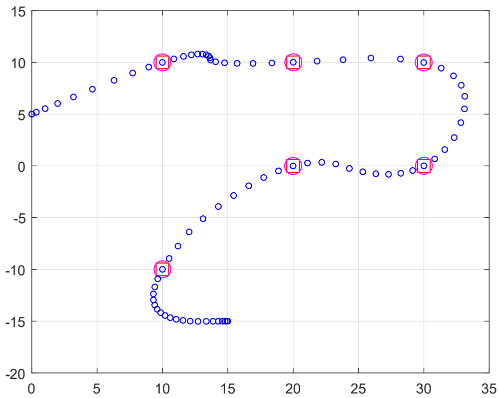
Código 2: `part1task2.m`

```
%[Part 1 - Task2]
%script that uses cvx tool to solve an optimization
%problem with the l_2 regularizor in the cost function

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 100;
T = 81;
K = 6;

for i = -3:3
    lambda = 10^i;
    cvx_begin quiet
        variables u(2,T-1) x(4,T) %state and control signal are the
            unknowns
        t = 1:T-1;
        cost = 0;

        for j = 2:T-1
                cost=cost+norm(u(:,j)-u(:,j-1),2);
```

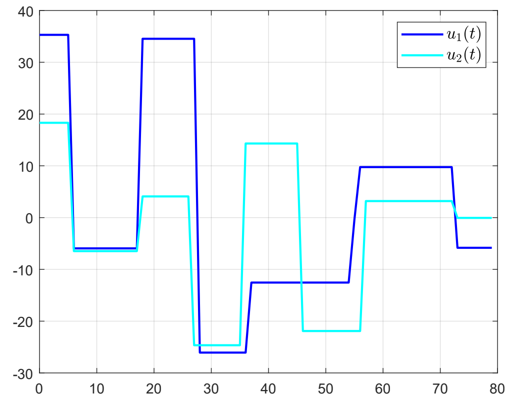
```
         end
        cost=cost*lambda;
        %cost function
        for j = 1:K
           cost = cost + square_pos(norm(E*x(:,time_wk(j))-wk(:,j), 2));
        end
        minimize(cost);
        %constraints
        subject to
            x(:,1) == x_initial
            x(:,T) == x_final
            for j = 1:T-1
                norm(u(:,j),2) ≤ Umax
            end
            x(:,t+1) == A*x(:,t) + B*u(:,t)
    cvx_end

    %plot robot positions
    figure;
    plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
    hold on;
    plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
    plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
    grid on;
    axis([0 35 -20 15]);

    %plot control signal
    figure;
    plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
    hold on;
    plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
    grid on;
    leg = legend('$u_1(t)$', '$u_2(t)$');
    set(leg, 'Interpreter','latex', 'FontSize', 12);

    %point (c)
    control_signal_changes(i+4) = 0;
    for j = 2:T-1
        if norm(u(:,j)-u(:,j-1),2) > 10^(-6)
            control_signal_changes(i+4) = control_signal_changes(i+4) + 1;
        end
    end

    %point (d)
    mean_dev(i+4) = 0;
    for j = 1:K
        mean_dev(i+4) = mean_dev(i+4) +( 1/K) *norm(E*x(:,time_wk(j))-wk(:,j)) ;
    end
end
```

3 Task 3

Da mesma forma que foi efetuado para os outros dois regularizadores, as posições ótimas do robô e o sinal de controlo encontrado para satisfazer os 4 desejos, utilizando o regularizador ℓ_1, estão expressos nas figuras 15 – 21.

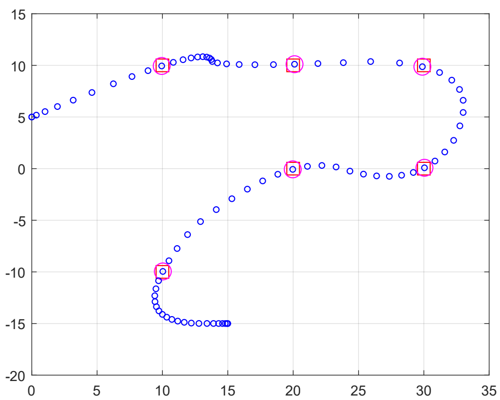


(a) Trajeto do robô

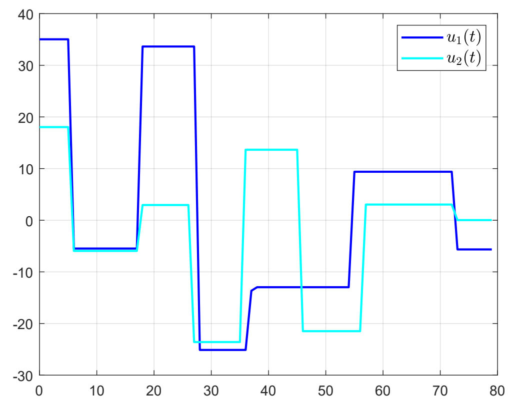


(b) Sinal de controlo

Figura 15: Resultados para $\lambda = 10^{-3}$ com o regularizador ℓ_1.

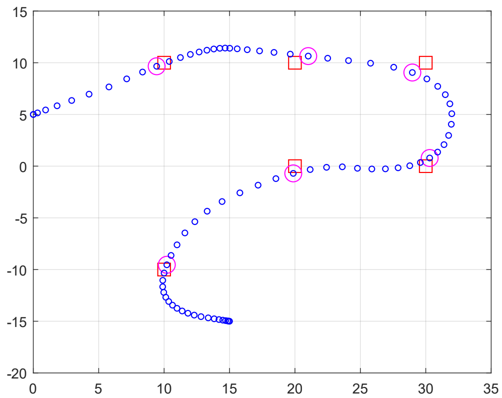


(a) Trajeto do robô

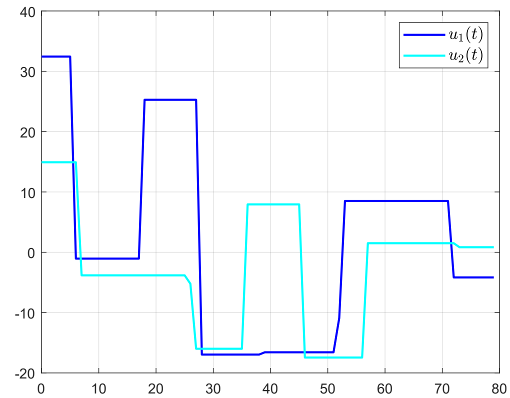


(b) Sinal de controlo

Figura 16: Resultados para $\lambda = 10^{-2}$ com o regularizador ℓ_1.

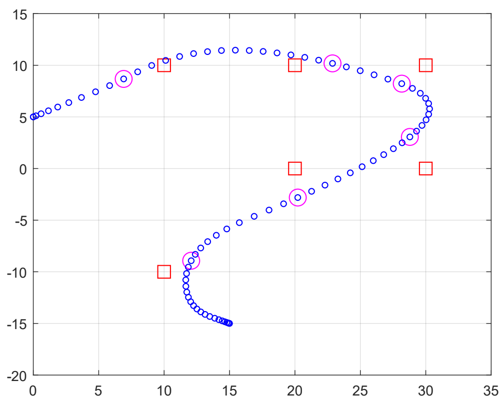


(a) Trajeto do robô

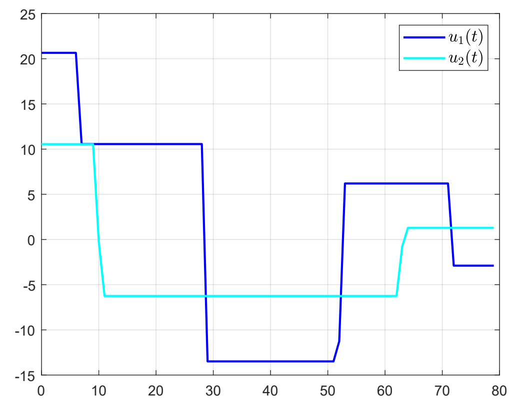


(b) Sinal de controlo

Figura 17: Resultados para $\lambda = 10^{-1}$ com o regularizador ℓ_1.

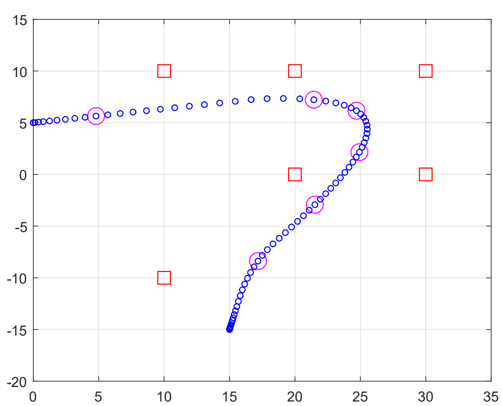


(a) Trajeto do robô

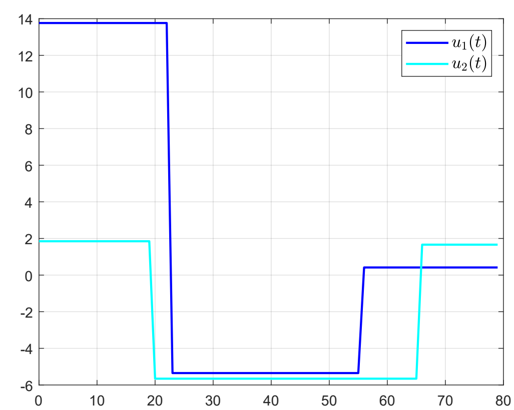


(b) Sinal de controlo

Figura 18: Resultados para $\lambda = 10^{0}$ com o regularizador ℓ_1.

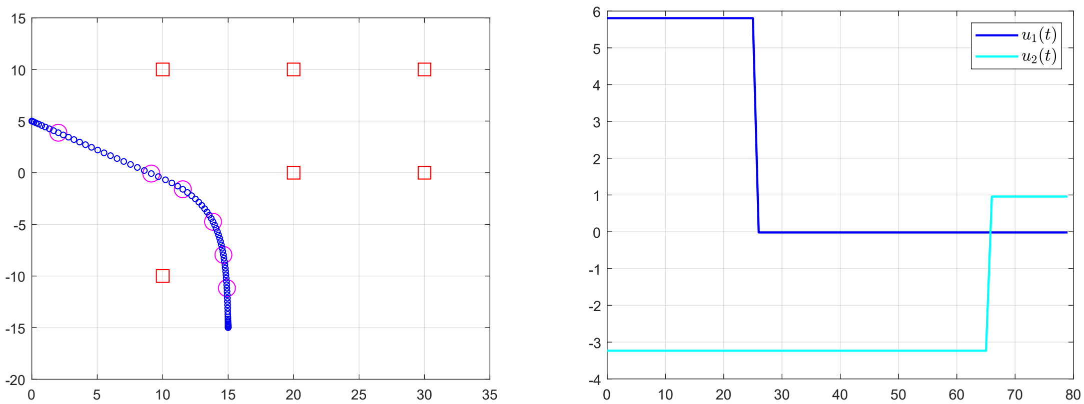


(a) Trajeto do robô



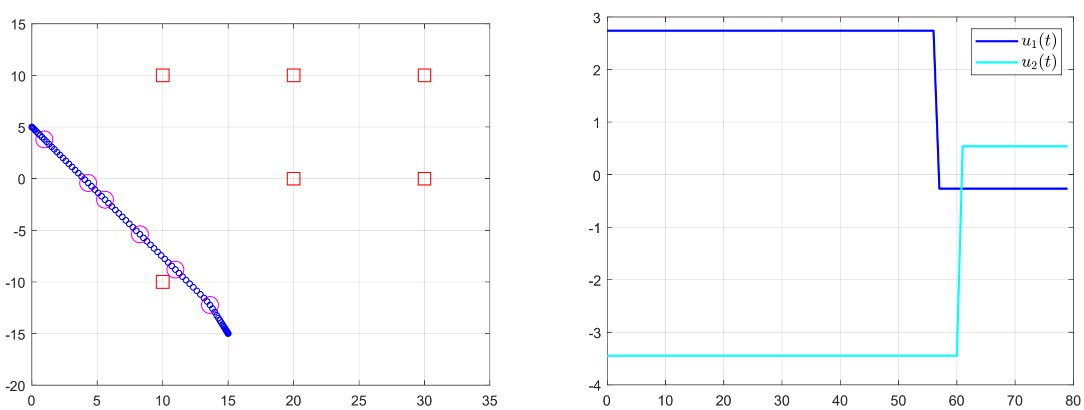
(b) Sinal de controlo

Figura 19: Resultados para $\lambda = 10^{1}$ com o regularizador ℓ_1.



(a) Trajeto do robô (b) Sinal de controlo

Figura 20: Resultados para $\lambda = 10^2$ com o regularizador ℓ_1.



(a) Trajeto do robô (b) Sinal de controlo

Figura 21: Resultados para $\lambda = 10^3$ com o regularizador ℓ_1.

Os parâmetros obtidos para o número de comutações no sinal de controlo e o desvio médio aos *waypoints* para cada λ encontram-se na Tabela 3.

λ	Comutações	Desvio médio
10^{-3}	12	0.0107
10^{-2}	12	0.1054
10^{-1}	14	0.8863
1	11	2.8732
10	5	5.4361
10^{2}	3	13.0273
10^{3}	2	16.0463

Tabela 3: Número de comutações do sinal de controlo de $t = 0$ a $t = T - 1$ e desvio médio dos *waypoints* nos instantes τ_k, em função do λ. Caso com o regularizador ℓ_1.

Encontra-se em `part1task3.m` o código utilizado para obter estes resultados.

Código 3: `part1task3.m`

```
%[Part 1 - Task3]
%script that uses cvx tool to solve an optimization
%problem with the l_1 regularizor in the cost function

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 100;
T = 81;
K = 6;

for i = -3:3
    lambda = 10^i;
    cvx_begin quiet
        variables u(2,T-1) x(4,T) %state and control signal are the
            unknowns
        t = 1:T-1;
        cost = 0;

         for j = 2:T-1
                 cost=cost+norm(u(:,j)-u(:,j-1),1);
```

```
             end
            cost=cost*lambda;
            %cost function
            for j = 1:K
               cost = cost + square_pos(norm(E*x(:,time_wk(j))-wk(:,j), 2));
            end
            minimize(cost);
            %constraints
            subject to
                x(:,1) == x_initial
                x(:,T) == x_final
                for j = 1:T-1
                    norm(u(:,j),2) ≤ Umax
                end
                x(:,t+1) == A*x(:,t) + B*u(:,t)
        cvx_end

        %plot robot positions
        figure;
        plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
        hold on;
        plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
        plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
        grid on;
        axis([0 35 -20 15]);

        %plot control signal
        figure;
        plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
        hold on;
        plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
        grid on;
        leg = legend('$u_1(t)$', '$u_2(t)$');
        set(leg, 'Interpreter','latex', 'FontSize', 12);

        %point (c)
        control_signal_changes(i+4) = 0;
        for j = 2:T-1
            if norm(u(:,j)-u(:,j-1),2) > 10^(-6)
                control_signal_changes(i+4) = control_signal_changes(i+4) + 1;
            end
        end

        %point (d)
        mean_dev(i+4) = 0;
        for j = 1:K
            mean_dev(i+4) = mean_dev(i+4) +( 1/K) *norm(E*x(:,time_wk(j))-wk(:,j)) ;
        end
    end
```

4 Task 4

Antes de mais, faz sentido referir que o parâmetro regularizador $\lambda > 0$ define a "importância" relativa entre cada um dos desejos 3 (proximidade aos *waypoints* nos tempos definidos da trajetória) e 4 (comutação no sinal de controlo) do problema, sendo que quanto maior for λ, mais priorizado será o desejo 4 (e consequentemente menos o desejo 3) e vice-versa. Assim sendo, para todos os regularizadores, um aumento do valor de λ leva a um aumento do desvio médio aos *waypoints*. Por outro lado, um aumento do valor de λ leva a uma diminuição do número de comutações do sinal de controlo, exceto para a formulação ℓ_2^2. Para este caso, o número de comutações do sinal de controlo manteve-se em 79 comutações para todos os valores de λ. Ou seja, o sinal comutou em todos os instantes analisados, independentemente do valor deste peso.

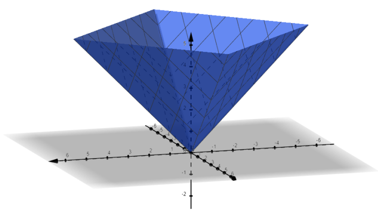
Ora, o regularizador ℓ_2^2 apresenta uma maior sensibilidade por os seus termos aparecerem o quadrado: esta norma tem o formato de uma parábola. Como tal, variações mais bruscas são muito mais penalizadas, ao passo que pequenas variações são atenuadas, o que não acontece nos outros dois casos. Assim, esta norma leva o sinal de controlo a ser contínuo, pois é mais custoso passar diretamente de, por exemplo, 0 para 3, do que ir aumentando gradualmente $0 \to 1 \to 2 \to 3$, pelo que o sinal de controlo irá variar gradualmente, evitando-se variações bruscas. Nota-se, pelos gráficos do seu sinal de controlo, que este varia com o formato de uma sinusoide.

Este regularizador, por apresentar variações contínuas, tem um sinal de controlo mais "suave", quando comparado com os restantes, que apresentam menos comutações. Por esta razão, com o regularizador ℓ_2^2 dá-se menos liberdade ao movimento do robô (para este fazer uma travagem brusca, por exemplo, acabando estas por ser sempre mais suaves que nos outros casos). Acrescenta-se ainda que o regularizador ℓ_2^2, por restringir esta liberdade do robô e por valorizar de tal forma a suavidade do sinal de controlo, acaba por levar o robô a passar mais longe dos *waypoints* quando comparado com os outros regularizadores.

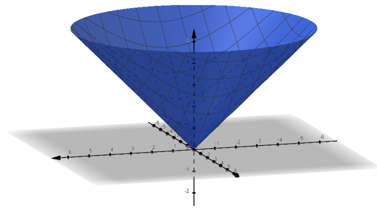
Focando nas tabelas $1-3$, nota-se que os regularizadores ℓ_2 e ℓ_1 apresentam valores muito inferiores de comutações no sinal de controlo, não têm a mesma distinção entre variações bruscas e suaves, isto é, não são tão sensíveis a variações mais bruscas. Consequentemente, estes dois regularizadores conseguem dar uma maior importância ao termo da função de minimização que dita a distância aos *waypoints*, conseguindo minimizar mais esta componente, pois o sinal de controlo pode variar apenas quando necessário.

De modo a comparar agora os regularizadores ℓ_1 e ℓ_2, refere-se que, para o caso genérico da norma ℓ_1, ao se caminhar de um ponto até outro, havendo variação de ambas as coordenadas, nunca se caminha na diagonal: primeiro caminha-se apenas numa coordenada e depois na outra. Assim, ℓ_1 só permite a alteração de uma coordenada de cada vez. Isto justifica os resultados observados nos gráficos do sinal de controlo: para o caso de ℓ_1, $u_1(t)$ e $u_2(t)$ não variam ao mesmo tempo, variando sempre um primeiro que o outro. Por outro lado, no outro caso, quando uma componente varia, a outra varia também (nesta norma, utilizando o mesmo exemplo, o percurso é feito na diagonal). Este resultado justifica que o número de comutações que o regularizador ℓ_1 apresenta sejam superiores aos do ℓ_2. No entanto, note-se que estes números de comutações não podem ser diretamente comparados, pois, por se querer um sinal de controlo que seja *piecewise constant*, é preferível que este comute apenas uma das suas coordenadas de cada vez e não as duas simultaneamente. Com esta perspetiva, a norma ℓ_2 acaba por promover mais comutações (pois, em cada uma, ambas as coordenadas variam).

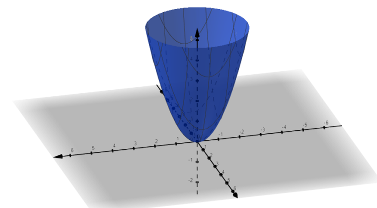
Finalmente, fez-se uma análise dos diversos regularizadores a nível computacional. Para esta análise é importante a observação dos gráficos das suas superfícies, que podem ser visualizados na figura 22.



(a) l_1



(b) l_2



(c) l_2^2

Figura 22: Superfícies para os três regularizadores – gráficos efetuados no *Geogebra*

A norma ℓ_2^2 tem uma forma parabólica que leva o gradiente a diminuir – tendendo para zero – à medida que as iterações se aproximam do mínimo, o que leva os passos a ter um tamanho cada vez mais reduzido. Assim, esta apresentará soluções subótimas pois o método acaba por parar antes de se chegar ao mínimo da função por o gradiente se ir, rapidamente, aproximando de 0. Nos outros dois casos, devido à forma apresentada pelas suas superfícies, o gradiente é sempre constante exceto na origem, encontrando-se o mínimo quando o gradiente muda o sentido para onde aponta. Por outro lado, devido à norma do gradiente se manter constante, os regularizadores ℓ_1 e ℓ_2 resultam em tempos de execução muito semelhantes e muito melhores que a norma ℓ_2^2. Note-se que, no entanto, a norma ℓ_2^2 seria a de resolução mais fácil, por ser um problema de mínimos quadrados, confirmando-se que o MATLAB não é capaz de o detetar.

Conclui-se que a norma ℓ_2^2 é, claramente, a que produz piores resultados no contexto deste problema, em que se visa tem um sinal de controlo que seja constante. Quanto às outras, por os erros cometidos serem muito semelhantes e como se pretende que cada componente do sinal de controlo se mantenha constante o maior tempo possível, escolhe-se a ℓ_1 por ser aquela que preserva melhor este sinal, comutando apenas uma parte do sinal de cada vez.

5 Task 5

Agora, pretende-se que o robô passe por certas regiões intermediárias, chamadas de discos, $D(c,r)$, representados na forma $\{x \in \mathbf{R}^2 : \|x-c\|_2 \leq r\}$, com centro $c \in \mathbf{R}^2$ e raio r.

Uma vez que não importa o quão perto do centro do disco o robô se encontra, e apenas o quanto se encontra perto da sua fronteira, define-se a distância do ponto $p \in \mathbf{R}^2$ até ao disco $(d\,(p, D(c,r)))$ como: $d\,(p, D(c,r)) = \|p-c\|_2 - r$, com $p, c \in \mathbf{R}^2$.

No entanto como se pode verificar na figura 23, se o resultado desta distância for negativo, quer dizer que o ponto p se encontra dentro do disco, pelo que tem de se impor uma restrição: quando o ponto se encontra num ponto intrínseco ao disco, ou seja, o resultado de $\|p-c\|_2-r$ é negativo, tem-se $d\left(p, D(c, r)\right)=0$.

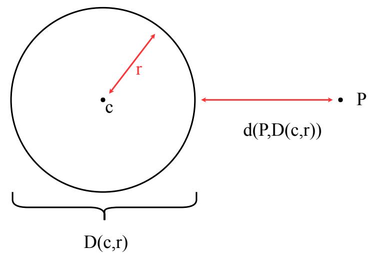


Figura 23: Representação da distância do ponto p ao disco D(c,r)

Condensando os dois casos, conclui-se que:

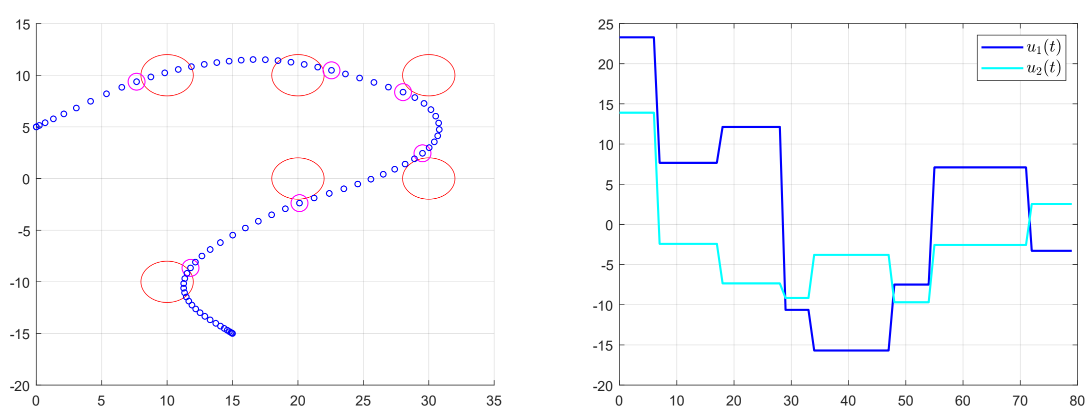
$$d\left(p, D(c, r)\right)=\begin{cases}0 & \text{se } \|p-c\|_2 \leq r \\ \|p-c\|_2 & \text{se } \|p-c\|_2>r\end{cases}$$

Assim, pode-se escrever a expressão em forma fechada como:

$$d\left(p, D(c, r)\right)=\max\left\{0,\|p-c\|_2-r\right\}.$$

6 Task 6

As posições ótimas encontradas de modo a que o robô passe nos instantes τ_k o mais perto possível do disco $D(c_k, r_k)$, bem como o sinal de controlo ótimo de $t=0$ a $t=T$, estão representadas na figura 24. Verificou-se que o desvio médio aos *waypoints* é 2.4483 e que o sinal de controlo comuta sete vezes.



(a) Trajeto do robô

(b) Sinal de controlo

Figura 24: Resultados para $\lambda=10^{-1}$ com o regularizador ℓ_2.

6.1 Comentário aos resultados, comparando com a Task 2 para o caso em que $\lambda = 10^{-1}$

Para ambos os casos, quanto maior o número de variações do sinal de controlo, mais perto dos *waypoints* (ou discos) o robô consegue passar, pelo que tem de haver um balanço entre estes objetivos, tentando minimizar o conjunto dos dois desejos.

A tabela 4 compara os resultados obtidos para as duas diferentes situações.

Task	Comutações	Desvio médio
2	11	0.7021
6	7	2.4483

Tabela 4: Comparação de valores das Tasks 2 e 6 para o caso $\lambda = 10^{-1}$

Observa-se que o sinal de controlo para o caso da *Task 6* mudou menos vezes que o da *Task 2* e que o desvio médio aumentou bastante. Com o aumento do raio do disco, a função de custo atribui um peso inferior ao termo que representa a distância a que o robô passa do disco: $\sum_{k=1}^{K} d\left(Ex(\tau_k), D(c_k, r_k)\right)^2$. Quando o raio é igual a 0 (*Task 2*), o peso atribuído a este termo é superior, daí o desvio médio ser inferior, no entanto o termo do sinal de controlo apresenta um custo superior, pois são necessárias várias variações deste sinal.

Detalhando de uma forma mais intuitiva, para o caso da *Task 6*, o custo associado à proximidade ao disco é nulo em qualquer ponto intrínseco ao disco (ao passo que no caso da *Task 2* só é nulo exatamente no *waypoint*). Por haver uma maior cobertura de área com custo nulo na *Task 6*, existe uma redução do custo associado ao sinal de controlo simples, pois o robô não está sujeito a tantas mudanças bruscas para se aproximar dos pontos de custo de proximidade nula, em comparação com o outro caso.

Quanto ao desvio médio do robô aos *waypoints*, é evidente que este resultado é superior para o caso em que se usam discos, pois o ponto de comparação para este desvio é o centro do disco, quando na verdade para que o custo de proximidade seja zero, o robô necessita apenas de intersetar a fronteira do disco, acabando assim por conseguir atingir o mesmo custo de proximidade que a outra solução, passando mais afastado do centro.

Como se pode observar pela figura 24, o robô não chega a intersetar o disco em nenhum ponto, pois compensa mais reduzir as variações do sinal de controlo, deixando o custo associado à proximidade do disco baixo, mas não nulo, garantindo assim que a conjugação de proximidade com o disco e desvios no sinal de controlo desça o custo da função.

Conclui-se, assim, que, aumentando o raio do disco, existe um maior número de pontos com custo de proximidade zero, conseguindo-se, através da atribuição de maior peso, otimizar bastante o custo do sinal de controlo, obtendo uma solução em que o robô passa muito perto da fronteira de todos os discos, com um sinal de controlo com menos desvios.

Encontra-se em `part1task6.m`, o código utilizado para obter os resultados da *Task 6*.

Código 4: `part1task6.m`

```
%[Part 1 - Task6]
%script that uses cvx tool to solve an
%optimization problem to get a trajectory to
%pass as close as possible to some disks in space

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing


%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 100;
T = 81;
K = 6;
lambda=0.1;
radiuses=2;

cvx_begin quietxbest
    variables u(2,T-1) x(4,T) %state and control signal are the unknowns
    t = 1:T-1;
    cost = 0;

     for j = 2:T-1
            cost=cost+norm(u(:,j)-u(:,j-1),2);
     end
     cost=cost*lambda;
    %cost function

    for i = 1:K
       cost = cost + square_pos(norm(E*x(:,time_wk(i))-wk(:,i), 2)-radiuses
           );
    end
    minimize(cost);
    %constraints
    subject to
        x(:,1) == x_initial
        x(:,T) == x_final
        for i = 1:T-1
            norm(u(:,i),2) ≤ Umax
        end
        x(:,t+1) == A*x(:,t) + B*u(:,t)
cvx_end

```

```
%plot robot positions
figure;
hold on;
n = 1000;
tc = linspace(0,2*pi,n);
for k=1:K
    xc = wk(1,k) + radiuses*sin(tc);
    yc = wk(2,k) + radiuses*cos(tc);
    line(xc,yc, 'Color', 'red');
end

%plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', '
    MarkerSize',12);
plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
grid on;
axis([0 35 -20 15]);

%plot control signal
figure;
plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
hold on;
plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
grid on;
leg = legend('$u_1(t)$', '$u_2(t)$');
set(leg, 'Interpreter','latex', 'FontSize', 12);


%point (c)
control_signal_changes = 0;
for j = 2:T-1
    if norm(u(:,j)-u(:,j-1),2) > 10^(-6)
        control_signal_changes = control_signal_changes + 1;
    end
end
control_signal_changes

%point (d)
mean_dev = 0;
for j = 1:K
    mean_dev = mean_dev +( 1/K) *norm(E*x(:,time_wk(j))-wk(:,j)) ;
end
mean_dev
```

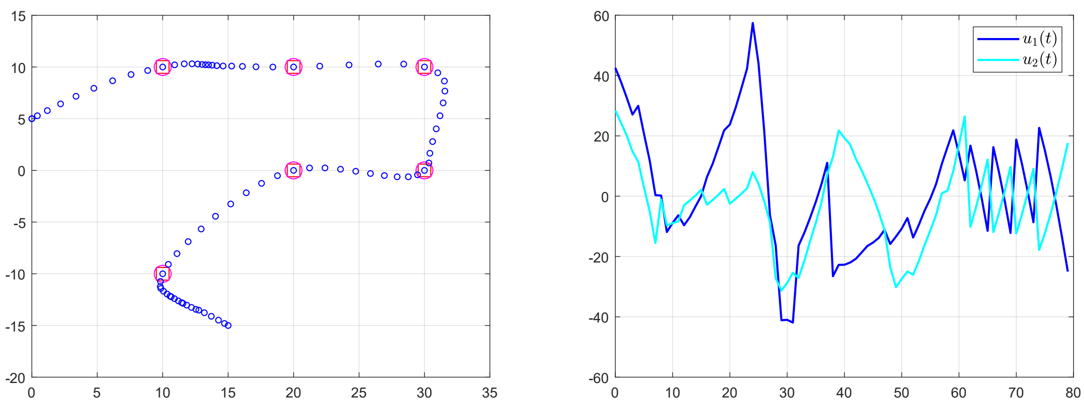
7 Task 7

De modo a forçar que o robô passe por todos os pontos w_k nos instantes τ_k, porque não igualar a função de custo a zero e adicionar esse desejo nas restrições? De notar que a única restrição aplicada ao sinal de controlo é que a sua magnitude máxima é $U_{max} = 15$, não havendo qualquer restrição quanto às variações que este pode sofrer, desde que dentro dos limites de magnitude possíveis.

Verifica-se que não existe nenhuma solução para o problema pretendido, isto é, não existe um sinal de controlo com magnitude máxima $U_{max} = 15$ que faça o robô passar por todos os K pontos nos instantes τ_k, mesmo que sofra quaisquer alterações no sinal de controlo dentro do intervalo de magnitude possível. Apesar do sinal de controlo ter toda a liberdade para alterar o número de vezes que pretender, o seu *range* não é suficiente para que consiga levar o robô a passar em todos os pontos – o robô é lento. Como tal, tanto a variável x como u apresentam o valor NaN. A variável *cvx optval* tem o valor de infinito neste caso – indicativo de que não existe solução.

Realça-se que utilizando um motor que apresenta maior magnitude, $U_{max} = 100$, é possível arranjar uma solução para o problema, pois restringindo a magnitude máxima a um valor superior, consegue-se que o sinal de controlo "impulsione" mais o robô, conseguindo este passar por todos os *waypoints* nos instantes pretendidos. O robô já não é tão lento, conseguindo ter variações do sinal num *range* superior.

A figura 25 ilustra os resultados obtidos para este caso, podendo-se observar que foram efetuadas muitas comutações no sinal de controlo entre valores muito superiores, em módulo, a 15, para atingir o objetivo de passar em todos os pontos.



(a) Trajeto do robô (b) Sinal de controlo

Figura 25: Resultados quando o robô passa exatamente nos *waypoints* para $U_{max} = 100$.

Encontra-se, em `part1task7.m`, o código utilizado para obter os resultados da *Task 7*.

Código 5: `part1task7.m`

```
%[Part 1 — Task7]
%script that uses cvx tool to solve an
%optimization problem to get a trajectory
%that aims to intersect some disks in space

clear;

```

```
A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing


%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 15;
T = 81;
K = 6;
lambda=0.1;
radiuses=2;

cvx_begin quiet
    variables u(2,T-1) x(4,T) %state and control signal are the unknowns
    t = 1:T-1;

    minimize(0);
    %constraints
    subject to
        x(:,1) == x_initial
        x(:,T) == x_final
        for i = 1:T-1
            norm(u(:,i),2) <= Umax
        end
        x(:,t+1) == A*x(:,t) + B*u(:,t)

        for k=1:K
            E*x(:,time_wk(k))==wk(:,k)
        end


cvx_end

%plot robot positions
figure;
plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
hold on;
plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
grid on;
axis([0 35 -20 15]);

%plot control signal
figure;
plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
```

```
hold on;
plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
grid on;
leg = legend('$u_1(t)$', '$u_2(t)$');
set(leg, 'Interpreter','latex', 'FontSize', 12);
```

8 Task 8

Uma função $f : \mathbf{R}^n \to \mathbf{R}$ é convexa se e só se

$$f((1-\alpha)x + \alpha y) \leq (1-\alpha)f(x) + \alpha f(y)$$
$$\text{para cada } x \in \mathbf{R}^n, y \in \mathbf{R}^n \text{e } \alpha \in [0,1]$$

Uma vez que não se consegue concluir através da análise gráfica, para mostrar que a função $\phi(x)$ é não convexa recorre-se à definição. Fixando $x = [-1,-1], y = [0,0]$, com $x, y \in \mathbf{R}^2$ e $\alpha = 0.5$, obtém-se:

$$f(0.5x + 0.5y) \leq 0.5f(x) + 0.5f(y)$$
$$f(0.5x) \leq 0.5f(x)$$

Concluindo-se assim que para função $\phi(x)$ se obtém:

$$\phi(0.5x) \leq 0.5\phi(x), x \neq (0,0)$$
$$1 \leq 0.5$$

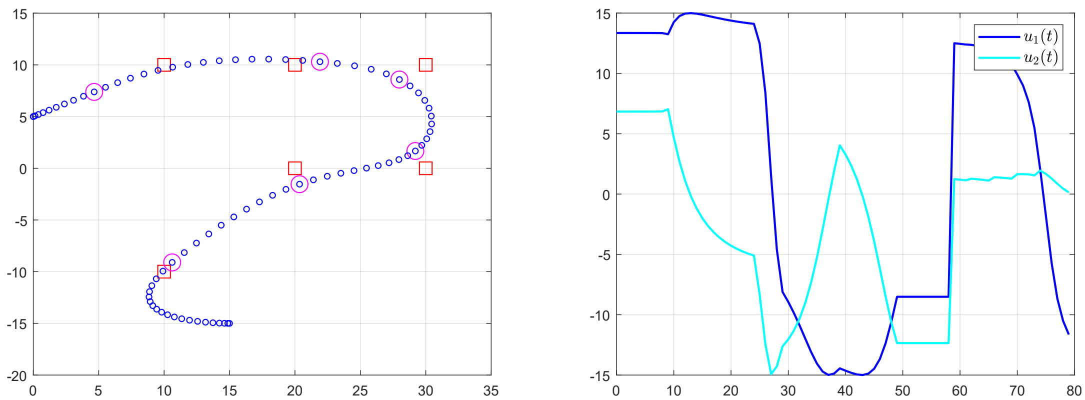
Como se viola a condição de convexidade, verifica-se que a função $\phi(x)$ é uma função não convexa.

9 Task 9

A função não convexa da *Task 8* pretende contar o número de *waypoints* que o robô falha. Pretende-se agora formular este problema com uma função convexa.

As posições ótimas do robô e o sinal de controlo, utilizando o regularizador ℓ_2^2, estão expressos na figura 26, podendo-se comprovar que não é capturado nenhum ponto de rota.

Encontra-se, em `part1task9.m`, o código utilizado para obter os resultados da *Task 9*.



(a) Trajeto do robô (b) Sinal de controlo

Figura 26: Resultados para a formulação ℓ_2^2.

Código 6: `part1task9.m`

```
%[Part 1 — Task9]
%script that uses cvx tool to solve an
%optimization problem with a l_2^2 formulation

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 —10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;—15;0;0];

Umax = 15;
T = 81;
K = 6;

cvx_begin quiet
    variables u(2,T—1) x(4,T) %state and control signal are the unknowns
    t = 1:T—1;
    cost = 0;
    %cost function
    for i = 1:K
       cost = cost + square_pos(norm(E*x(:,time_wk(i))—wk(:,i), 2));
    end
    minimize(cost);
    %constraints
    subject to
        x(:,1) == x_initial
```

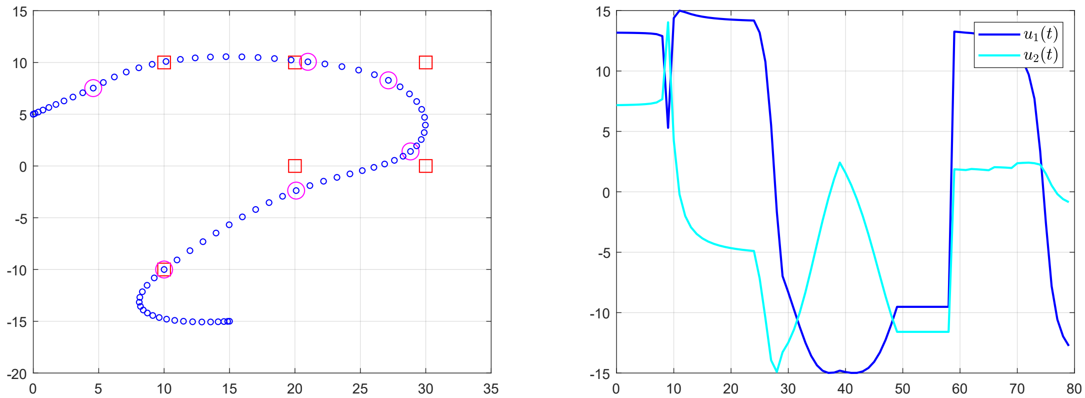
```
        x(:,T) == x_final
        for i = 1:T-1
            norm(u(:,i),2) <= Umax
        end
        x(:,t+1) == A*x(:,t) + B*u(:,t)
cvx_end

%plot robot positions
figure;
plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
hold on;
plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
grid on;
axis([0 35 -20 15]);

%plot control signal
figure;
plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
hold on;
plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
grid on;
leg = legend('$u_1(t)$', '$u_2(t)$');
set(leg, 'Interpreter','latex', 'FontSize', 12);

%captured waypoint
captured = 0;
for i=1:K
    if(norm(E*x(:,time_wk(i)) - wk(:,i)) <= 10^(-6))
        captured = captured + 1;
    end
end
```

10 Task 10

As posições ótimas do robô e o sinal de controlo, utilizando o regularizador ℓ_2, estão expressos na figura 27, podendo-se comprovar que, agora, passa a haver um *waypoint* capturado.

Nota-se que os gráficos do sinal de controlo presentes nas figuras 26 e 27 são semelhantes em todos os instantes de tempo, exceto na proximidade de $t = 10$ (onde o do regularizador ℓ_2 captura um ponto). Isto deve-se ao facto já mencionado e representado na figura 22: com a norma ℓ_2^2 atingem-se soluções subótimas, ao passo que com o ℓ_2 consegue-se atingir mesmo o mínimo, sendo o suficiente para o regularizador ℓ_2^2 não conseguir capturar esse ponto.

Encontra-se, em `part1task10.m`, o código utilizado para obter os resultados da Task10.



(a) Trajeto do robô (b) Sinal de controlo

Figura 27: Resultados para a formulação ℓ_2.

Código 7: `part1task10.m`

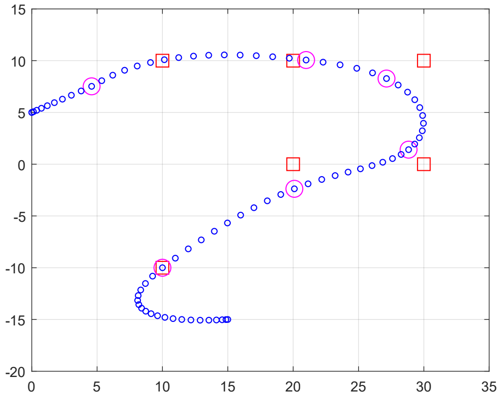
```
%[Part 1 - Task10]
%script that uses cvx tool to solve an
%optimization problem with a l_2 formulation

clear;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61]; %add one because of matlab indexing

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 15;
T = 81;
K = 6;

cvx_begin quiet
    variables u(2,T-1) x(4,T) %state and control signal are the unknowns
    t = 1:T-1;
    cost = 0;
    %cost function
    for i = 1:K
       cost = cost + norm(E*x(:,time_wk(i))-wk(:,i), 2);
    end
    minimize(cost);
    %constraints
    subject to
        x(:,1) == x_initial
```

```
        x(:,T) == x_final
        for i = 1:T-1
            norm(u(:,i),2) <= Umax
        end
        x(:,t+1) == A*x(:,t) + B*u(:,t)
cvx_end

%plot robot positions
figure;
plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
hold on;
plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta', 'MarkerSize',12);
plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
grid on;
axis([0 35 -20 15]);

%plot control signal
figure;
plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
hold on;
plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
grid on;
leg = legend('$u_1(t)$', '$u_2(t)$');
set(leg, 'Interpreter','latex', 'FontSize', 12);

%captured waypoint
captured = 0;
for i=1:K
    if(norm(E*x(:,time_wk(i)) - wk(:,i)) <= 10^(-6))
        captured = captured + 1;
    end
end
```

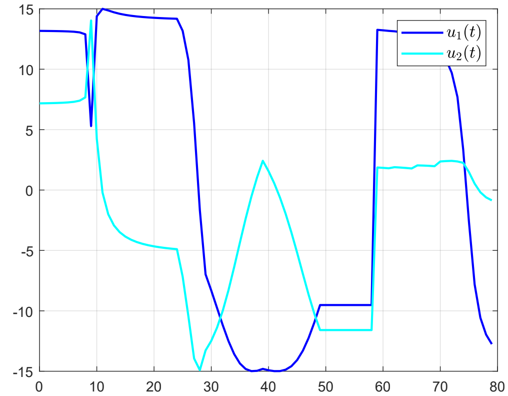
11 Task 11

Introduzindo pesos na função de custo, obtêm-se as figuras 28 a 37, cujos números de pontos de rota capturados se encontram na tabela 5

Encontra-se, em `part1task11.m`, o código utilizado para obter os resultados da *Task 11*.

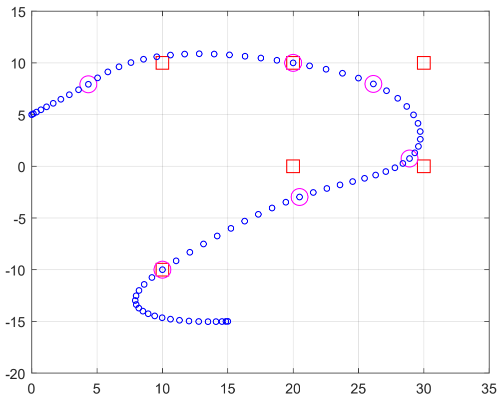


(a) Trajeto do robô

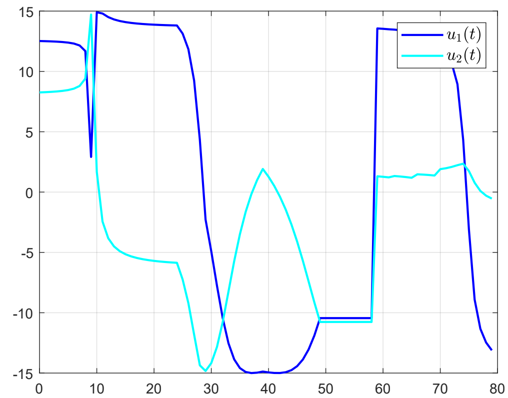


(b) Sinal de controlo

Figura 28: Resultados para $m = 0$.

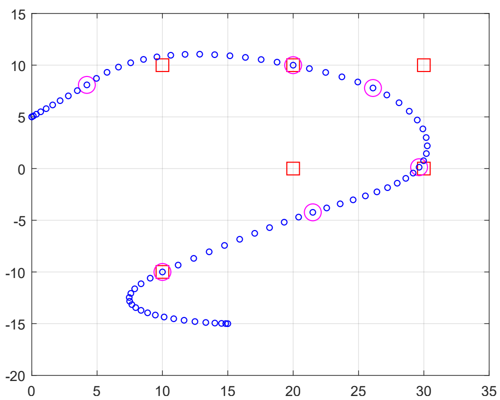


(a) Trajeto do robô

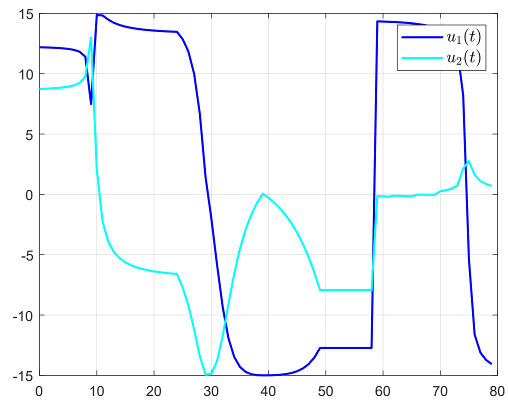


(b) Sinal de controlo

Figura 29: Resultados para $m = 1$.

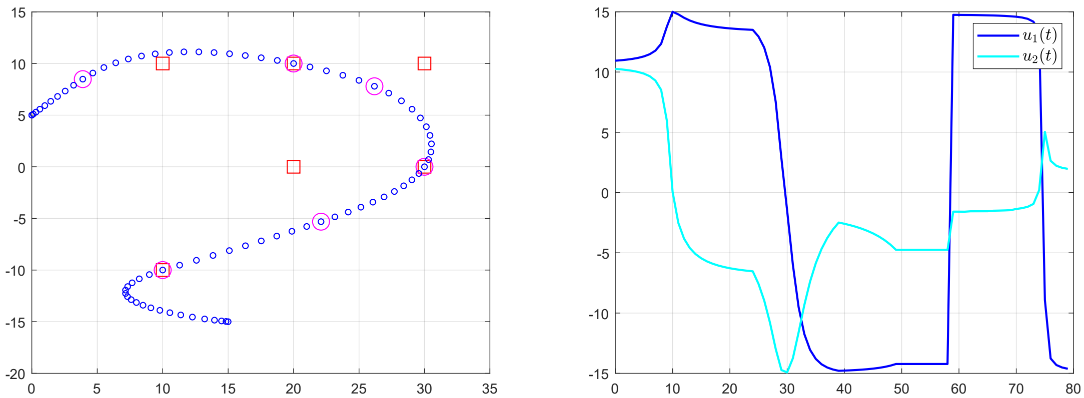


(a) Trajeto do robô



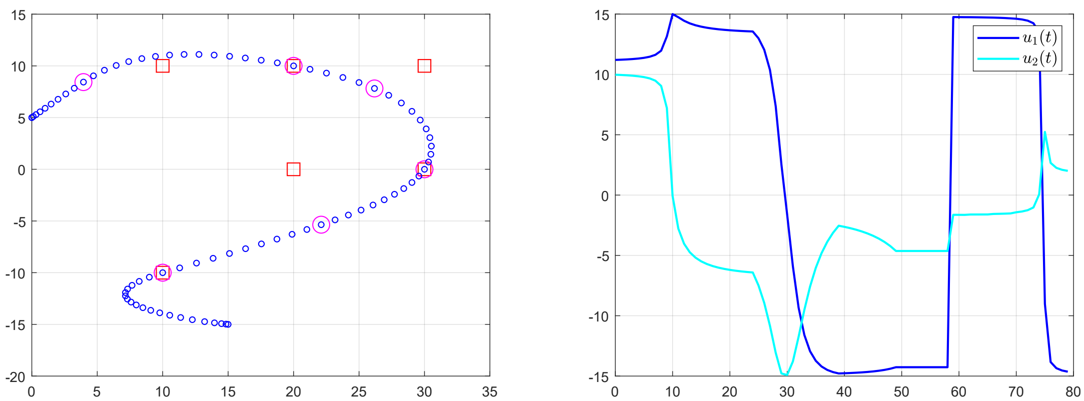
(b) Sinal de controlo

Figura 30: Resultados para $m = 2$.



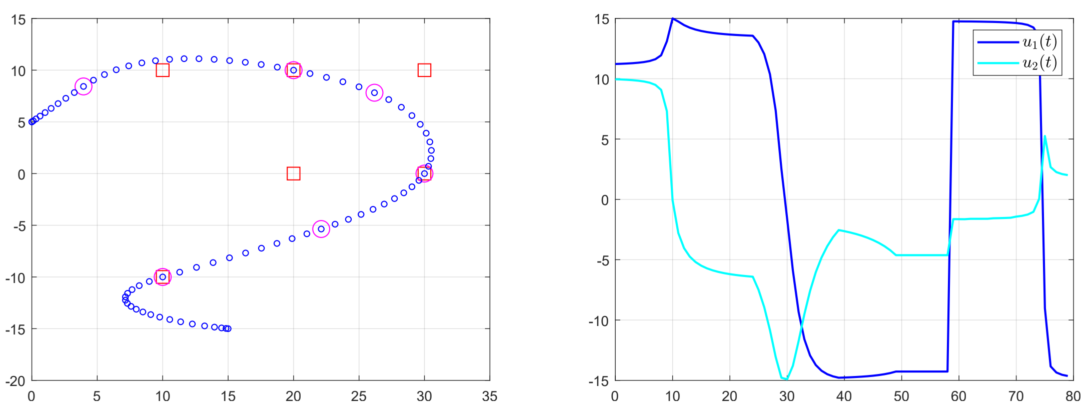
(a) Trajeto do robô (b) Sinal de controlo

Figura 31: Resultados para $m = 3$.



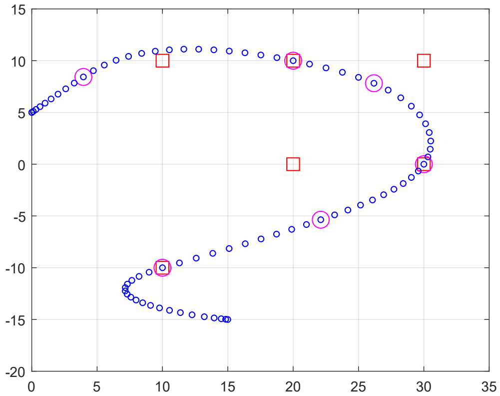
(a) Trajeto do robô (b) Sinal de controlo

Figura 32: Resultados para $m = 4$.

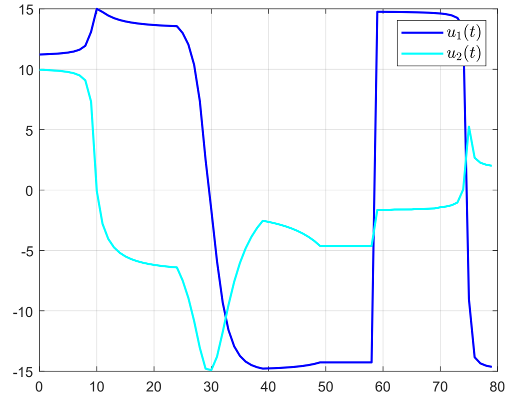


(a) Trajeto do robô (b) Sinal de controlo

Figura 33: Resultados para $m = 5$.

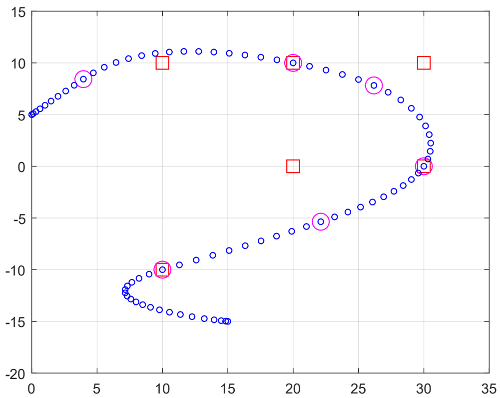


(a) Trajeto do robô

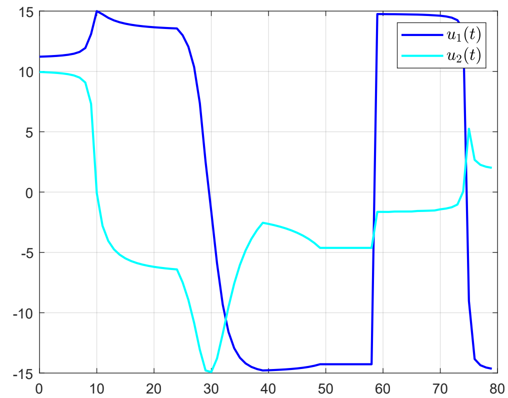


(b) Sinal de controlo

Figura 34: Resultados para $m = 6$.

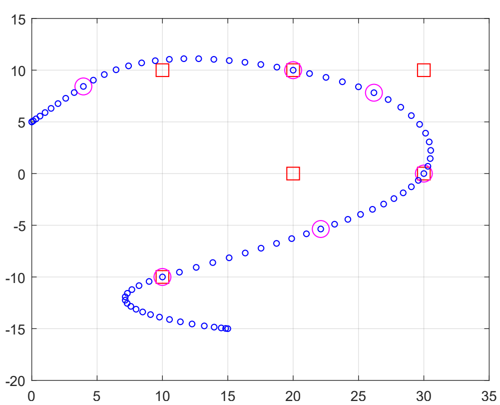


(a) Trajeto do robô

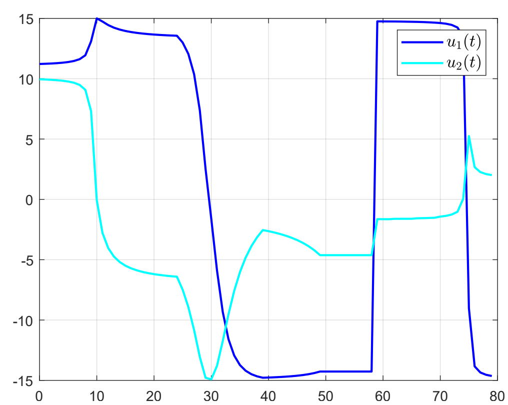


(b) Sinal de controlo

Figura 35: Resultados para $m = 7$.

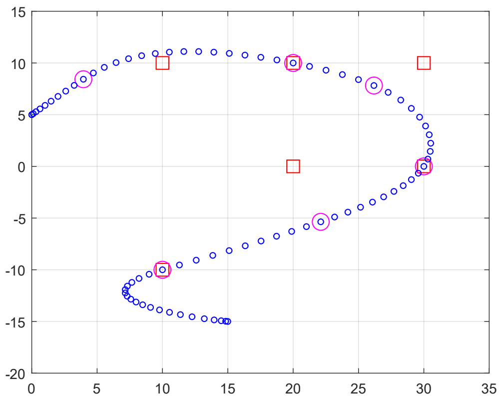


(a) Trajeto do robô

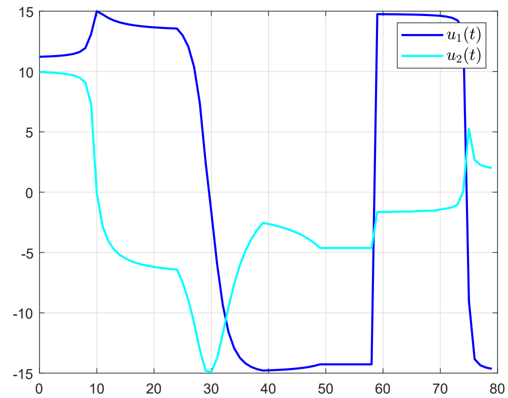


(b) Sinal de controlo

Figura 36: Resultados para $m = 8$.



(a) Trajeto do robô



(b) Sinal de controlo

Figura 37: Resultados para $m = 9$.

m	Capturas
0	1
1	2
2	2
3	3
4	3
5	3
6	3
7	3
8	3
9	3

Tabela 5: Número de pontos de rota capturados.

Código 8: `part1task11.m`

```
%[Part 1 - Task11]
%script that solves an optimization problem
%using the interactive reweighting technique

clear;
load x0.mat;

A = [1 0 0.1 0; 0 1 0 0.1;0 0 0.9 0; 0 0 0 0.9];
B = [0 0;0 0;0.1 0;0 0.1];
E = [1 0 0 0; 0 1 0 0];
wk = [10 20 30 30 20 10; 10 10 10 0 0 -10];
time_wk = [11 26 31 41 51 61];  %add one because of matlab indexing
e = 10^(-6);

%as the robot is stopped, velocity=0
x_initial = [0;5;0;0];
x_final = [15;-15;0;0];

Umax = 15;
T = 81;
K = 6;
x_ant = x0;

for m = 1:9
    cvx_begin quiet
        variables u(2,T-1) x(4,T) %state and control signal are the unknowns
        t = 1:T-1;
        cost = 0;
        %cost function
        for i = 1:K
           cost = cost + 1/(norm(E*x_ant(:,time_wk(i))-wk(:,i), 2) + e) * norm(E*x(:,time_wk(i))-wk(:,i), 2);
        end
        minimize(cost);
        %constraints
        subject to
            x(:,1) == x_initial
            x(:,T) == x_final
            for i = 1:T-1
                norm(u(:,i),2) <= Umax
            end
            x(:,t+1) == A*x(:,t) + B*u(:,t)
    cvx_end

    %plot robot positions
    figure;
    plot(wk(1,1:K),wk(2,1:K),'s','MarkerEdgeColor','red', 'MarkerSize',12);
    hold on;
    plot(x(1,time_wk(1:K)),x(2,time_wk(1:K)),'o','MarkerEdgeColor','magenta
```

```
        ', 'MarkerSize',12);
    plot(x(1,:),x(2,:),'o', 'MarkerEdgeColor','blue', 'MarkerSize',4);
    grid on;
    axis([0 35 -20 15]);

    %plot control signal
    figure;
    plot(t-1, u(1,:), 'blue', 'LineWidth', 1.5);
    hold on;
    plot(t-1, u(2,:), 'cyan', 'LineWidth', 1.5);
    grid on;
    leg = legend('$u_1(t)$', '$u_2(t)$');
    set(leg, 'Interpreter','latex', 'FontSize', 12);

    %captured waypoint
    captured(m) = 0;
    for i=1:K
        if(norm(E*x(:,time_wk(i)) - wk(:,i)) ≤ 10^(-6))
            captured(m) = captured(m) + 1;
        end
    end

    x_ant = x;
end
```

12 Task 12

Ao introduzir os pesos, consegue-se obter uma solução em que três *waypoints* são capturados. Analisando a expressão, percebe-se o seu propósito: dar, na iteração seguinte, mais importância aos pontos do trajeto mais próximos dos *waypoints* e menos aos restantes, por ser menos provável que a rota consiga fazer ajustes maiores. Esta técnica é possível por não interessar o quão perto se está de um ponto mas sim passar por ele, não fazendo sentido tentar um ajuste a pontos distantes quando isso impede passar exatamente por cima de outros.

Parte II

1 Task 1

Pretende-se provar a convexidade da função $f : \mathbf{R}^n \to \mathbf{R}$, descrita em (1).

$$f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\left(\log\left(1+\exp(s^T x_k - r)\right) - y_k\left(s^T x_k - r\right)\right). \tag{1}$$

Para tal, segue-se a decomposição da figura 38.

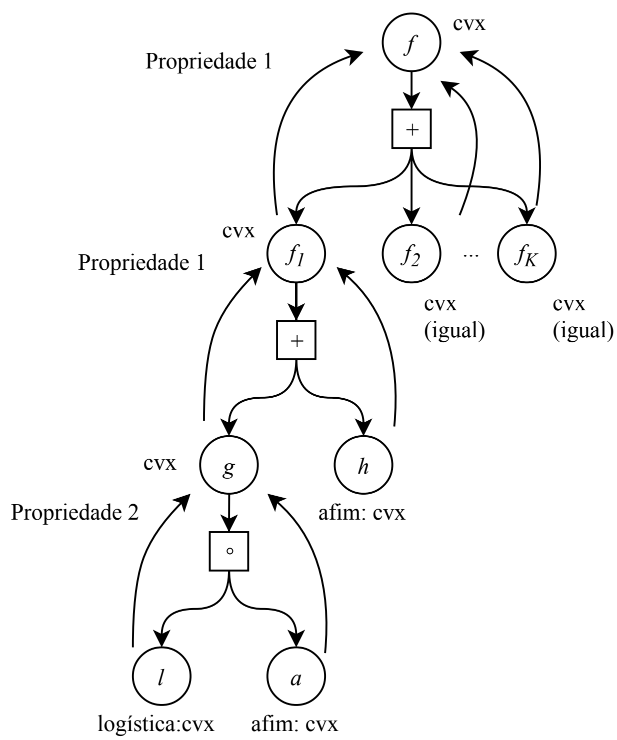


Figura 38: Prova de convexidade.

Primeiro, note-se que f é a soma (à parte de uma constante de multiplicação, que não afeta a convexidade), de K funções semelhantes, f_k, descritas em (2).

$$f_k(s,r) = \log\left(1+\exp(s^T x_k - r)\right) - y_k\left(s^T x_k - r\right). \tag{2}$$

Esta função pode, por sua vez, ser decomposta na soma de $g : \mathbf{R}^n \to \mathbf{R}$ e $h : \mathbf{R}^n \to \mathbf{R}$, tendo-se (3) e (4).

$$g(s,r) = \log\left(1+\exp(s^T x - r)\right). \tag{3}$$

$$h(s,r) = y\left(r - s^T x\right). \tag{4}$$

Sendo y igual a 0 ou 1, tem-se imediatamente que h ou é uma função afim ou é nula e, por isso, é convexa. Por outro lado, g pode ser vista como a composição das funções $a : \mathbf{R}^{n+1} \to \mathbf{R}$ e $l : \mathbf{R} \to \mathbf{R}$, descritas em (5) e (6) , tendo-se $g(s, r) = (l \circ a)(s, r) = l(a(s, r))$.

$$a(s, r) = s^T x - r. \tag{5}$$

$$l(z) = \log(1 + \exp(z)). \tag{6}$$

Ora, a é uma simples função afim, cuja convexidade está provada. Por outro lado, l trata-se de uma função logística, sendo por isso também uma função convexa.

Pode-se, então, retroceder no diagrama, das folhas para a raiz, por forma a comprovar a convexidade da função inicial. Primeiro, note-se que a composição $(l \circ a)(s, r)$ se trata de um mapeamento afim, esquematizado na figura 39, propriedade que preserva a convexidade. Assim, fica provada a convexidade de g.

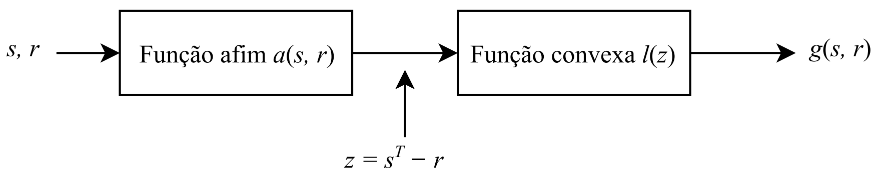


Figura 39: Propriedade 2.

De seguida, tem-se mais propriedade que preserva a convexidade: a soma de funções convexas com coeficientes de multiplicação positivos, descrita na figura 40. Como g e h satisfazem esta condição, prova-se facilmente que f_1 é convexa e, pelo mesmo raciocínio, qualquer f_k o será.

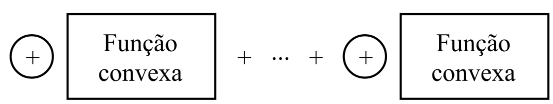


Figura 40: Propriedade 1.

Finalmente, tem-se de novo a propriedade 1, por se ter $f(s, r) = \sum_{k=1}^{K} f_k(s, r)$. Prova-se, então, que f é convexa.

2 Task 2

Na realização dos programas de MATLAB, usou-se o cálculo do gradiente demonstrado na *Task 5 – Parte 2.*

Primeiro que tudo, simplifica-se $s^T x_k - r$ (com $x_k, s \in \mathbf{R}^n$) a:

$$s^T x_k - r = s_1 x_1 + s_2 x_2 + ... + s_n x_n - r = \begin{bmatrix} x_{k1} & x_{k2} & \dots & x_{kn} & -1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \\ r \end{bmatrix}$$

.

Daqui retira-se que:

$$a_k = \begin{bmatrix} x_{k1} \\ x_{k2} \\ \vdots \\ x_{kn} \\ -1 \end{bmatrix} \qquad z = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \\ r \end{bmatrix}$$

. Posto isto, a função a minimizar (em $z \in \mathbf{R}^{n+1}$ pode então ser escrita na forma:

$$f(z) = \frac{1}{K}\sum_{k=1}^{K}(log(1 + exp(a_k^T z)) - y_k a_k^T z)$$

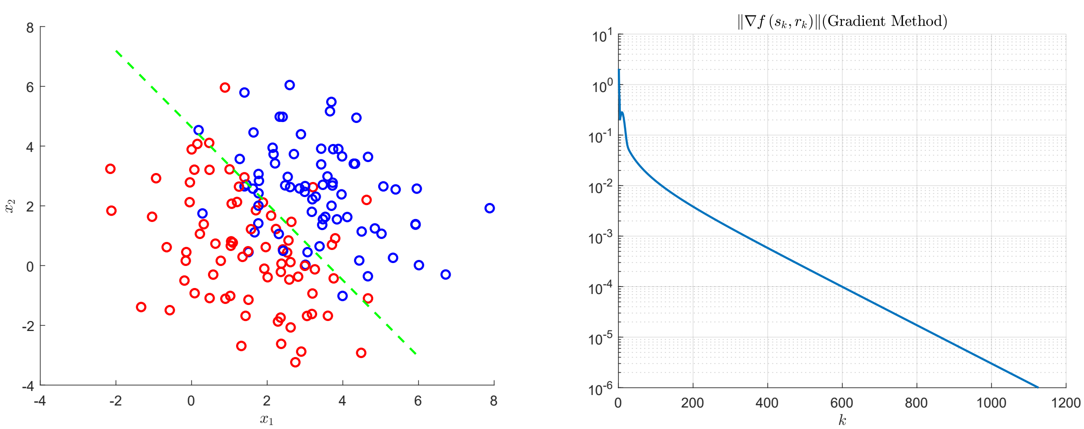
$$f(z) = \frac{1}{K}\sum_{k=1}^{K}(\phi(a_k^T z)),$$

com$\phi(t) = log(1 + exp(t)) - y_k t$e t, $y_k \in \mathbf{R}$.

Posteriormente, obtém-se

$$\dot{\phi}(t) = \frac{exp(t)}{1 + exp(t)} + y_k.$$

Utilizando o método do Gradiente, obtiveram-se os gráficos da figura 41 Os resultados para o conjunto de dados `data1.mat` foram $s = (1.3495, 1.0540)$ e $r = 4.8815$.



(a) Conjunto de dados e reta obtida

(b) Norma do gradiente

Figura 41: Resultados para o método do Gradiente, aplicado ao conjunto de dados `data1.mat`.

Está expresso em `part2task2.m` e `getf_x.m`, o código utilizado para obter os resultados da *Task 2*.

Código 9: `part2task2.m`

```
%[Part2 - task2, task3, task4]
%script that solves an optimization problem
%using the gradient method.

% differences between tasks 2,3 and 4
%reside solely in the dataset loaded

tic
K=length(X(1,:));
r=0;
dim=length(X(:,1));
epsilon=10^(-6);
alpha=1;
gama=10^(-4);
beta= 0.5;
s=-ones(1,dim);
x=[s' ;r];
ak= [X;-ones(1,K)];

alphak=alpha;

norms=[];
alphas=[];

 while(1)

    gk=(ak*((exp(ak'*x)./(1+ exp(ak'*x)))-Y'))/K;
     norms=[norms norm(gk)];
    if(norm(gk)<epsilon)
        break;
    end

    dk=-gk;

        alphak=alpha;
        fx=getf_x(x,K,ak,Y);
     while(1)
         fx_akdk=getf_x(x+(alphak*dk),K,ak,Y);
        if(fx_akdk<(fx+(gama*gk'*(alphak*dk))))
            break;
        end
        alphak=alphak*beta;
     end

     x=x+alphak*dk;
    alphas=[alphas alphak];


 end
 toc
 %% Requested s and r values
```

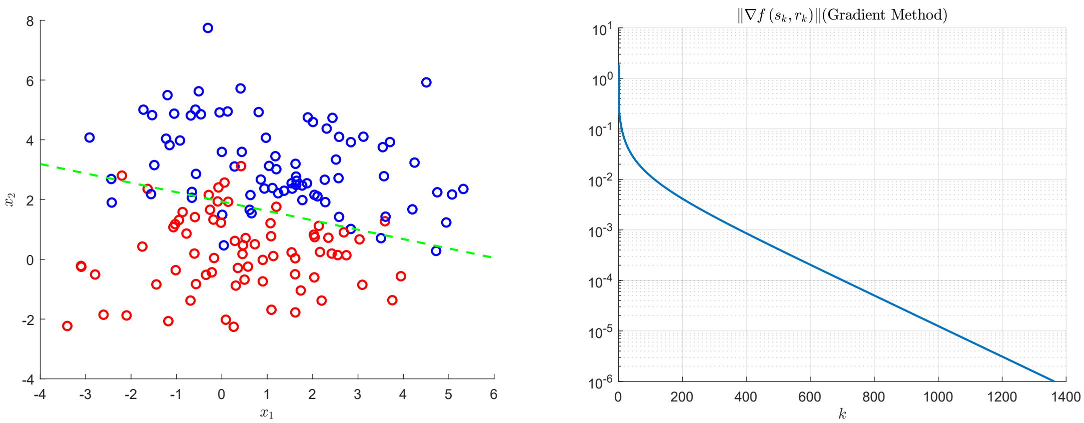
```
  s=x(1:(length(x)-1),1)
 r=x(length(x),1)

 figure()
 grid on
 hold on
 title('(Gradient Method)','Interpreter','Latex')
 aux=1:length(norms);
 plot(aux,norms);
 xlabel('k','Interpreter','Latex');
set(gca,'yscale','log');

figure()
for j=1:K
    if Y(j)==1
        scatter(X(1,j),X(2,j),[],'b');
    end
    if Y(j)==0
         scatter(X(1,j),X(2,j),[],'r');
    end
    hold on

end

%linear regression
x1_reg= -2:6;
for i=1:length(x1_reg)
    x2_reg(i)=(r-s(1)*x1_reg(i))/s(2);
end

plot(x1_reg,x2_reg, '--g');
```

Código 10: `getf_x.m`

```
%function that gets the output of f(x)

function [somat] = getf_x(x,K,ak,Y)
somat=sum(log(1+ exp(ak'*x)));
somat=(somat-Y*(ak'*x))/K;
end
```

3 Task 3

Utilizando o método do Gradiente, obtiveram-se os gráficos da figura 42. Os resultados para o conjunto de dados `data2.mat` foram $s = (0.7402, 2.3577)$ e $r = 4.5553$.



(a) Conjunto de dados e reta obtida (b) Norma do gradiente

Figura 42: Resultados para o método do Gradiente, aplicado ao conjunto de dados `data2.mat`.

Para a *Task 3*, pode também observar-se o código utilizado em `part2task2.m` e `getf_-x.m` mas, desta vez, para obter os resultados pretendidos carregou-se o *workspace* com um *dataset* diferente.

4 Task 4

Utilizando o método do Gradiente, obtiveram-se os gráficos das figuras 43 e 44, para os conjuntos de dados `data3.mat` e `data4.mat`, respetivamente. Como estes *datasets* não são de duas dimensões, representa-se apenas a norma do gradiente.

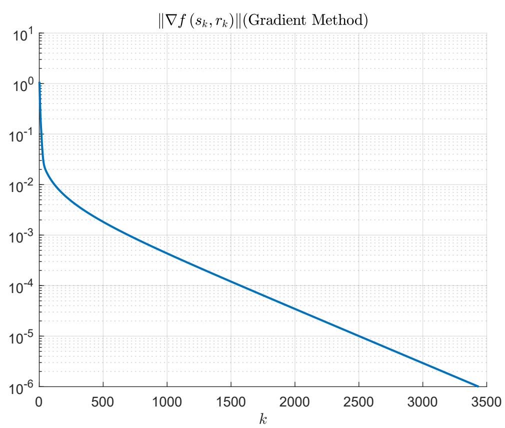


Figura 43: Resultados para o método do Gradiente, aplicado ao conjunto de dados `data3.mat`.

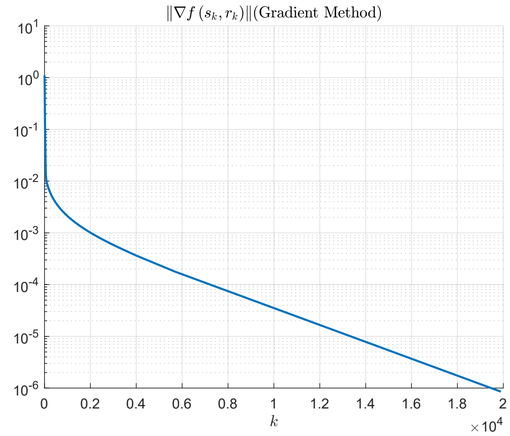


Figura 44: Resultados para o método do Gradiente, aplicado ao conjunto de dados `data4.mat`.

À semelhança da *Task 3*, para a *Task 4*, pode também observar-se o código utilizado em `part2task2.m` e `getf_x.m`, onde, novamente, para obter os resultados pretendidos, se carregou o *workspace* com dois *datasets* diferentes.

5 Task 5

Tem-se que $p : \mathbf{R}^3 \to \mathbf{R}$ é dada por:

$$p(x) = \sum_{k=1}^{K} \phi(a_k^T x) = \phi(a_1^T x) + \phi(a_2^T x) + ... + \phi(a_K^T x)$$

em que $\phi : \mathbf{R} \to \mathbf{R}$, $a_k \in \mathbf{R}^3$ para $k = 1, 2, .., K$ e $x \in \mathbf{R}^3$ para que $a_k x \in \mathbf{R}$ (ou seja $a_k = \begin{bmatrix} a_{k_1} & a_{k_2} & a_{k_3} \end{bmatrix}^T$ e $x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$).

(a)

Definindo $g(x_k) = a_k^T x = a_{k_1} x_1 + a_{k_2} x_2 + a_{K_3} x_3$, com para $k = 1, 2, .., K$ e $g : \mathbf{R}^3 \to \mathbf{R}$ tem-se que:

$$\nabla g_k(x) = \begin{bmatrix} \frac{\partial g_k}{\partial x_1}(x) \\ \frac{\partial g_k}{\partial x_2}(x) \\ \frac{\partial g_k}{\partial x_3}(x) \end{bmatrix} = \begin{bmatrix} a_{k_1} \\ a_{k_2} \\ a_{k_3} \end{bmatrix} = a_k.$$

Define-se também o gradiente de p como:

$$\nabla p(x_1, x_2, x_3) = \begin{bmatrix} \frac{\partial p}{\partial x_1}(x_1, x_2, x_3) \\ \frac{\partial p}{\partial x_2}(x_1, x_2, x_3) \\ \frac{\partial p}{\partial x_3}(x_1, x_2, x_3) \end{bmatrix}.$$

Através da regra da cadeia aplicada a cada parcela $\phi(g(x_k))$ obtém-se as derivadas parciais de p, dadas por:

$$\frac{\partial p}{\partial x_1}(x_1, x_2, x_3) = \sum_{k=1}^{K} \frac{\partial g_k}{\partial x_1}\dot{\phi}(g(x_k)) = a_{1_1}\dot{\phi}(g(x_1)) + a_{2_1}\dot{\phi}(g(x_2)) + ... + a_{K_1}\dot{\phi}(g(x_K))$$

$$\frac{\partial p}{\partial x_2}(x_1, x_2, x_3) = \sum_{k=1}^{K} \frac{\partial g_k}{\partial x_2}\dot{\phi}(g(x_k)) = a_{1_2}\dot{\phi}(g(x_1)) + a_{2_2}\dot{\phi}(g(x_2)) + ... + a_{K_2}\dot{\phi}(g(x_K))$$

$$\frac{\partial p}{\partial x_3}(x_1, x_2, x_3) = \sum_{k=1}^{K} \frac{\partial g_k}{\partial x_3}\dot{\phi}(g(x_k)) = a_{1_3}\dot{\phi}(g(x_1)) + a_{2_3}\dot{\phi}(g(x_2)) + ... + a_{K_3}\dot{\phi}(g(x_K))$$

Pode-se então escrever o gradiente de p na forma

$$\nabla p(x_1, x_2, x_3) = \begin{bmatrix} a_{1_1} \\ a_{1_2} \\ a_{1_3} \end{bmatrix} \dot{\phi}(g(x)) + \begin{bmatrix} a_{2_1} \\ a_{2_2} \\ a_{2_3} \end{bmatrix} \dot{\phi}(g(x)) + ... + \begin{bmatrix} a_{K_1} \\ a_{K_2} \\ a_{K_3} \end{bmatrix} \dot{\phi}(g(x))$$

que se simplifica a

$$\nabla p(x) = a_1\dot{\phi}(a_1^T x) + a_2\dot{\phi}(a_2^T x) + ... + a_k\dot{\phi}(a_K^T x).$$

Conclui-se assim que $\nabla p(x) = Av$, com $A = \begin{bmatrix} a_1 & a_2 & ... & a_K \end{bmatrix}$ e $v = \begin{bmatrix} \dot{\phi}(a_1^T x) \\ \dot{\phi}(a_2^T x) \\ \vdots \\ \dot{\phi}(a_K^T x) \end{bmatrix}$.

(b)

A hessiana de p é

$$\nabla^2 p(x_1, x_2, x_3) = \begin{bmatrix} \frac{\partial^2 p}{\partial x_1^2}(x) & \frac{\partial^2 p}{\partial x_1 \partial x_2}(x) & \frac{\partial^2 p}{\partial x_1 \partial x_3}(x) \\ \frac{\partial^2 p}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 p}{\partial x_2^2}(x) & \frac{\partial^2 p}{\partial x_2 \partial x_3}(x) \\ \frac{\partial^2 p}{\partial x_3 \partial x_1}(x) & \frac{\partial^2 p}{\partial x_3 \partial x_2}(x) & \frac{\partial^2 p}{\partial x_3^2}(x) \end{bmatrix},$$

As derivadas parciais de p, para o caso da primeira linha da hessiana, são dadas por

$$\frac{\partial^2 p}{\partial x_1^2}(x_1, x_2, x_3) = \sum_{k=1}^{K} \left(\frac{\partial g_k}{\partial x_1}\right)^2 \ddot{\phi}(g(x_k)) = a_{1_1}^2\ddot{\phi}(g(x_1)) + a_{2_1}^2\ddot{\phi}(g(x_2)) + ... + a_{K_1}^2\ddot{\phi}(g(x_K))$$

$$\frac{\partial^2 p}{\partial x_1 \partial x_2}(x_1, x_2, x_3) = \sum_{k=1}^{K} \frac{\partial g_k}{\partial x_1}\frac{\partial g_k}{\partial x_2}\ddot{\phi}(g(x_k)) = a_{1_1}a_{1_2}\ddot{\phi}(g(x_1)) + a_{2_1}a_{2_2}\ddot{\phi}(g(x_2)) + ... + a_{K_1}a_{K_2}\ddot{\phi}(g(x_K))$$

$$\frac{\partial^2 p}{\partial x_1 \partial x_3}(x_1, x_2, x_3) = \sum_{k=1}^{K} \frac{\partial g_k}{\partial x_1}\frac{\partial g_k}{\partial x_3}\ddot{\phi}(g(x_k)) = a_{1_1}a_{1_3}\ddot{\phi}(g(x_1)) + a_{2_1}a_{2_3}\ddot{\phi}(g(x_2)) + ... + a_{K_1}a_{K_3}\ddot{\phi}(g(x_K))$$

Calculando as restantes entradas da matriz da mesma forma, e isolando cada termo $\ddot{\phi}(g(x_k))$ por manipulação matemática obtém-se

$$\nabla^2 p(x_1, x_2, x_3) = \ddot{\phi}(g(x_1)) \begin{bmatrix} a_{1_1}^2 & a_{1_1}a_{1_2} & a_{1_1}a_{1_3} \\ a_{1_1}a_{1_2} & a_{1_2}^2 & a_{1_2}a_{1_3} \\ a_{1_1}a_{1_3} & a_{1_2}a_{1_3} & a_{1_3}^2 \end{bmatrix} + \ddot{\phi}(g(x_2)) \begin{bmatrix} a_{2_1}^2 & a_{2_1}a_{2_2} & a_{2_1}a_{2_3} \\ a_{2_1}a_{2_2} & a_{2_2}^2 & a_{2_2}a_{2_3} \\ a_{2_1}a_{2_3} & a_{2_2}a_{2_3} & a_{2_3}^2 \end{bmatrix} + \\ ... + \ddot{\phi}(g(x_K)) \begin{bmatrix} a_{K_1}^2 & a_{K_1}a_{K_2} & a_{K_1}a_{K_3} \\ a_{K_1}a_{K_2} & a_{K_2}^2 & a_{K_2}a_{K_3} \\ a_{K_1}a_{K_3} & a_{K_2}a_{K_3} & a_{K_3}^2 \end{bmatrix}$$

que se simplifica para

$$\nabla^2 p(x_1, x_2, x_3) = \ddot{\phi}(g(x_1)) \begin{bmatrix} a_{1_1} & a_{1_2} & a_{1_3} \end{bmatrix}^T \begin{bmatrix} a_{1_1} & a_{1_2} & a_{1_3} \end{bmatrix} + \\ + \ddot{\phi}(g(x_2)) \begin{bmatrix} a_{2_1} & a_{2_2} & a_{2_3} \end{bmatrix}^T \begin{bmatrix} a_{2_1} & a_{2_2} & a_{2_3} \end{bmatrix} + \\ ... + \ddot{\phi}(g(x_K)) \begin{bmatrix} a_{K_1} & a_{K_2} & a_{K_3} \end{bmatrix}^T \begin{bmatrix} a_{K_1} & a_{K_2} & a_{K_3} \end{bmatrix}$$

$$\nabla^2 p(x_1, x_2, x_3) = \ddot{\phi}(g(x_1))a_1 a_1^T + \ddot{\phi}(g(x_2))a_2 a_2^T + ... + \ddot{\phi}(g(x_K))a_K a_K^T$$

Em termos matriciais, pode-se escrever como

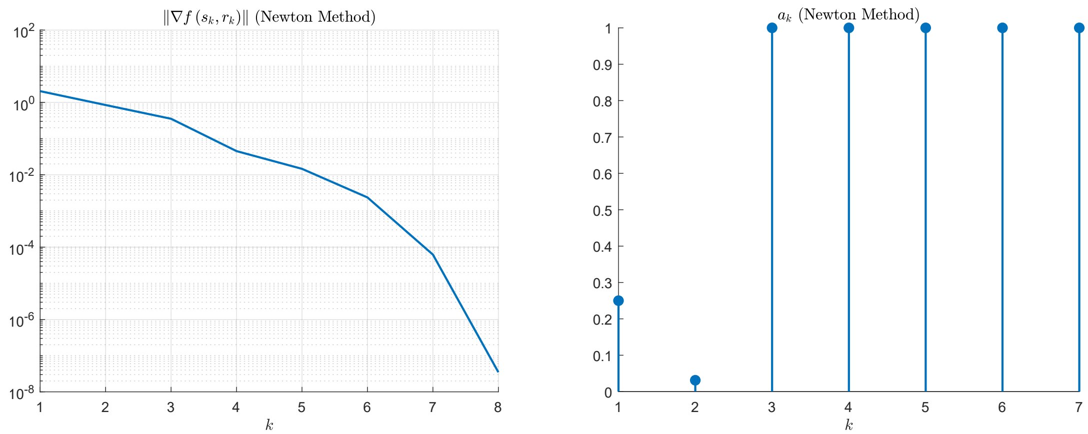
$$\nabla^2 p(x_1, x_2, x_3) = \begin{bmatrix} a_1 & a_2 & \dots & a_K \end{bmatrix} \begin{bmatrix} \ddot{\phi}(g(x_1)) & & & \\ & \ddot{\phi}(g(x_2)) & & \\ & & \ddots & \\ & & & \ddot{\phi}(g(x_K)) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix}$$

Conclui-se, assim, que $\nabla^2 p(x) = ADA^T$, com $A = \begin{bmatrix} a_1 & a_1 & ... & a_K \end{bmatrix} \in \mathbf{R}^{n,k}$ e D a matriz diagonal:

$$D = \begin{bmatrix} \ddot{\phi}(a_1^T x) & & & \\ & \ddot{\phi}(a_2^T x) & & \\ & & \ddots & \\ & & & \ddot{\phi}(a_K^T x) \end{bmatrix}.$$

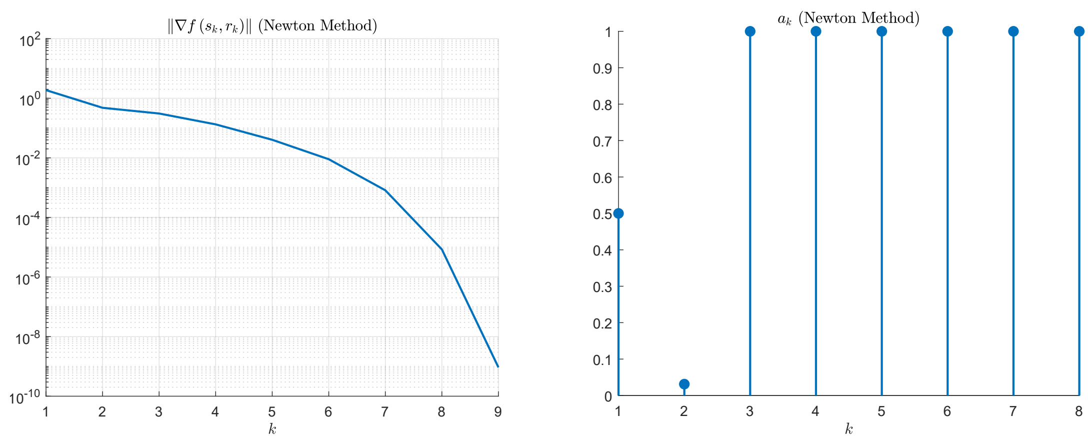
6 Task 6

Analogamente à *Task 2 – Parte 2*, reescreveu-se o problema no formato da *Task 5*. Além do que foi definido na *Task 2*, calculou-se também $\ddot{\phi}(t) = \frac{\exp(t)}{(1+\exp(t))^2}$, $t \in \mathbf{R}$, que servirá para o cálculo da matriz diagonal D.



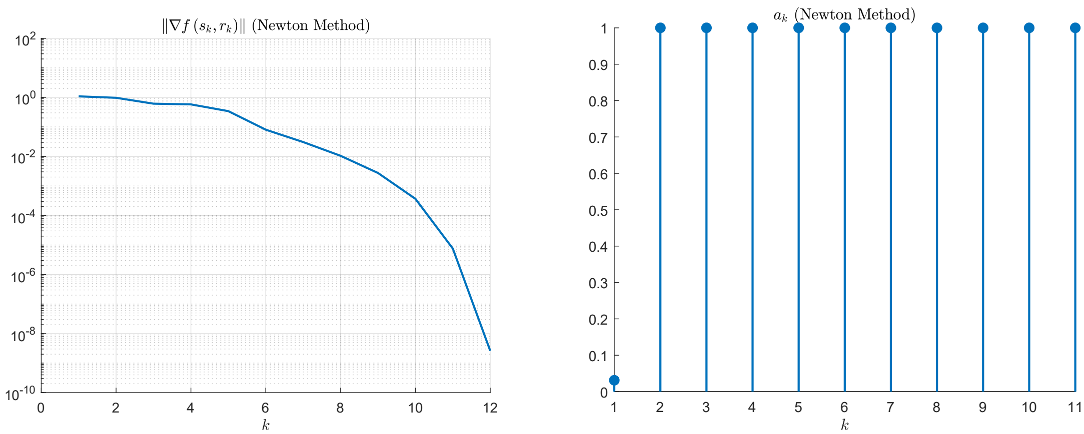
(a) Norma do gradiente (b) Valores dos *stepsizes*

Figura 45: Resultados ao longo das iterações para o método de Newton, aplicado ao conjunto de dados `data1.mat`.



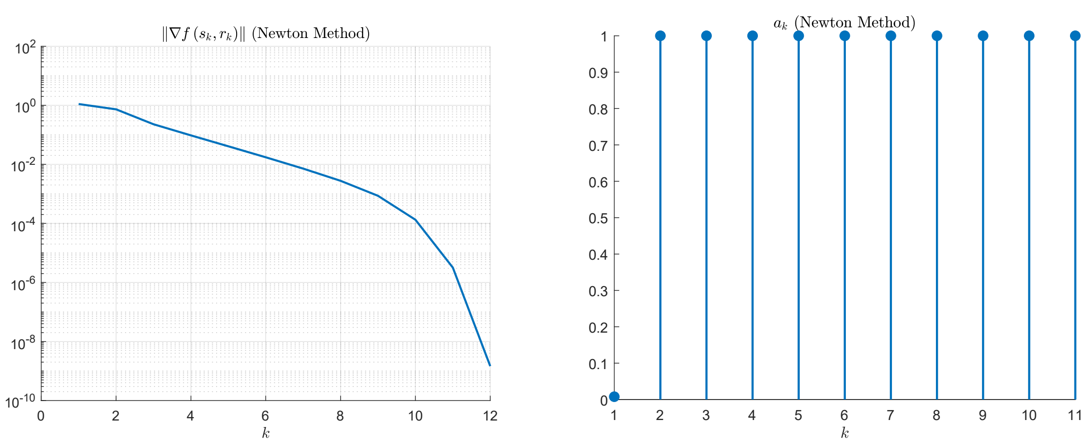
(a) Norma do gradiente (b) Valores dos *stepsizes*

Figura 46: Resultados ao longo das iterações para o método de Newton, aplicado ao conjunto de dados `data2.mat`.



(a) Norma do gradiente (b) Valores dos *stepsizes*

Figura 47: Resultados ao longo das iterações para o método de Newton, aplicado ao conjunto de dados `data3.mat`.



(a) Norma do gradiente (b) Valores dos *stepsizes*

Figura 48: Resultados ao longo das iterações para o método de Newton, aplicado ao conjunto de dados `data4.mat`.

Está expresso em `part2task6.m` e `getf_x.m` (o segundo já utilizado nas *Tasks 2, 3 e 4*), o código utilizado para obter os resultados da *Task 6*.

Código 11: `part2task6.m`

```
%[Part 2 - task6]
%script that solves an optimization problem
%using the Newton method.

%this script is meant to run for 4 different datasets
%it is assumed that the intended one is (the only) loaded

tic
K=length(X(1,:));
r=0;
dim=length(X(:,1));
epsilon=10^(-6);
alpha=1;
gama=10^(-4);
beta= 0.5;
s=-ones(1,dim);
x=[s' ;r];
ak= [X;-ones(1,K)];

alphak=alpha;

norms=[];
alphas=[];

 while(1)

    gk=(ak*((exp(ak'*x)./(1+ exp(ak'*x)))-Y'))/K;
     norms=[norms norm(gk)];
    if(norm(gk)<epsilon)
        break;
    end

    hessian= (ak*diag(exp(ak'*x)./(1+ exp(ak'*x)).^2)*ak')/K;
    dk=-(hessian)\gk;

        alphak=alpha;
        fx=getf_x(x,K,ak,Y);
     while(1)
         fx_akdk=getf_x(x+(alphak*dk),K,ak,Y);
        if(fx_akdk<(fx+(gama*gk'*(alphak*dk))))
            break;
        end
        alphak=alphak*beta;
     end

```

```
     x=x+alphak*dk;
    alphas=[alphas alphak];


 end
 toc

 %%
 figure()
 grid on
 hold on
 title('(Newton Method)','Interpreter','Latex')
 aux=1:length(norms);
 plot(aux,norms);
 xlabel('k','Interpreter','Latex');
set(gca,'yscale','log');

figure()
 grid on
 hold on
 title('(Newton Method)','Interpreter','Latex')
 aux=1:length(alphas);
 xlabel('k','Interpreter','Latex');
stem(aux,alphas);
```

7 Task 7

Sabe-se que o método de Newton apresenta menos iterações que o método do Gradiente até à convergência da função. No entanto, existe um preço a pagar: cada iteração é mais demorada. No método do Gradiente, a *descent direction* é $d_k = -\nabla f(x_k)$, ao passo que para o de Newton $d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$. Assim, por cada iteração, o método de Newton apresenta uma complexidade temporal média, associada ao cálculo do gradiente bem como o cálculo e posterior inversão da matriz hessiana, de $\mathcal{O}(N^3)$. Já o método de Gradiente, por cada iteração apresenta uma complexidade temporal média de $\mathcal{O}(N)$, para o cálculo do gradiente.

A tabela 6 mostra os resultados obtidos para todos os *datasets* para os métodos do Gradiente e de Newton.

Como se pode observar pela tabela 6, os resultados mostram que o método de Newton foi sempre mais rápido a convergir que o método de Gradiente, apresentando também menos iterações até à convergência. O menor número de iterações deve-se ao facto de no método de Newton se ter um ritmo de convergência dado por $\|x_k - x_*\|_2 \approx \|x_* - x_{k-1}\|_2^2$ (por cada iteração aumenta-se o "passo" de forma quadrática), ao passo que no do Gradiente se tem $\|x_k - x_*\|_2 \approx \|x_* - x_{k-1}\|_2$.

O *dataset* de maior dimensão apresenta apenas uma dimensão de $n = 100$, pelo que a matriz Hessiana a calcular e inverter é de dimensão 100×100. Se esta apresentasse valores

	dataset	Gradiente	Newton
Tempo (s)	1	0.06	0.03
	2	0.07	0.02
	3	0.59	0.05
	4	57.74	4.25
Iterações	1	1125	7
	2	1362	8
	3	> 3000	11
	4	$> 2 \times 10^4$	11

Tabela 6: Comparação entre os métodos do Gradiente e Newton para os datasets apresentados.

consideravelmente superiores (por exemplo $n = 5000$) esperar-se-ia que o tempo de convergência do método de Newton fosse superior ao do método do gradiente, devido ao aumento do custo de cada iteração com N^3. Observa-se que o método de Newton com o aumento de iterações, aumenta o tempo médio por iteração. Comparando por exemplo o *dataset 3* com o *dataset 4* (de maior dimensão que o anterior), verifica-se que ambos apresentam o mesmo número de iterações, no entanto o *dataset 4* tem um tempo de convergência bastante superior - pois a matriz hessiana a calcular e inverter passa de $n = 30 \times 30$ para $n = 100 \times 100$ assim como o cálculo da função de $f(x)$ que passa de um somatório de $k = 500$ para $k = 8000$.

O tempo por iteração no método do Gradiente também aumenta devido ao aumento do tempo associado ao cálculo da função $f(x)$ assim como o cálculo do seu gradiente. Realça-se que este aumento não é tão acentuado quanto o observado para o outro método. O número de iterações aumenta muito com o aumento do tamanho do *dataset*.

Apesar de ambos os métodos terem a limitação de poder convergir para um mínimo relativo, quando o que se pretende é atingir um mínimo absoluto (como solução), pode-se afirmar que o método de Newton tem maior precisão.

É de se notar que existem casos em que vale mais a pena utilizar o método de Gradiente que o de Newton – por exemplo para *datasets* de elevada dimensão, pois apesar de o número de iterações que o de Newton usa ser menor, pode não compensar o tempo e a memória a mais que leva a fazer cada iteração. Esta escolha vai depender também do objetivo pretendido: se por exemplo se pretender, para um *dataset* de grande dimensão, obter a maior precisão possível, pode compensar o tempo a mais que o método de Newton vai levar para convergir. Existem também casos em que se opta por utilizar os dois métodos para a obtenção de um mínimo: executa-se primeiro o do Gradiente e depois o de Newton.

8 Task 8

Para se resolver este problema utilizando o método LM, teve de se calcular o gradiente das funções f_a e f_s, descritas em (7) e (8).

$$f_a(x) = \left(\|a_m - s_p\| - y_{mp}\right)^2. \tag{7}$$

$$f_s(x) = \left(\|s_p - s_q\| - z_{pq}\right)^2. \tag{8}$$

Como x é um vetor que contém ambas as coordenadas de todos os pontos, cria-se um conjunto de matrizes $E \in \mathbf{R}^{2\times 16}$ que permitem fazer esta passagem. Por exemplo, para s_1, ter-se-á

$$s_1 = E_1 x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} x.$$

Por outro lado, a subtração $s_1 - s_2$ fica resumida a

$$s_1 - s_2 = E_{12} x = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & \dots & 0 & 0 \end{bmatrix} x.$$

Assim, podem-se reescrever as funções f_a e f_s:

$$f_a(x) = \left(\|a_m - E_a x\| - y_{mp}\right)^2.$$

$$f_s(x) = \left(\|E_s x\| - z_{pq}\right)^2.$$

Como f_a é mais geral, demonstrar-se-á apenas o cálculo do seu gradiente, sendo $\nabla f_s(x)$ obtido de forma idêntica. Começa-se por definir as funções f_p e g, tal que $f_a(x) = (f_p \circ g)(x)$:

$$g(x) = E_a x$$

$$f_p(y) = \left(\|a_m - y\| - y_{mp}\right)^2.$$

F Agora, pela regra da cadeia, sabe-se que $\nabla f_a(x) = (Dg(x))^T \nabla f_p(g(x))$. Tem, então, de se calcular o gradiente de f_p e a jacobiana de g. Para o primeiro:

$$\nabla f_p(x) = 2\left(\|a_m - y\| - y_{mp}\right) \nabla\left(\|a_m - y\|\right) = -2\left(\|a_m - y\| - y_{mp}\right) \frac{a_m - y}{\|a_m - y\|}.$$

Para a segunda, pode-se desenvolver E_a e g como:

$$E_a = \begin{bmatrix} e_{1,1} & e_{1,2} & \dots & e_{1,16} \\ e_{2,1} & e_{2,2} & \dots & e_{2,16} \end{bmatrix}.$$

$$g(x) = \begin{bmatrix} g_1(x) \\ g_2(x) \end{bmatrix} = \begin{bmatrix} e_{1,1}x_1 & e_{1,2}x_2 & \dots & e_{1,16}x_{16} \\ e_{2,1}x_1 & e_{2,2}x_2 & \dots & e_{2,16}x_{16} \end{bmatrix}.$$

Ora, para calcular a jacobiana de g, basta derivar em todas as coordenadas:

$$Dg(x) = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x} \\ \frac{\partial g_2(x)}{\partial x} \end{bmatrix} = \begin{bmatrix} \frac{\partial g_1(x)}{\partial x_1} & \frac{\partial g_1(x)}{\partial x_2} & \dots & \frac{\partial g_1(x)}{\partial x_{16}} \\ \frac{\partial g_2(x)}{\partial x_1} & \frac{\partial g_2(x)}{\partial x_2} & \dots & \frac{\partial g_2(x)}{\partial x_{16}} \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{1,2} & \dots & e_{1,16} \\ e_{2,1} & e_{2,2} & \dots & e_{2,16} \end{bmatrix} = E_a.$$

Finalmente, podem-se fazer as substituições adequadas para ter a expressão final dos gradientes necessários, incluindo o da função original, f.

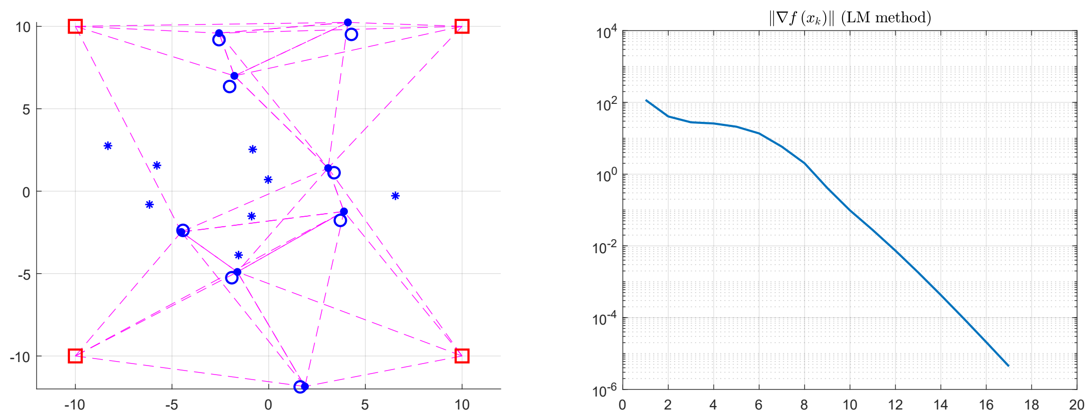
$$\nabla f_a(x) = (Dg(x))^T \, \nabla f_p\left(g(x)\right) = -2E_a^T\left(\|a_m - E_a x\| - y_{mp}\right)\frac{a_m - E_a x}{\|a_m - E_a x\|}$$

$$\nabla f_s(x) = 2E_s^T\left(\|E_s x\| - z_{pq}\right)\frac{E_s x}{\|E_s x\|}$$

$$\nabla f(x) = -\,2\sum_{(m,p)\in\mathcal{A}} E_a^T\left(\|a_m - E_{amp}x\| - y_{mp}\right)\frac{a_m - E_{amp}x}{\|a_m - E_{amp}x\|}$$

$$+\,2\sum_{(p,q)\in\mathcal{S}} E_{spq}^T\left(\|E_{spq}x\| - z_{pq}\right)\frac{E_{spq}x}{\|E_{spq}x\|}$$

Consegue-se, assim, criar o programa em MATLAB que permite reproduzir o enunciado – figura 49.



(a) Localização da rede (b) Norma do gradiente

Figura 49: Resultados do método LM para os dados do ficheiro `lmdata1`.

Está expresso em `part2task8.m`, `f.m`, `gradf.m`, `gradg.m`, `gradh.m` e `computeAb.m`, o código utilizado para obter os resultados da *Task 8*.

Código 12: `part2task8.m`

```
%[Part 2 — task8]
%script that solves an optimization problem
%using the Levenberg—Marquardt method

clear
load lmdata2
load xbest
xinit = xbest;

```

```
global iA
iA = iA;
global iS
iS = iS;
global A;
A = A;
global y;
y = y;
global z;
z = z;

% create Es
global Es
for i=1:size(iS(:,1),1)
    Es(:,:,i)=0;
    Es(2,2*iS(i,1),i) = 1;
    Es(1,2*iS(i,1)-1,i) = 1;
    Es(2,2*iS(i,2),i) = -1;
    Es(1,2*iS(i,2)-1,i) = -1;
end

% create Ea
global Ea
for i=1:size(iA(:,1),1)
    Ea(:,:,i)=0;
    Ea(2,2*iA(i,2),i) = 1;
    Ea(1,2*iA(i,2)-1,i) = 1;
end


lambda = 1;
e = 10^-6;
k = 1;
xk = xinit;
fk = f(xinit);
gk(k) = norm(gradf(xk));

while(gk(k) ≥ e)

    [Am, b] = computeAb(xk, lambda);
    xk1 = Am\b;

    fk1 = f(xk1);
    if(fk1 < fk)
        xk = xk1;
        lambda = 0.7*lambda;
        fk = fk1;
    else
        lambda = 2*lambda;
    end

```

```
    k = k + 1;

    gk(k) = norm(gradf(xk));
end

figure
hold on;
xlim([-12 12]);
ylim([-12 max(S(:))]);
for i=1:size(iA(:,1),1)
    plot([A(1,iA(i,1)) S(1,iA(i,2))], [A(2,iA(i,1)) S(2,iA(i,2))], 'magenta--');
end
for i=1:size(iS(:,1),1)
    plot([S(1,iS(i,1)) S(1,iS(i,2))], [S(2,iS(i,1)) S(2,iS(i,2))], 'magenta--');
end

plot(A(1,:),A(2,:),'s','MarkerEdgeColor','red', 'MarkerSize',12, 'LineWidth', 1.5)
hold on;
plot(S(1,:),S(2,:),'o','MarkerEdgeColor','blue', 'MarkerFaceColor','blue', 'MarkerSize',4, 'LineWidth', 1.5);
plot(xk(1:2:16)',xk(2:2:16)','o','MarkerEdgeColor','blue', 'MarkerSize',8, 'LineWidth', 1.5);
grid on;
plot(xinit(1:2:16),xinit(2:2:16),'*', 'MarkerEdgeColor','blue', 'MarkerFaceColor','blue', 'MarkerSize',6, 'LineWidth', 1);

figure;
semilogy(1:(k-1), gk(1:k-1), 'LineWidth', 1.5);
title('$\left\Vert\nabla f\left(x_k\right)\right\Vert$ (LM method)','Interpreter','Latex')
grid on;
xlim([0 20]);
ylim([10^-6 10^4]);
```

Código 13: `f.m`

```
%function that calculates f(x)

function sum = f(x)
    global Ea
    global Es
    global iA
    global iS
    global A
    global y
    global z

```

```
    M = size(iA(:,1),1);
    P = size(iS(:,1),1);
    Max = max(M, P);
    sum = 0;
    for i = 1:Max
        if(i≤M)
            sum = sum + (norm(A(:,iA(i,1))−Ea(:,:,i)*x)−y(i))^2;
        end
        if(i≤P)
            sum = sum + (norm(Es(:,:,i)*x)−z(i))^2;
        end
    end
end
```

Código 14: `gradf.m`

```
%function that calculates the
%gradient of the function f(x)

function grad = gradf(x)
    global Ea
    global Es
    global iA
    global iS
    global A
    global y
    global z

    M = size(iA(:,1),1);
    P = size(iS(:,1),1);
    Max = max(M, P);
    grad = zeros(1, 16);

    for i = 1:Max
        if(i≤M)
            grad = grad + gradg(i, x)*2*(norm(A(:,iA(i,1))−Ea(:,:,i)*x)−y(i
                ));
        end
        if(i≤P)
            grad = grad +gradh(i, x)*2*(norm(Es(:,:,i)*x)−z(i));
        end
    end
end
```

Código 15: `gradg.m`

```
%function that calculates the
%gradient of the function gi(x)

```

```
function grad = gradg(i, x)
    global Ea
    global A
    global iA
    E = Ea(:,:,i);
    grad = -E'*(A(:,iA(i,1))-E*x)/norm(A(:,iA(i,1))-E*x);
    grad = grad';
end
```

Código 16: `gradh.m`

```
%function that calculates the
%gradient of the function hi(x)

function grad = gradh(i, x)
    global Es
    E = Es(:,:,i);
    grad = E'*(E*x)/norm(E*x);
    grad = grad';
end
```

Código 17: `computeAb.m`

```
%function that gets de A matrix and b vector
%for the Levenberg-Marquardt method.

function [Am, b] = computeAb(x, lambda)
    global Ea
    global Es
    global iA
    global iS
    global A
    global y
    global z

    M = size(iA(:,1),1);
    P = size(iS(:,1),1);
    Max = max(M, P);
    I = eye(16);
    Am = zeros(56, 16);
    b = zeros(56, 1);

    for i = 1:Max
        if(i<=M)
            Am(i,:) = gradg(i, x);
            b(i) = gradg(i, x)*x - (norm(A(:,iA(i,1))-Ea(:,:,i)*x)-y(i));
        end
        if(i<=P)
            Am(i+M,:) = gradh(i, x);
```

```
            b(i+M) = gradh(i, x)*x - (norm(Es(:,:,i)*x)-z(i));
        end
    end
    Am(41:56,:) = sqrt(lambda)*I;
    b(41:56) = sqrt(lambda)*x;
end
```

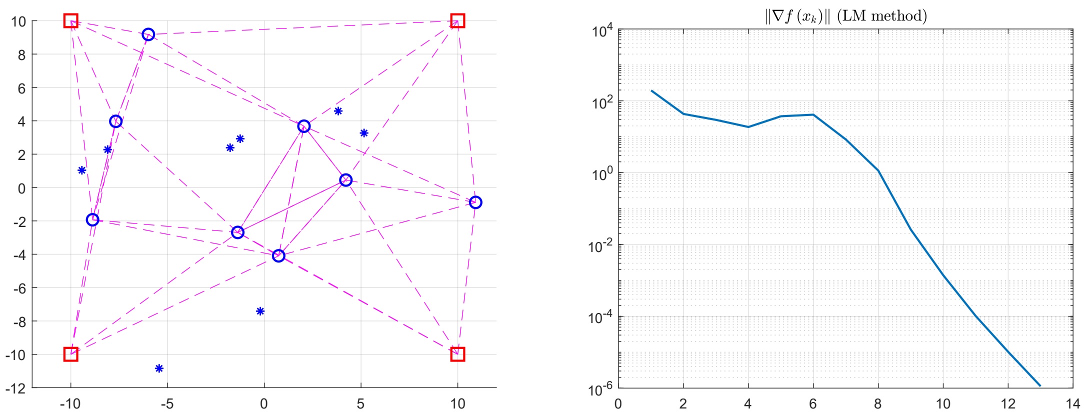
9 Task 9

Contendo o programa utilizado na questão anterior num ciclo por forma a corrê-lo N vezes, sempre com inicializações aleatórias, verificou-se que rapidamente se encontrava o custo mínimo de 4.4945, que, por mais que se aumentasse o número de iterações, não melhorava. Assim, considerou-se a melhor solução:

$$x = \begin{bmatrix} 5.16 \\ 3.27 \\ 3.82 \\ 4.59 \\ -0.21 \\ -7.41 \\ -8.10 \\ 2.27 \\ -1.25 \\ 2.93 \\ -9.43 \\ 1.03 \\ -1.77 \\ 2.39 \\ -5.43 \\ -10.84 \end{bmatrix} \tag{9}$$

Para esta situação, obtêm-se os gráficos da figura 50. Por não serem necessárias muitas tentativas para descobrir o mínimo da função de custo (a partir de 100 não se verificaram melhorias) e analisando a figura 50 (b), percebe-se que este exemplo não terá muitos mínimos locais onde a minimização possa ficar presa e convergirá para as posições reais independentemente da inicialização. De facto, verifica-se que as posições finais nada têm a ver com as iniciais.

Está expresso em `part2task9.m`, `part2task9plot.m`, `f.m`, `gradf.m`, `gradg.m`, `gradh.m` e `computeAb.m`, o código utilizado para obter os resultados da *Task 9*.



(a) Localização da rede

(b) Norma do gradiente

Figura 50: Resultados do método LM para os dados do ficheiro `lmdata2`.

Código 18: `part2task9.m`

```
%[Part 2 — task9]
%script that solves an optimization problem
%using the Levenberg—Marquardt method

clear
load lmdata2

global iA
iA = iA;
global iS
iS = iS;
global A;
A = A;
global y;
y = y;
global z;
z = z;

% create Es
global Es
for i=1:size(iS(:,1),1)
    Es(:,:,i)=0;
    Es(2,2*iS(i,1),i) = 1;
    Es(1,2*iS(i,1)—1,i) = 1;
    Es(2,2*iS(i,2),i) = —1;
    Es(1,2*iS(i,2)—1,i) = —1;
end

% create Ea
global Ea
for i=1:size(iA(:,1),1)
```

```
    Ea(:,:,i)=0;
    Ea(2,2*iA(i,2),i) = 1;
    Ea(1,2*iA(i,2)-1,i) = 1;
end

fbest = 1000;
e = 10^-6;

for i = 1:1000

    lambda = 1;
    k = 1;
    xinit = rand(16,1)*22-11;
    xk = xinit;
    fk = f(xinit);
    gk(k) = norm(gradf(xk));

    while(gk(k) ≥ e)

        [Am, b] = computeAb(xk, lambda);
        xk1 = Am\b;

        fk1 = f(xk1);
        if(fk1 < fk)
            xk = xk1;
            lambda = 0.7*lambda;
            fk = fk1;
        else
            lambda = 2*lambda;
        end

        k = k + 1;

        gk(k) = norm(gradf(xk));
    end

    if(fk < fbest)
        fbest = fk;
        xbest = xinit;
    end
end

fprintf("\ndone\n");
```

Código 19: `part2task9plot.m`

```
%script that plots the results from task9 (Part 2)

figure
hold on;
```

```
xlim([-12 12]);
S(1,:) = xk(1:2:16);
S(2,:) = xk(2:2:16);
ylim([-12 max(A(:))]);
for i=1:size(iA(:,1),1)
    plot([A(1,iA(i,1)) S(1,iA(i,2))], [A(2,iA(i,1)) S(2,iA(i,2))], 'magenta--');
end
for i=1:size(iS(:,1),1)
    plot([S(1,iS(i,1)) S(1,iS(i,2))], [S(2,iS(i,1)) S(2,iS(i,2))], 'magenta--');
end

plot(A(1,:),A(2,:),'s','MarkerEdgeColor','red', 'MarkerSize',12, 'LineWidth', 1.5)
hold on;
%plot(S(1,:),S(2,:),'o','MarkerEdgeColor','blue', 'MarkerFaceColor','blue', 'MarkerSize',4, 'LineWidth', 1.5);
plot(xk(1:2:16)',xk(2:2:16)','o','MarkerEdgeColor','blue', 'MarkerSize',8, 'LineWidth', 1.5);
grid on;
plot(xbest(1:2:16),xbest(2:2:16),'*', 'MarkerEdgeColor','blue', 'MarkerFaceColor','blue', 'MarkerSize',6, 'LineWidth', 1);

figure;
semilogy(1:(k-1), gk(1:k-1), 'LineWidth', 1.5);
title('$\left\Vert\nabla f\left(x_k\right)\right\Vert$ (LM method)','Interpreter','Latex')
grid on;
```

Parte III

1 Task 1

Pretende-se encontrar uma expressão em forma fechada para a distância do ponto p ao disco. Para isso, calcula-se a projeção do ponto p no disco – y – e a distância do ponto até à projeção,

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p-y\|_2 \\ \text{subject to} & y \in D(c,r). \end{array}$$

pode ser escrito como

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p-y\|_2 \\ \text{subject to} & \|y-c\|_2 - r \leq 0 \end{array}$$

De modo a tornar a função de custo, assim como a restrição, diferenciáveis eleva-se ao quadrado, obtendo-se:

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p-y\|_2^2 \\ \text{subject to} & \|y-c\|_2^2 - r^2 \leq 0 \end{array}$$

em que $f(y) = \|p-y\|_2^2$ e $g(y) = \|y-c\|_2^2 - r^2$.

Segundo o método de KKT:

$$\begin{cases} \nabla f(y) + \nabla g(y)\mu = 0 \\ g(y) \leq 0 \\ \mu \geq 0 \\ g(y)\mu = 0 \end{cases}$$

Para o caso deste problema:

$$\begin{cases} -(p+y) + (y-c)\mu* = 0 \\ \|y-c\|_2 \leq r \\ \mu \geq 0 \\ g(y)\mu = 0 \end{cases} \tag{10}$$

Na resolução pelo método de KKT, são considerados dois casos:

1. $g(y) < 0$
2. $g(y) = 0$

Para o primeiro caso, a partir da última equação de 10 impõem-se $\mu = 0$, e da primeira impõe-se $\nabla f(y) = 0 \leftrightarrow y = p$. Resumem-se os resultados para este caso em 11.

$$\begin{cases} p = y \\ \|y - c\| < r => \|p - c\| < r \\ \mu = 0 \end{cases} \tag{11}$$

Ou seja, se o ponto p se encontrar dentro do disco, a projeção do ponto p no disco é o próprio ponto.

Para o segundo caso apresentado, quando $g(y) = 0$ tem-se que:

$$\begin{cases} \nabla f(y) = -\nabla g(y)\mu \\ \|y - c\|_2 = r \\ \mu \geq 0 \end{cases} \tag{12}$$

A partir da primeira equação de 12, consegue-se chegar a uma relação de y com μ:

$$-(p - y) = -\mu(y - c) \leftrightarrow y = \frac{p + \mu c}{1 + \mu} \tag{13}$$

Conjugando (13) com a segunda igualdade de 12 obtém-se:

$$\left\| \frac{p + \mu c}{1 + \mu} - c \right\|_2 = r \leftrightarrow \left\| \frac{p - c}{1 + \mu} \right\|_2 = r \leftrightarrow \frac{\|p - c\|_2}{\|1 + \mu\|_2} = r$$

obtendo-se assim dois valores possíveis como o valor de μ:

$$\mu = -1 \pm \frac{\|p - c\|_2}{r}$$

Como uma restrição imposta é que $\mu \geq 0$ a única destas soluções que é válida é a que $\mu = -1 + \frac{\|p-c\|_2}{r}$ com a restrição de que $\|p - c\|_2 \geq r$, pois $\mu \geq 0$.

Encontrado o valor de μ, substitui-se o seu valor na equação 13 de modo a encontrar o valor de y:

$$y = \frac{p + c \cdot (-1 + \|p - c\|_2 / r)}{\|p - c\|_2 / r}$$

definindo-se assim, através de manipulação matemática, a projeção do ponto p no disco, quando este não se encontra no seu interior, como:

$$y = \frac{r}{\|p - c\|_2}(p - c) + c$$

Juntando os dois casos especificados anteriormente, tem-se:

$$y = \begin{cases} p & \text{se } \|p - c\|_2 \leq r \\ c + \frac{r}{\|p-c\|_2}(p - c) & \text{se } \|p - c\|_2 > r \end{cases}$$

Que se pode escrever em forma fechada como:

$$y = c + \frac{p-c}{\|p-c\|_2} \min(\|p-c\|_2, r)$$

2 Task 2

Para esta tarefa, optou-se pelo primeiro problema, por apresentar duas normas ℓ_2^2 que, como se verá adiante, podem ser simplificadas de forma semelhante. Note-se que esta demonstração tem por base o reconhecimento imediato de que a $\|x\|_2^2 = x^T x$.

Para abordar este problema, começou-se por juntar x e u num vetor coluna z, tal que:

$$z = \begin{bmatrix} p_1(0) \\ p_2(0) \\ v_1(0) \\ v_2(0) \\ p_1(1) \\ \vdots \\ v_2(T) \\ u_1(0) \\ u_2(0) \\ \vdots \\ u_2(T-1) \end{bmatrix} \in \mathbf{R}^{6T+4}$$

Para se poder utilizar os elementos deste vetor, criam-se as matrizes $E_{x_t} \in \mathbf{R}^{4\times(6T+4)}$, $E_{u_t} \in \mathbf{R}^{2\times(6T+4)}$ e $E_{u_{dt}} \in \mathbf{R}^{2\times(6T+4)}$ tais que, por exemplo, $x(0) = E_{x_0} z$:

$$E_{x_t} = \begin{array}{cccccc} & & t & & & \\ [0 & \dots & I & 0 & \dots & 0] \end{array}$$

$$E_{u_t} = \begin{array}{ccccccc} & & T & & t & & \\ [0 & \dots & 0 & \dots & I & \dots & 0] \end{array}$$

$$E_{u_{dt}} = E_{u_t} - E_{u_{t-1}} = \begin{array}{cccccccc} & & T & & & t & & \\ [0 & \dots & 0 & \dots & -I & I & \dots & 0] \end{array}$$

Neste momento, pode-se reescrever facilmente o primeiro somatório de forma matricial:

$$\begin{aligned}
\sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 = \sum_{k=1}^{K} \|E_{x_k} z - w_k\|_2^2 &= \sum_{k=1}^{K} \left(z^T E_{x_k}^T - w_k^T\right)\left(E_{x_k} z - w_k\right) \\
&= \sum_{k=1}^{K} z^T E_{x_k}^T E_{x_k} z - 2\sum_{k=1}^{K} z^T E_{x_k}^T w_k + \sum_{k=1}^{K} \|w_k^T w_k\|_2^2 \\
&= z^T Q z - 2z^T v + \sum_{k=1}^{K} \|w_k^T w_k\|_2^2,
\end{aligned}$$

com $Q = \sum_{k=1}^{K} E_{x_k}^T E_{x_k} \in \mathbf{R}^{(6T+4)\times(6T+4)}$ e $v = \sum_{k=1}^{K} E_{x_k}^T w_k \in \mathbf{R}^{6T+4}$. Note-se que estes são preenchidos com zeros nas entradas não correspondentes a valores de τ.

Consegue-se, ainda, simplificar o somatório em u:

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 = \lambda \sum_{t=1}^{T-1} \|E_{u_{dt}} z\|_2^2 = \lambda \sum_{t=1}^{T-1} z^T E_{u_{dt}}^T E_{u_{dt}} z = z^T \lambda S z,$$

com $S = \sum_{t=1}^{T-1} E_{u_{dt}} E_{u_{dt}}$.

Assim, pode-se reescrever a função de custo de forma simplificada, sem os termos constantes e dividida por 2:

$$\frac{1}{2} z^T (Q + \lambda S) z - z^T v.$$

É, agora, necessário encontrar uma expressão para as restrições. Para tal, pode-se analisar a recursividade da terceira restrição e obter uma expressão matricial.

$$\begin{cases}
x(1) = Ax(0) + Bu(0) \\
x(2) = A^2 x(0) + ABu(0) + Bu(1) \\
x(3) = A^3 x(0) + A^2 Bu(0) + ABu(1) + Bu(2) \\
\qquad \vdots \\
x(T) = A^T x(0) + A^{T-1} Bu(0) + A^{T-2} Bu(1) + \cdots + Bu(T)
\end{cases}$$

Esta restrição pode ser reescrita, em conjunto com as outras duas, como uma equação matricial da forma $Cz = d$, com:

$$C = \begin{array}{c} \begin{array}{ccccccc} & & & T & \phantom{A^{T-1}B} & & \end{array} \\ \begin{bmatrix} I & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & I & 0 & \dots & 0 \\ -I & 0 & \dots & A^T & A^{T-1}B & \dots & B \end{bmatrix} \end{array} \in \mathbf{R}^{12\times(6T+4)} \qquad d = \begin{bmatrix} x_{initial} \\ x_{final} \\ 0 \end{bmatrix} \in \mathbf{R}^{12}$$

Resumindo, e retirando a parcela independente da função de custo, bem como dividindo-a por 2, este problema pode ser reescrito da forma

$$\begin{array}{ll} \underset{z}{\text{minimize}} & \frac{1}{2} z^T (Q + \lambda S) z - z^T v \\ \text{subject to} & Cz = d. \end{array}$$

Para encontrar a solução em forma fechada, resolve-se o sistema KKT:

$$\begin{cases} (Q+\lambda S)z^* - v + C^T\mu^* = 0 \\ Cz^* = d \end{cases} \leftrightarrow \begin{cases} z^* = (Q+\lambda S)^{-1}(v - C^T\mu^*) \\ C(Q+\lambda S)^{-1}(v - C^T\mu^*) = d \end{cases} \leftrightarrow$$

$$\begin{cases} z^* = (Q+\lambda S)^{-1}(v - C^T\mu^*) \\ C(Q+\lambda S)^{-1}v - C(Q+\lambda S)^{-1}C^T\mu^* = d \end{cases} \leftrightarrow \begin{cases} z^* = (Q+\lambda S)^{-1}(v - C^T\mu^*) \\ \mu^* = \left(C\left(Q+\lambda S\right)^{-1}C^T\right)^{-1}(C(Q+\lambda S)^{-1}v - d) \end{cases}$$