Optimization and Algorithms
Project report

Group 42
José Neves 89683, Leonardo Pedroso 89691, Gustavo Bakker 100660

1 Part 1

In this section, the results for the robot trajectory optimization problem are shown, demonstrating how we can obtain simple control signals so that the robot trajectory satisfies 4 wishes: i) initial and final state; ii) boundededness of the control signal; iii) trajectory waypoints; and iv) regularization of control signal. The problem is formulated for three distinct regularizers i) ℓ_2^2, ii) ℓ_2, and iii) ℓ_1. The results for the optimal positions of the robot and the optimal variation of the control signal are shown. Also the number of control signal changes and the mean deviation of the trajectory in relation to the waypoints are computed and analysed, for different values of the regularizer weight $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$.

1.1 Task 1

In this task the ℓ_2^2 regularizer is used for the formulation of the optimization problem, which is in this case

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1. \end{array} \tag{1}$$

For each value of λ, the plot of the optimal trajectory and control signal are shown in Figs. 1–7. The mean deviation of the optimal trajectory in relation to the desired waypoints, as well as the number of control signal changes are depicted in Table 1. It is said that there is a signal control change if

$$||u(t) - u(t-1)||_2 > 10^{-4} \tag{2}$$

for $t \in \{1, \ldots, T-1\}$. On one hand, it is noticeable, analysing Table 1, that for this particular regularizer the number of control signal changes remains unchanged as λ varies. In fact, it is considered, according to (2), that there is a change between every consecutive values of the discrete control signal, no matter the weight of the regularizer. On the other hand, increasing the value of λ more weight is given to the regularizer in comparison to the weight given to waypoint tracking performance. It is, thus, expected that the optimal control signal varies more slowly, as λ is increased, leading to a smother control signal, at the expense of a poorer waypoint tracking. As a matter of fact, it is visible in Table 1 that as λ increases so does the mean deviation of the optimal trajectory in relation to the waypoints, as predicted. While in Fig. 1 the waypoints are followed accurately at the expense of a rapidly varying control signal, in Fig. 7 the control signal is very smooth but the waypoint tracking performance is very poor. For this reason, the value of λ suitable for the application to the robot is a compromise between waypoint tracking performance and smoothness of the control signal.

The following MATAB script was used to solve this task.

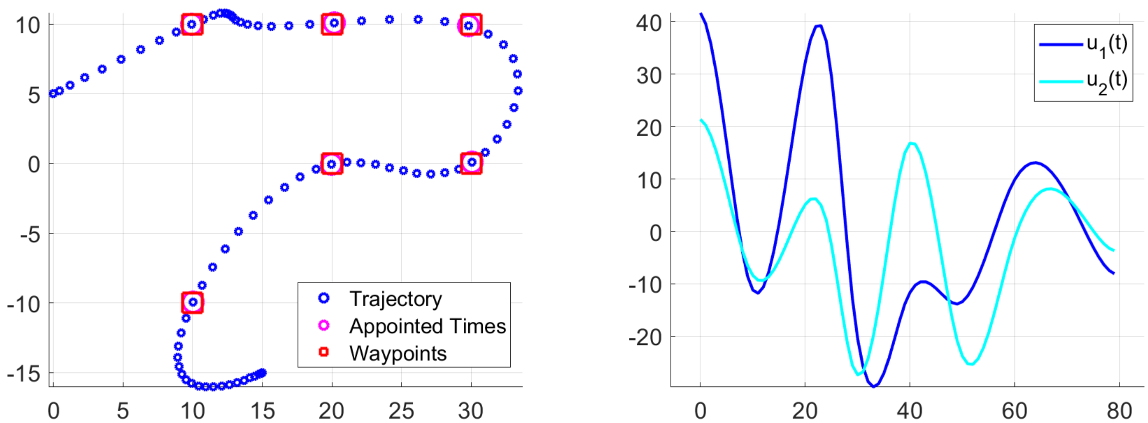
```
% part1task1.m
% Optimal robot trajectory and control signal using l_2^2 regularizer

clear
close all

mkdir('results')

% Given dynamics matrices
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];

% Given parameters
T = 80;
Umax = 100;
tau = [10 25 30 40 50 60];
w = [[10;10] [20;10] [30;10] [30;0] [20;0] [10;-10]];
pInitial = [0; 5];
pFinal = [15;-15];

% Used parameters
lambda_log = -3:3;
% selects the position components of x
E=[eye(2) zeros(2)];
% when multiplied by u at the right, operates the difference between
% consecutive control inputs
u_differences = diag(-[ones(T-1,1);0])+diag(ones(T-1,1),-1);
% when multiplied by x at the right, gives only the states after the first
% time instant
next_x = [zeros(1,T); eye(T)];
```

```
% when multiplied by x at the right, gives only the states until the final
% instant
present_x = [eye(T); zeros(1,T)];
% transforms Umax into a vector so that it can be compared with the control
% signal without using a for loop
Umax_vec = ones(1,T)*Umax;

% control changes
u_changes = zeros(length(lambda_log),1);
% mean deviations
mean_devs = zeros(length(lambda_log),1);

% Loop over lambdas
for i = 1:length(lambda_log)
    % ------------------- Optimization -------------------
    cvx_begin quiet
        % we want to optimize variables u and x
        variables x(4,T+1) u(2,T)

        % objective function
        minimize(sum(sum_square(E*x(:,tau+1)-w))+(10^lambda_log(i))*...
            sum(sum_square(u*u_differences)));

        %subject to
        x(:,1) == [pInitial;0;0]
        x(:,T+1) == [pFinal;0;0]
        norms(u) ≤ Umax_vec
        x*next_x == A*x*present_x+B*u;
    cvx_end

    % ------------------- Plot Result -------------------
    % plot the trajectory
    figure();
    hold on;
    set(gca,'FontSize',16);
    scatter(x(1,:),x(2,:),20,'blue','LineWidth',2);
    scatter(x(1,tau+1),x(2,tau+1),200,'magenta','LineWidth',2);
    scatter(w(1,:),w(2,:),300,'red','s','LineWidth',2)
    axis equal
    grid on;
    legend({'Trajectory', 'Appointed Times', 'Waypoints'},'Location',...
        'best');
    % saves the trajectory plot
    saveas(gcf,sprintf('./results/trajectory_l22_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % plot the control signal
    figure()
    hold on;
    set(gca,'FontSize',16);
```

```
    plot((0:T-1),u(1,:),'blue','LineWidth',2);
    plot(0:(T-1),u(2,:),'cyan','LineWidth',2);
    axis equal
    grid on;
    legend('u_1(t)', 'u_2(t)');
    % saves the control changes plot
    saveas(gcf,sprintf('./results/controlSignal_l22_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % sum up control changes
    for t = 1:T-1
        if norm(u(:,t+1)-u(:,t))>1e-4
            u_changes(i) = u_changes(i) + 1;
        end
    end

    % sum up mean deviations
    mean_devs(i) = (1/length(w))*sum(sqrt(sum_square(E*x(:,tau+1)-w)));
end

disp(u_changes) % displays control changes in the console
disp(mean_devs) % displays mean deviations in the console
```

Table 1: Control signal and mean deviation using ℓ_2^2 regularizer for each λ values.

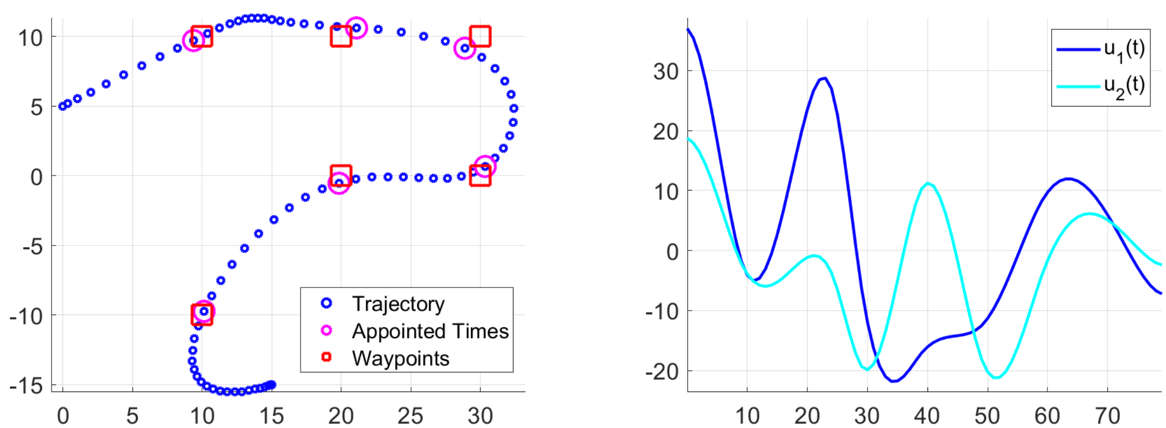
λ	Control changes	Mean deviation
10^{-3}	79	0.1257
10^{-2}	79	0.8242
10^{-1}	79	2.1958
10^{0}	79	3.6826
10^{1}	79	5.6317
10^{2}	79	10.9041
10^{3}	79	15.3304



(a) Optimal trajectory.

(b) Optimal control signal.

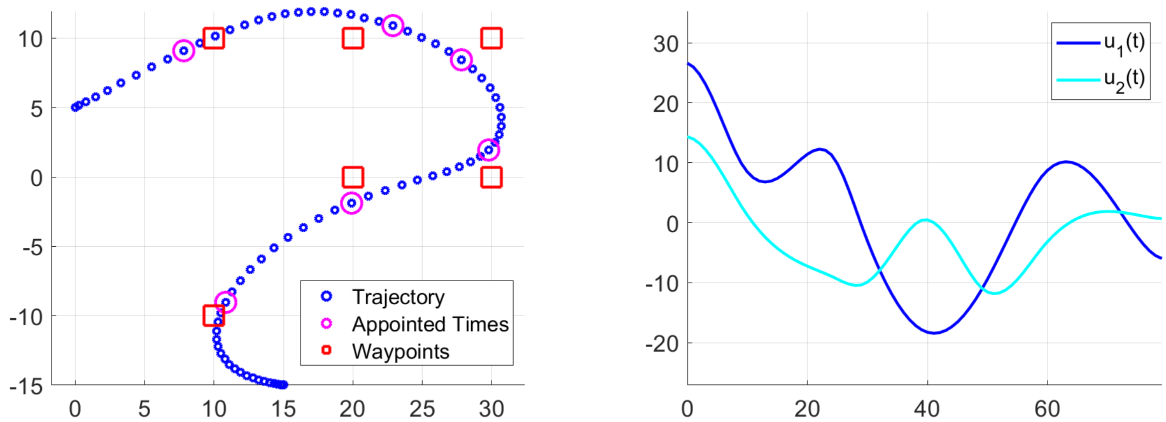
Figure 1: Case $\lambda = 10^{-3}$ with ℓ_2^2 regularizer.



(a) Optimal trajectory.

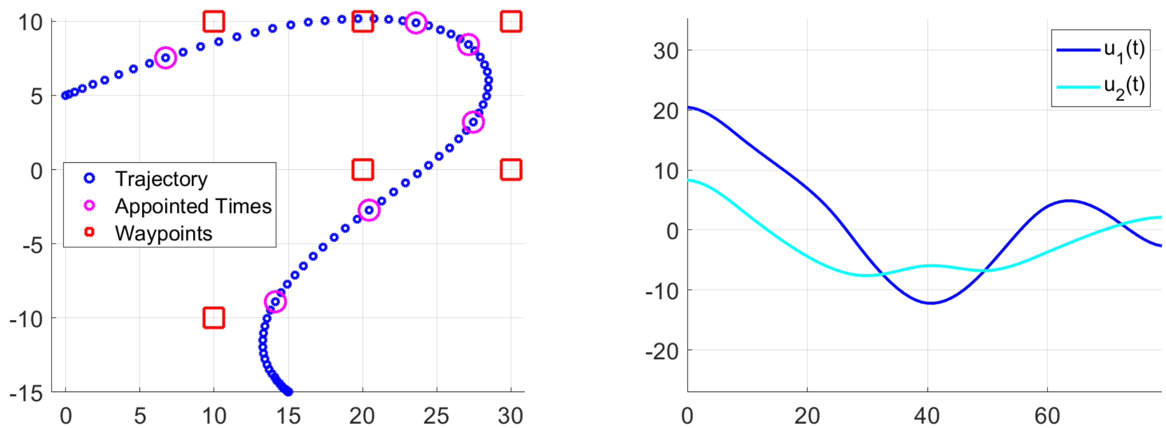
(b) Optimal control signal.

Figure 2: Case $\lambda = 10^{-2}$ with ℓ_2^2 regularizer.



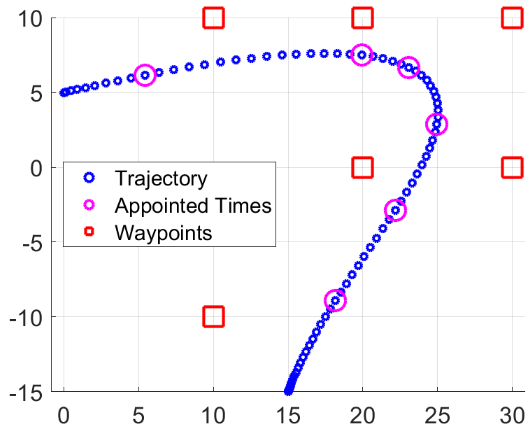
(a) Optimal trajectory. (b) Optimal control signal.

Figure 3: Case $\lambda = 10^{-1}$ with ℓ_2^2 regularizer.

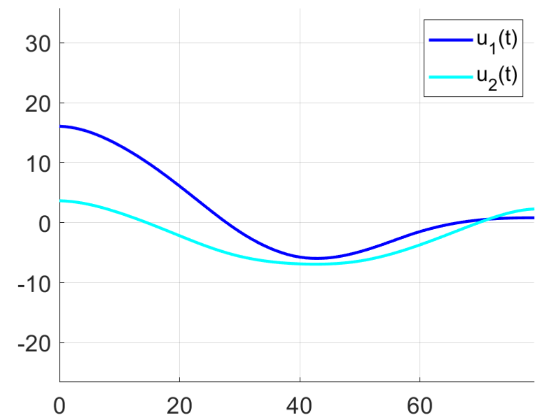


(a) Optimal trajectory. (b) Optimal control signal.

Figure 4: Case $\lambda = 10^0$ with ℓ_2^2 regularizer.

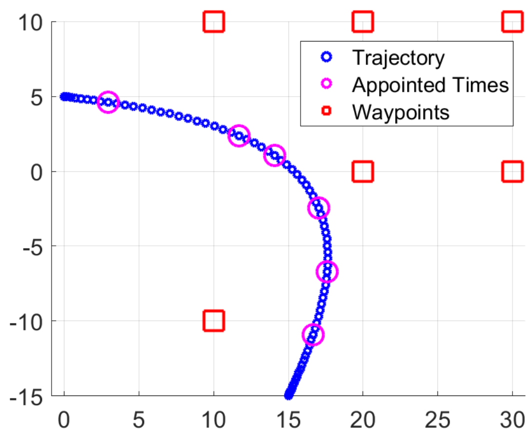


(a) Optimal trajectory.

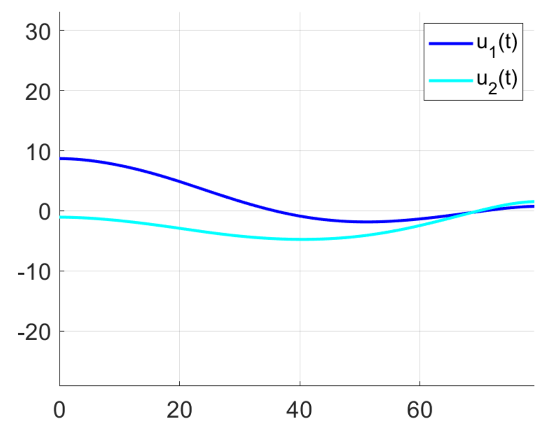


(b) Optimal control signal.

Figure 5: Case $\lambda = 10^1$ with ℓ_2^2 regularizer.

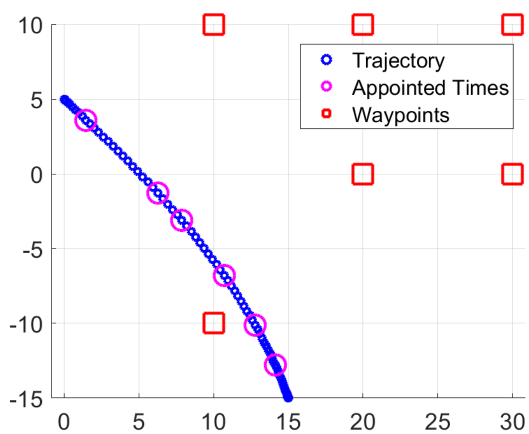


(a) Optimal trajectory.

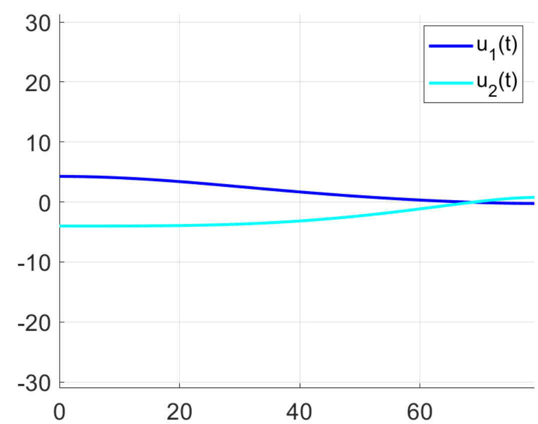


(b) Optimal control signal.

Figure 6: Case $\lambda = 10^2$ with ℓ_2^2 regularizer.



(a) Optimal trajectory.



(b) Optimal control signal.

Figure 7: Case $\lambda = 10^3$ with ℓ_2^2 regularizer.

1.2 Task 2

In this task the ℓ_2 regularizer is used for the formulation of the optimization problem, which is in this case

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1. \end{array} \tag{3}$$

For each value of λ, the plot of the optimal trajectory and control signal are shown in Figs. 8–14. The mean deviation of the optimal trajectory in relation to the desired waypoints, as well as the number of control signal changes are depicted in Table 2, where a control signal change is defined as in (2).

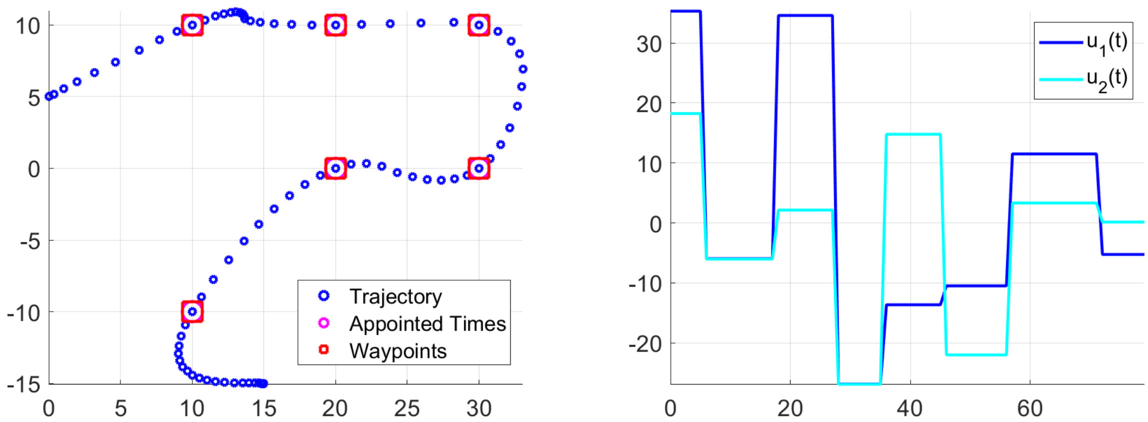
Contrarily of what is observed for the ℓ_2^2 regularizer in the previous task, the number of control signal changes for this regularizer varies with λ. In fact, it is important to remark the differences between the shape of the control signal. While the control signal is smooth for the ℓ_2^2 regularizer, for the ℓ_2 the control signal is piecewise constant. Furthermore, it is also noticeable that, most of the times, a transition in one of the components of the control signal vector is accompanied with a transition in the other component in the same time instant. This facts are discussed in detail in task 4, as well as other aspects of the comparison of the three regularizers. As λ increases more weight is given to the regularizer, thus the number of control signals changes has a tendency to decrease as λ increases. Conversely, as analysed in task 1, the greater the weight given to the regularizer the poorer the waypoint tracking performance is.

The following MATAB script was used to solve this task.

```
% part1task2.m
% Optimal robot trajectory and control signal using l_2 regularizer

clear
close all

mkdir('results')

% Given dynamics matrices
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];
```

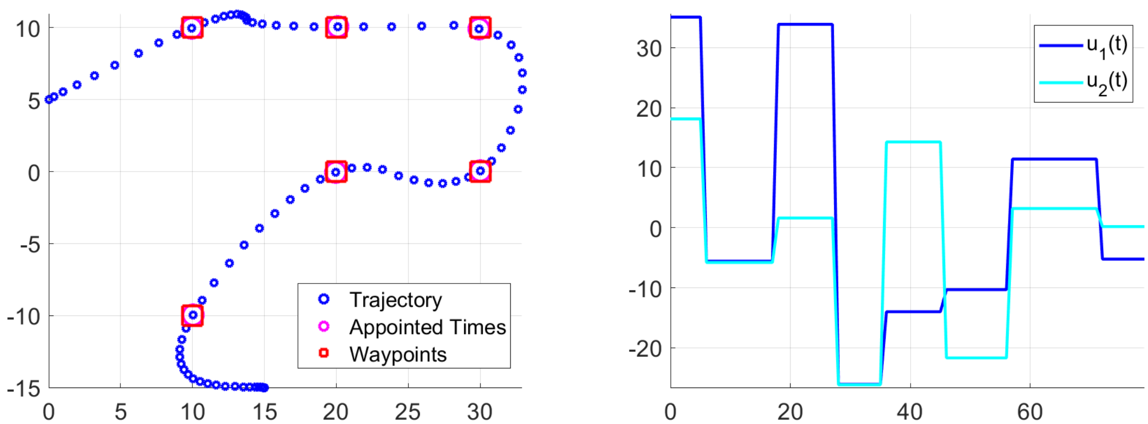
```
% Given parameters
T = 80;
Umax = 100;
tau = [10 25 30 40 50 60];
w = [[10;10] [20;10] [30;10] [30;0] [20;0] [10;-10]];
pInitial = [0; 5];
pFinal = [15;-15];

% Used parameters
lambda_log = -3:3;
% selects the position components of x
E=[eye(2) zeros(2)];
% when multiplied by u at the right, operates the difference between
% consecutive control inputs
u_differences = diag(-[ones(T-1,1);0])+diag(ones(T-1,1),-1);
% when multiplied by x at the right, gives only the states after the first
% time instant
next_x = [zeros(1,T); eye(T)];
% when multiplied by x at the right, gives only the states until the final
% instant
present_x = [eye(T); zeros(1,T)];
% transforms Umax into a vector so that it can be compared with the control
% signal without using a for loop
Umax_vec = ones(1,T)*Umax;

% control changes
u_changes = zeros(length(lambda_log),1);
% mean deviations
mean_devs = zeros(length(lambda_log),1);

% Loop over lambdas
for i = 1:length(lambda_log)
    % ------------------ Optimization ------------------
    cvx_begin quiet
        % we want to optimize variables u and x
        variables x(4,T+1) u(2,T)

        % objective function
        minimize(sum(sum_square(E*x(:,tau+1)-w))+(10^lambda_log(i))*...
            sum(norms(u*u_differences)));

        %subject to
        x(:,1) == [pInitial;0;0]
        x(:,T+1) == [pFinal;0;0]
        norms(u) <= Umax_vec
        x*next_x == A*x*present_x+B*u;
    cvx_end

    % ------------------ Plot Result ------------------
    % plot the trajectory
```

```
    figure();
    hold on;
    set(gca,'FontSize',16);
    scatter(x(1,:),x(2,:),20,'blue','LineWidth',2);
    scatter(x(1,tau+1),x(2,tau+1),200,'magenta','LineWidth',2);
    scatter(w(1,:),w(2,:),300,'red','s','LineWidth',2)
    axis equal
    grid on;
    legend({'Trajectory', 'Appointed Times', 'Waypoints'},'Location',...
        'best');
    % saves the trajectory plot
    saveas(gcf,sprintf('./results/trajectory_l2_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % plot the control signal
    figure()
    hold on;
    set(gca,'FontSize',16);
    plot((0:T-1),u(1,:),'blue','LineWidth',2);
    plot(0:(T-1),u(2,:),'cyan','LineWidth',2);
    axis equal
    grid on;
    legend('u_1(t)', 'u_2(t)');
    % saves the control changes plot
    saveas(gcf,sprintf('./results/controlSignal_l2_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % sum up control changes
    for t = 1:T-1
        if norm(u(:,t+1)-u(:,t))>1e-4
            u_changes(i) = u_changes(i) + 1;
        end
    end

    % sum up mean deviations
    mean_devs(i) = (1/length(w))*sum(sqrt(sum_square(E*x(:,tau+1)-w)));
end

disp(u_changes) % displays control changes in the console
disp(mean_devs) % displays mean deviations in the console
```



(a) Optimal trajectory. (b) Optimal control signal.

Figure 8: Case $\lambda = 10^{-3}$ with ℓ_2 regularizer.

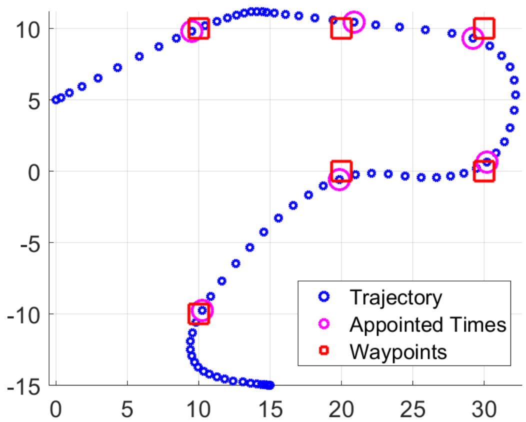


(a) Optimal trajectory. (b) Optimal control signal.

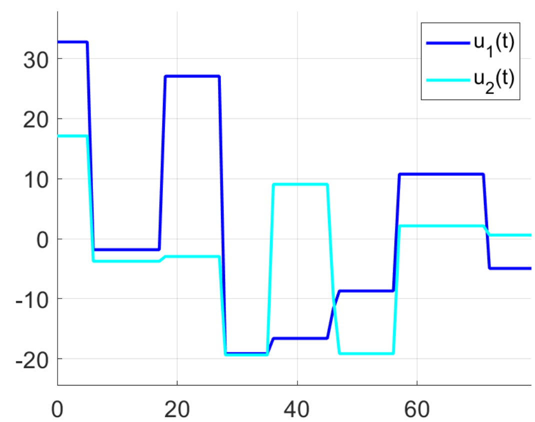
Figure 9: Case $\lambda = 10^{-2}$ with ℓ_2 regularizer.

Table 2: Control signal and mean deviation using ℓ_2 regularizer for each λ values.

λ	Control changes	Mean deviation
10^{-3}	7	0.0075
10^{-2}	7	0.0747
10^{-1}	8	0.7021
10^{0}	4	2.8876
10^{1}	3	5.3689
10^{2}	2	12.5914
10^{3}	1	16.2266

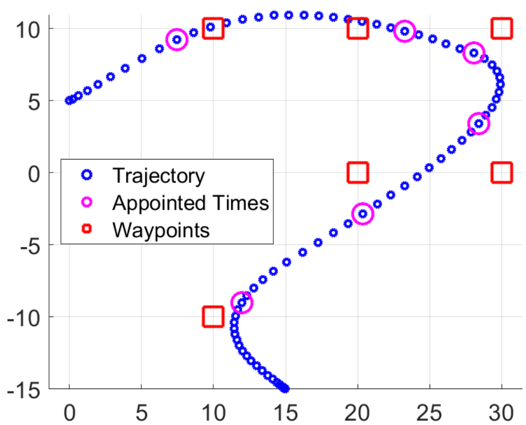


(a) Optimal trajectory.

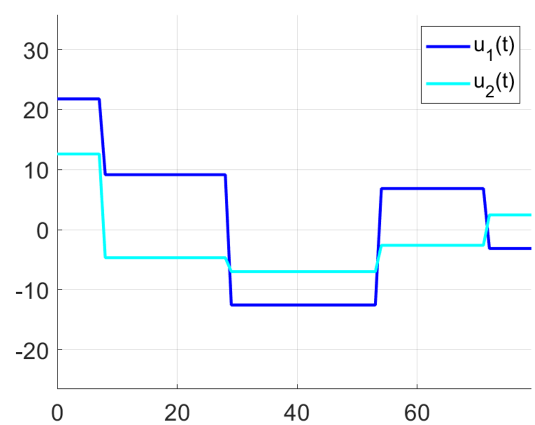


(b) Optimal control signal.

Figure 10: Case $\lambda = 10^{-1}$ with ℓ_2 regularizer.

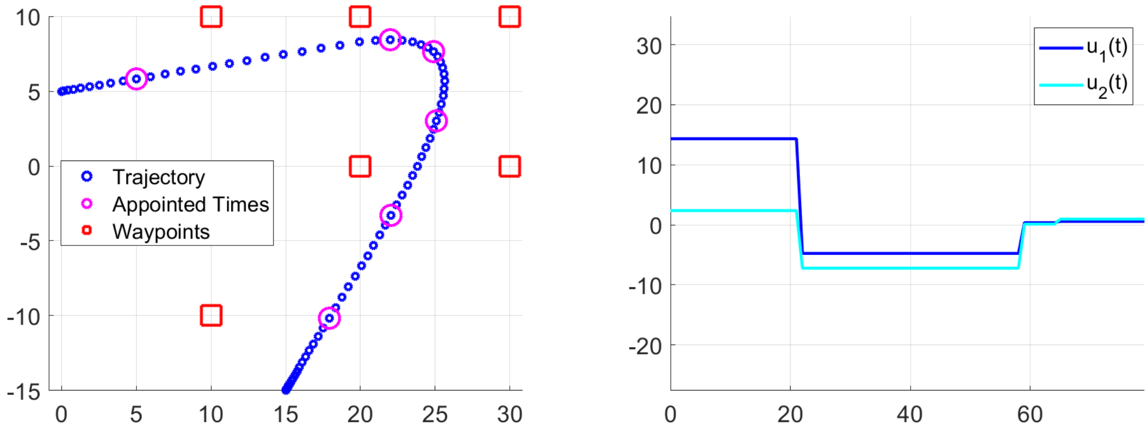


(a) Optimal trajectory.



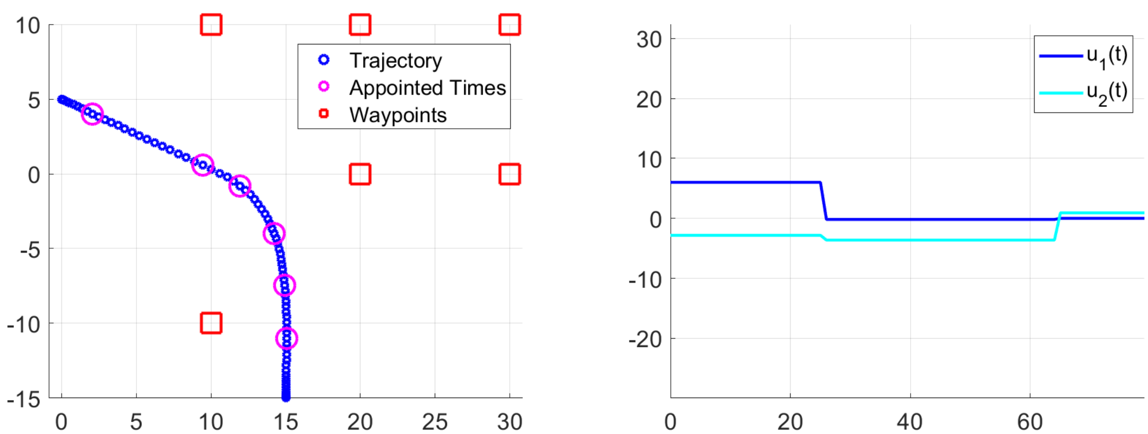
(b) Optimal control signal.

Figure 11: Case $\lambda = 10^{0}$ with ℓ_2 regularizer.



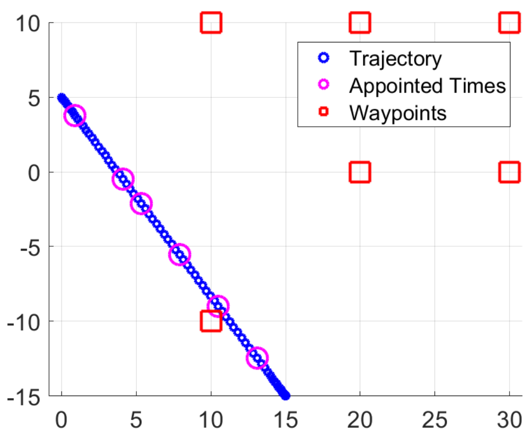
(a) Optimal trajectory. (b) Optimal control signal.

Figure 12: Case $\lambda = 10^1$ with ℓ_2 regularizer.

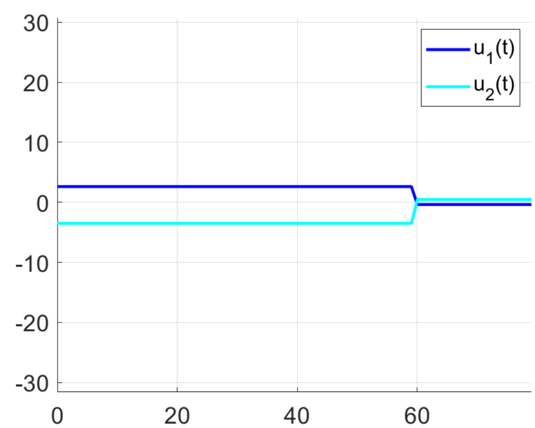


(a) Optimal trajectory. (b) Optimal control signal.

Figure 13: Case $\lambda = 10^2$ with ℓ_2 regularizer.



(a) Optimal trajectory.



(b) Optimal control signal.

Figure 14: Case $\lambda = 10^3$ with ℓ_2 regularizer.

1.3 Task 3

In this task the ℓ_1 regularizer is used for the formulation of the optimization problem, which is in this case

$$\begin{array}{ll}\underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_1 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1. \end{array} \tag{4}$$

For each value of λ, the plot of the optimal trajectory and control signal are shown in Figs. 8–14. The mean deviation of the optimal trajectory in relation to the desired waypoints, as well as the number of control signal changes are depicted in Table 2, where a control signal change is defined as in (2).

Contrarily of what is observed for the ℓ_2^2 regularizer in the previous task, the number of control signal changes for this regularizer varies with λ. While for the ℓ_2^2 regularizer the control signal is smooth, for the ℓ_1 regularizer, similarly to the ℓ_2, the control signal is piecewise constant. Furthermore, it is also noticeable that, unlinke what it verified for the ℓ_2, the changes in the components of the control signal are not generally simultaneous. This fact is explored thoroughly in the nest task. Again, as λ increases more weight is given to the regularizer, thus the number of control signals changes has a tendency to decrease as λ increases. Conversely, as analysed in task 1, the greater the weight given to the regularizer the poorer the waypoint tracking performance is.

The following MATAB script was used to solve this task.

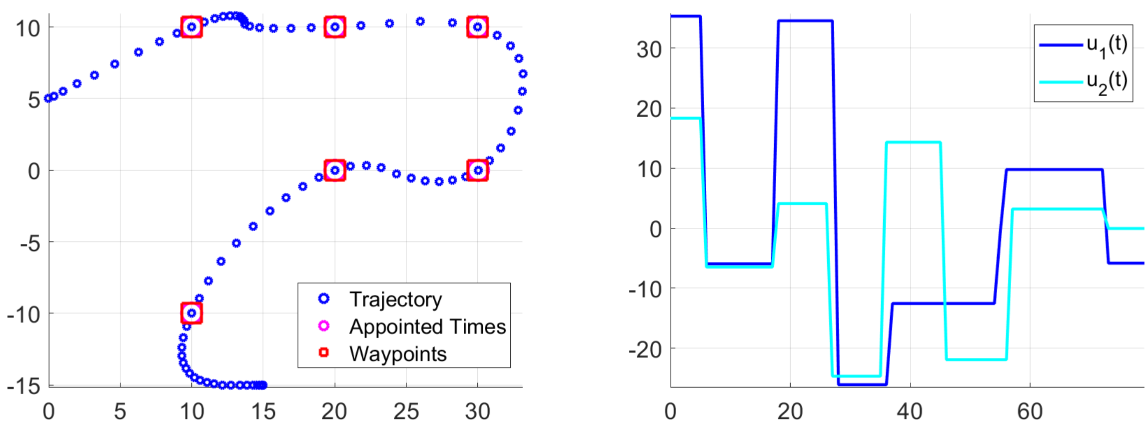
```
% part1task3.m
% Optimal robot trajectory and control signal using l_1 regularizer

clear
close all

mkdir('results')

% Given dynamics matrices
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];

% Given parameters
```

```
T = 80;
Umax = 100;
tau = [10 25 30 40 50 60];
w = [[10;10] [20;10] [30;10] [30;0] [20;0] [10;-10]];
pInitial = [0; 5];
pFinal = [15;-15];

% Used parameters
lambda_log = -3:3;
% selects the position components of x
E=[eye(2) zeros(2)];
% when multiplied by u at the right, operates the difference between
% consecutive control inputs
u_differences = diag(-[ones(T-1,1);0])+diag(ones(T-1,1),-1);
% when multiplied by x at the right, gives only the states after the first
% time instant
next_x = [zeros(1,T); eye(T)];
% when multiplied by x at the right, gives only the states until the final
% instant
present_x = [eye(T); zeros(1,T)];
% transforms Umax into a vector so that it can be compared with the control
% signal without using a for loop
Umax_vec = ones(1,T)*Umax;

% control changes
u_changes = zeros(length(lambda_log),1);
% mean deviations
mean_devs = zeros(length(lambda_log),1);

% Loop over lambdas
for i = 1:length(lambda_log)
    % ------------------ Optimization ------------------
    cvx_begin quiet
        % we want to optimize variables u and x
        variables x(4,T+1) u(2,T)

        % objective function
        minimize(sum(sum_square(E*x(:,tau+1)-w))+(10^lambda_log(i))*...
            sum(norms(u*u_differences,1)));

        %subject to
        x(:,1) == [pInitial;0;0]
        x(:,T+1) == [pFinal;0;0]
        norms(u) <= Umax_vec
        x*next_x == A*x*present_x+B*u;
    cvx_end

    % ------------------ Plot Result ------------------
    % plot the trajectory
    figure();
    hold on;
```

```
    set(gca,'FontSize',16);
    scatter(x(1,:),x(2,:),20,'blue','LineWidth',2);
    scatter(x(1,tau+1),x(2,tau+1),200,'magenta','LineWidth',2);
    scatter(w(1,:),w(2,:),300,'red','s','LineWidth',2)
    axis equal
    grid on;
    legend({'Trajectory', 'Appointed Times', 'Waypoints'},'Location',...
        'best');
    % saves the trajectory plot
    saveas(gcf,sprintf('./results/trajectory_l1_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % plot the control signal
    figure()
    hold on;
    set(gca,'FontSize',16);
    plot((0:T-1),u(1,:),'blue','LineWidth',2);
    plot(0:(T-1),u(2,:),'cyan','LineWidth',2);
    axis equal
    grid on;
    legend('u_1(t)', 'u_2(t)');
    % saves the control changes plot
    saveas(gcf,sprintf('./results/controlSignal_l1_lambda_log_%d.png',...
        lambda_log(i)));
    hold off;

    % sum up control changes
    for t = 1:T-1
        if norm(u(:,t+1)-u(:,t))>1e-4
            u_changes(i) = u_changes(i) + 1;
        end
    end

    % sum up mean deviations
    mean_devs(i) = (1/length(w))*sum(sqrt(sum_square(E*x(:,tau+1)-w)));
end

disp(u_changes) % displays control changes in the console
disp(mean_devs) % displays mean deviations in the console
```

Table 3: Control signal and mean deviation using ℓ_1 regularizer for each λ values.

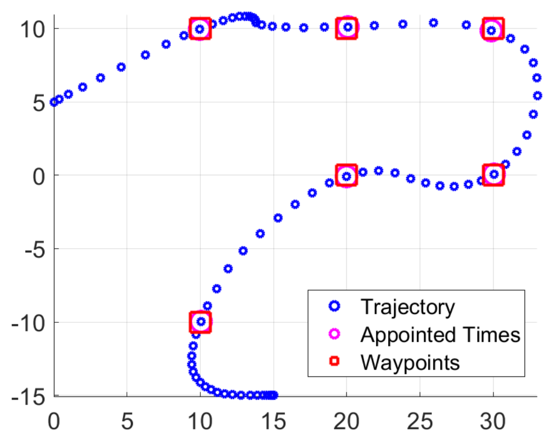
λ	Control signal	Mean deviation
10^{-3}	11	0.0107
10^{-2}	11	0.1055
10^{-1}	14	0.8863
10^{0}	9	2.8732
10^{1}	4	5.4361
10^{2}	2	13.0273
10^{3}	2	16.0463



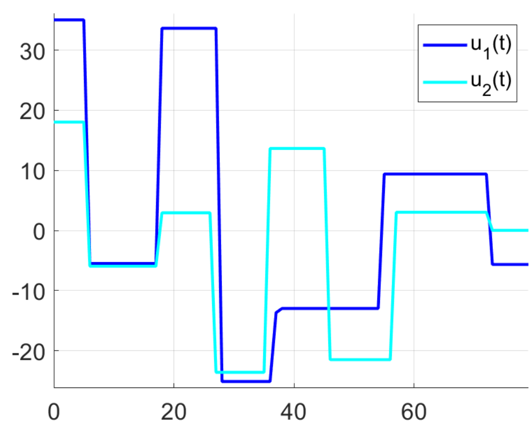
(a) Optimal trajectory.

(b) Optimal control signal.

Figure 15: Case $\lambda = 10^{-3}$ with ℓ_1 regularizer.

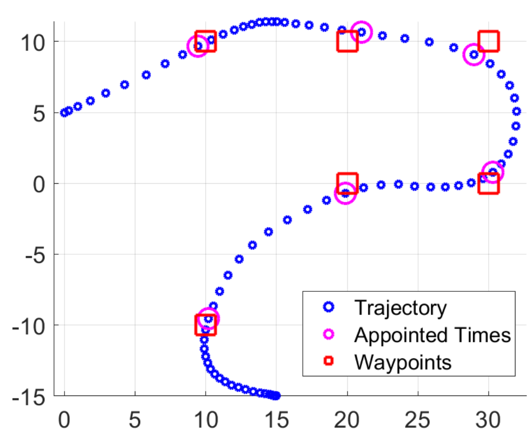


(a) Optimal trajectory.

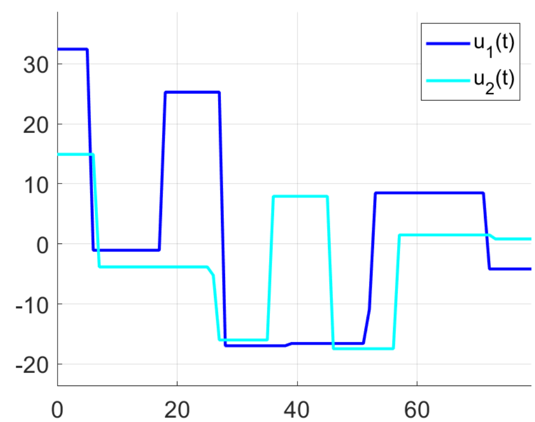


(b) Optimal control signal.

Figure 16: Case $\lambda = 10^{-2}$ with ℓ_1 regularizer.

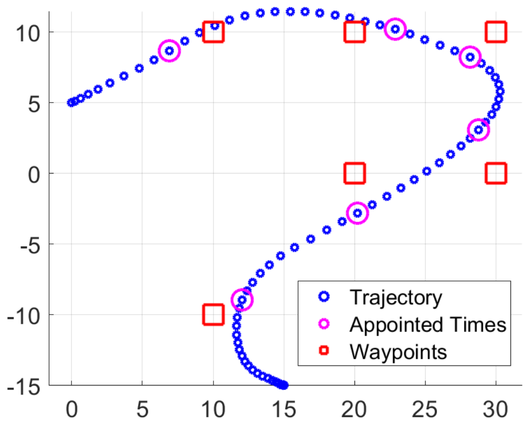


(a) Optimal trajectory.

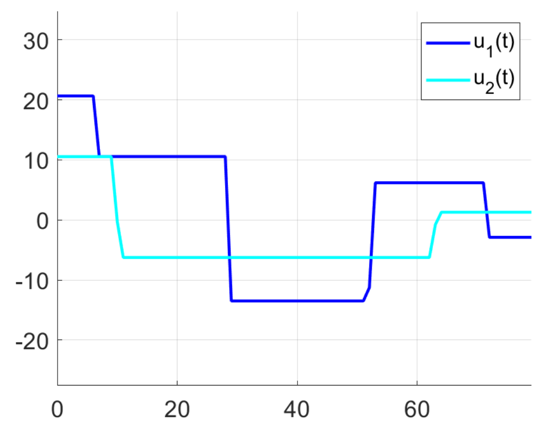


(b) Optimal control signal.

Figure 17: Case $\lambda = 10^{-1}$ with ℓ_1 regularizer.

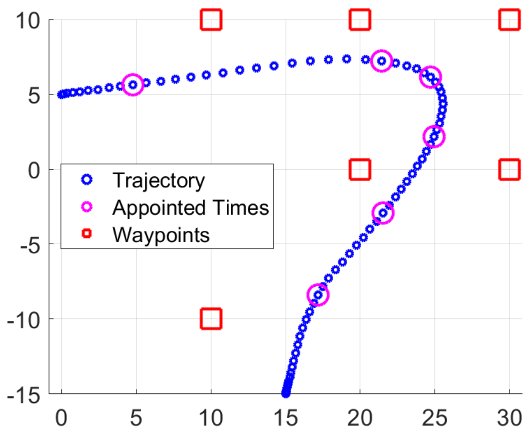


(a) Optimal trajectory.

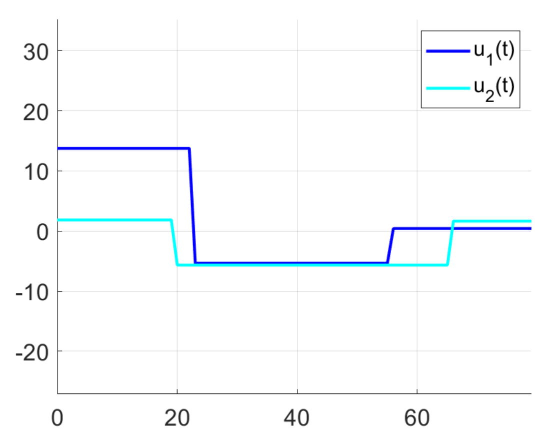


(b) Optimal control signal.

Figure 18: Case $\lambda = 10^0$ with ℓ_1 regularizer.

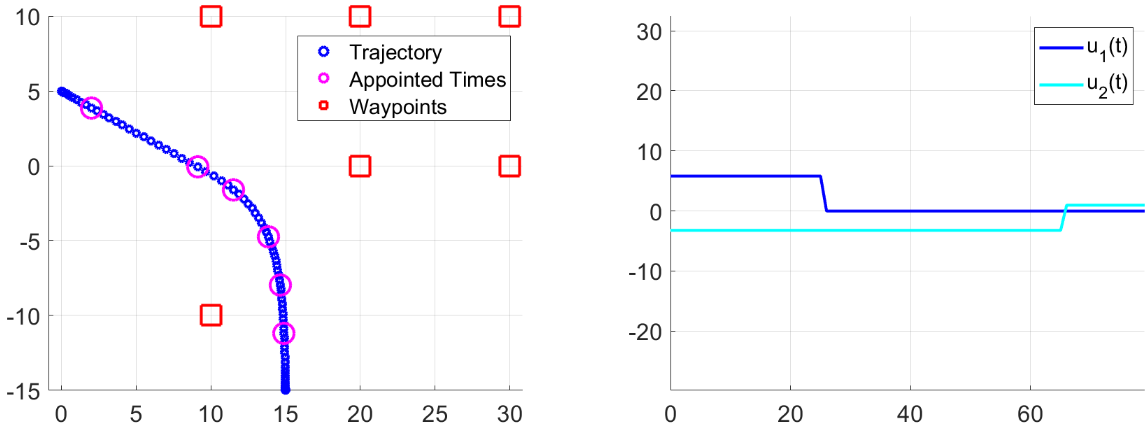


(a) Optimal trajectory.



(b) Optimal control signal.

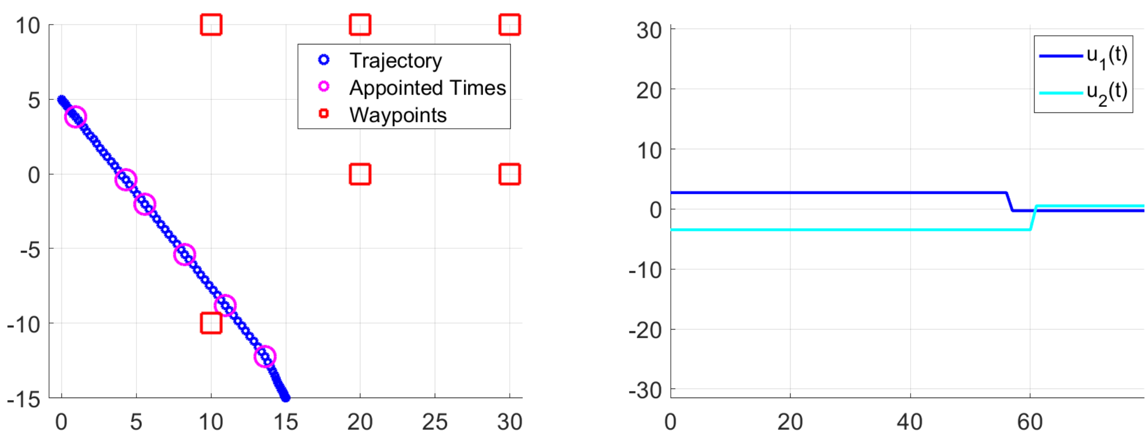
Figure 19: Case $\lambda = 10^1$ with ℓ_1 regularizer.



(a) Optimal trajectory.

(b) Optimal control signal.

Figure 20: Case $\lambda = 10^2$ with ℓ_1 regularizer.



(a) Optimal trajectory.

(b) Optimal control signal.

Figure 21: Case $\lambda = 10^3$ with ℓ_1 regularizer.

1.4 Task 4

First, it is important to analyse the differences in the shape of the optimal control signals obtained with each of the regularizers. On one hand, it is possible to notice that using the ℓ_2^2 regularizer, whatever the value of λ is, there is always the maximum number of changes of the optimal control signal, as can be observed in Table 4. In fact, as detailed in task 1, using this regularizer, the optimal control obtained is a smooth continuous signal. On the other hand, using the ℓ_2 and ℓ_1 regularizers, the signal obtained is piecewise constant. This difference in behavior can be explained analysing the evolution of each of the norms in a neighborhood near the origin. In fact, expanding each of the norms in first order Taylor series about $\mathbf{a} \in \mathbb{R}^n$ one obtains

$$\begin{aligned} ||\mathbf{x}||_2^2 &= ||\mathbf{a}||_2^2 + D_{\mathbf{x}}||\mathbf{x}||_2^2\big|_{\mathbf{a}}(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \\ &= ||\mathbf{a}||_2^2 + 2\mathbf{a}^T(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \end{aligned} \tag{5}$$

for the ℓ_2^2 regularizer,

$$\begin{aligned} ||\mathbf{x}||_2 &= ||\mathbf{a}||_2 + D_{\mathbf{x}}||\mathbf{x}||_2\big|_{\mathbf{a}}(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \\ &= ||\mathbf{a}||_2 + \frac{\mathbf{a}^T}{||\mathbf{a}||_2}(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \end{aligned} \tag{6}$$

for the ℓ_2 regularizer, and

$$\begin{aligned} ||\mathbf{x}||_1 &= ||\mathbf{a}||_1 + D_{\mathbf{x}}||\mathbf{x}||_1\big|_{\mathbf{a}}(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \\ &= ||\mathbf{a}||_1 + [\mathrm{sgn}(\mathbf{a}_1), \ldots, \mathrm{sgn}(\mathbf{a}_\mathrm{n})]\,(\mathbf{x}-\mathbf{a}) + \mathcal{O}((\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a})) \end{aligned} \tag{7}$$

for the ℓ_1 regularizer, where $\mathrm{sgn}(t)$ denotes the signal of $t \in \mathbb{R}^n$. In fact, given that the solution to the optimization problem aims at minimizing the regularizer terms, then, approximating each of the regularizers with their first order Taylor series about a point near the origin, one can more easily gather insight into the effect of each of the regularizers when they take small values. On one hand, notice that only the derivative of the norm of the l_2^2 regularizer is null at the origin. This means, that for the ℓ_2^2 regularizer terms, near the origin, changes in the vector $\mathbf{x}$ alter very slightly the value of the regularizer term, which is evident performing the expansion (5) about $\mathbf{a} = \mathbf{0}$. As a result, this regularizer is benevolent with small differences of $\mathbf{u}(t)-\mathbf{u}(t-1)$, therefore its use originates a smooth control signal. On the other hand, both the ℓ_2 and ℓ_1 reguralizer terms have a non null derivative at the origin. For this reason, no matter how small, if the difference $\mathbf{u}(t) - \mathbf{u}(t-1)$ is non null, then a change in vector $\mathbf{x}$ alters significantly the value of the regularizer term, which is evident performing the expansions (6) and (7) about $\mathbf{a} \to \mathbf{0}$. As a result, the differences $\mathbf{u}(t) - \mathbf{u}(t-1)$ are most of the time null, unless when absolutely necessary as imposed by the waypoint tracking cost. Therefore, for both this regularizers it is expected that the optimal control signal is piecewice constant. Besides, if the regulators were applied to a vector, instead of a difference of vectors, the optimal evolution of such vector would be sparse, in the sense that some of the entries would be null, which is of significant practical importance as far as applications to signal processing and machine learning are concerned. In conclusion, if a smooth control signal is sought, one must select a regularizer norm whose derivative at the origin is null. Conversely, if a piecewise optimal control signal is sought, one must selet a regularizer norm whose derivative at the origin is non null.

Table 4: Comparison between regularizers.

	Control signal			Mean deviation		
λ	ℓ_2^2	ℓ_2	ℓ_1	ℓ_2^2	ℓ_2	ℓ_1
10^{-3}	79	7	11	0.1257	0.0075	0.0107
10^{-2}	79	7	11	0.8242	0.0747	0.1055
10^{-1}	79	8	14	2.1958	0.7021	0.8863
10^{0}	79	4	9	3.6826	2.8876	2.8732
10^{1}	79	3	4	5.6317	5.3689	5.4361
10^{2}	79	2	2	10.9041	12.5914	13.0273
10^{3}	79	1	2	15.3304	16.2266	16.0463

Consider now the differences in results between the ℓ_2 and the ℓ_1 regularizers. On one hand, one can observe in the plots of the control signals obtained for the ℓ_1 regularizer in task 3, that the components tend to change separately along time. On the other hand, for the ℓ_2 regularizer, analysed in task 2, the components of the optimal control signal tend to change jointly along time. As a means of explaining this behavior it is important, in the first place, to explicitly expand the norm of the ℓ_1 regularizer terms of the objective function of (4)

$$\sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_1 = \sum_{t=1}^{T-1} |u_1(t) - u_1(t-1)| + \sum_{t=1}^{T-1} |u_2(t) - u_2(t-1)| \ ,$$

where $u_1(t)$ and $u_2(t)$ are the first and second components of $u(t)$, respectively. This shows that minimizing the changes in the control signal for the ℓ_1 norm is, in fact, equivalent to minimizing the changes in each of the components of u independently. As a result, in order to minimize the ℓ_1 norm, the control signal changes happen only in the direction of one of the components or, by chance, of both at the same time instant. As a consequence the control signal of the solution for the ℓ_1 norm is more prone to change separately in its components, given that the regularization can be regarded as independent for each of the components of the control signal vector. In fact, this effect can be observed in Fig. 22 for $\lambda = 10^0$ where the optimal control signal is represented in the plane. In this figure, the control inputs are plotted and each two consecutive ones are connected by lines in order to show that the changes only occur in the directions of the axes. However, making use of the ℓ_2 norm, given that its derivative is $D_{\mathbf{x}}||\mathbf{x}||_2 = \mathbf{x}/||\mathbf{x}||_2$, a change in one of the components is coupled with the change in the other. As a matter of fact, a control signal change that presents the highest similarity with the unregularized control signal is a change of the vector towards the origin, leading to changes in both of the components. The optimal control signal is also represented in the plane in this case in Fig. 23. In fact, it can be observed that the optimal control inputs components change jointly.

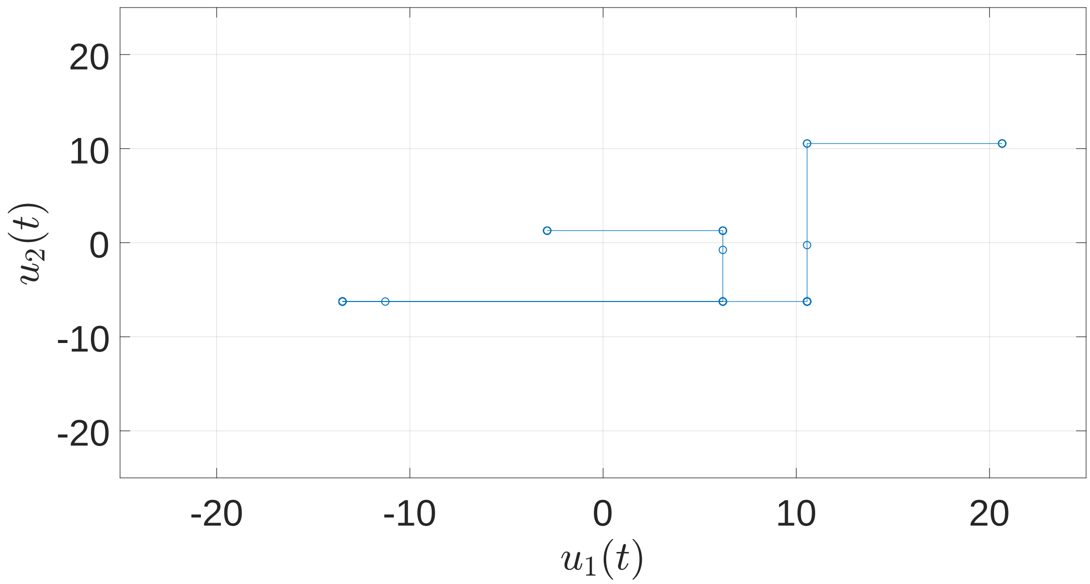


Figure 22: Representation of the optimal control signal for $\lambda = 10^0$ with the ℓ_1 regularizer in the plane.

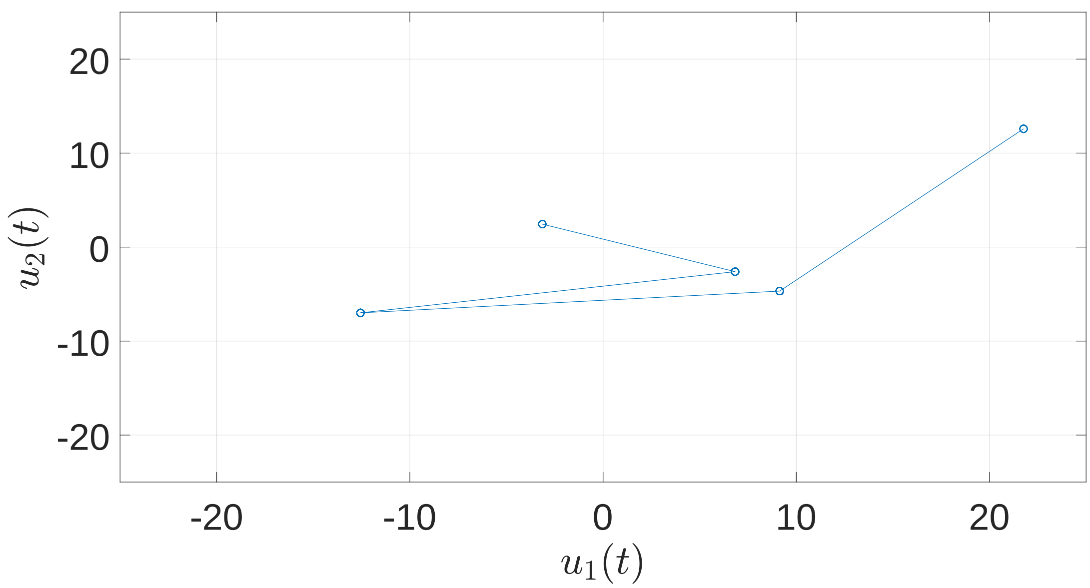


Figure 23: Representation of the optimal control signal for $\lambda = 10^0$ with the ℓ_2 regularizer in the plane.

The reason for the control signal using the ℓ_1 norm to change separately in its components and the control signal using the ℓ_2 norm to change jointly in its components has been established. Since the ℓ_2 regularizer tends to change both its components at the same time and the ℓ_1 tends to change one at a time, it is understandable why the ℓ_1 regularizer presents more changes in the optimal control signal than the ℓ_2 regularizer, as verified in tasks 2 and 3.

1.5 Task 5

In this task, one aims to find the smallest distance a moving target can be to a critical point at a specific point in time. The initial position p_0 and velocity v are unknown but the model of the movement of the target is know to be modeled by

$$p(t) = p_0 + tv. \tag{8}$$

The only information given about the position of the target are the disks where it was located at a given point in time. For the target to be in a disk in a given point in time, its distance to the center of the disk must be smaller or equal to the radius of the disk itself, *i.e.*,

$$||p_0 + t_k v - c_k|| \leq R_k \ , \tag{9}$$

for a disk k, at time t_k, whose center and radius are denoted as c_k and R_k, respectively. Given that it is impossible to determine the initial position p_0 and the velocity v, based on the disks information (unless the disks are colinear and of null radius) we would like to find the worst case possible, *i.e.*, the initial position and velocity such that, provided the disk information, allow for the target to be the closest possible to x^* at t^*. For the reasons above, the optimization problem can be formulated as

$$\begin{array}{ll} \underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & ||p_0 + t^* v - x^*|| \\ \text{subject to} & ||p_0 + t_k v - c_k|| \leq R_k, \quad k = 1, ..., K, \end{array}$$

where t_k is time at which the object was detected inside disk k, c_k is the center of disk k, and R_k is the radius of disk k, for $k = 1, ..., K$. After obtaining p_0 and v, one can use (8) to obtain p, the closest point possible to x^* at t^*, $p = p_0 + t^* v$ and the corresponding distance to x^*, $d = \|x^* - p\|_2$. Taking this into consideration, the following code was written to obtain the solutions $p_0 = [-0.5368 \quad -3.2715]^T$, $v = [1.2497 \quad 1.6803]^T$, $p = [9.4609 \quad 10.1709]^T$, and $d = 3.4651$ and Fig. 24.

```
% To ensure independent executions
clear
close all

% Example data
% tStar = 8;
```

```
% xStar = [6;10];
% t = [0,1,1.5,3,4.5];
% c = [[-1.721;-4.3454], [1.0550;-3.0293], [2.9619;-1.5857], ...
%      [3.8476;1.2253], [7.1086;4.9975]];
% R = [0.9993,1.4618,2.2617,1.0614,1.6983];

% Exercise data
tStar = 8;
xStar = [6;10];
t = [0,1,1.5,3,4.5];
c = [[0.6332;-3.2012], [-0.0054;-1.7104], [2.3322;-0.7620], ...
    [4.4526;3.1001], [6.1752;4.2391]];
R = [2.2727,0.7281,1.3851,1.8191,1.0895];

% Optimization problem
cvx_begin quiet
    variable p0(2,1)
    variable v(2,1)

    minimize(norm(p0+tStar*v-xStar,2))

    norms(p0*ones(1,length(t))+v*t-c) <= R
cvx_end

% Other solutions obtention
p = p0 + tStar*v;
d = norm(xStar-p);

% Draw graphic
N = 100;
th = 0:2*pi/N:2*pi;

figure();
for i=1:size(c,2)
    plot(c(1,i) + R(i)*cos(th),c(2,i) + R(i)*sin(th),'b')
    text(c(1,i),c(2,i),num2str(i))
    hold on
end
scatter(xStar(1), xStar(2),'k','filled')
text(xStar(1)-1,xStar(2)-1,'$x^*$','Interpreter','latex','FontSize',16)
scatter(p(1,:), p(2,:),'s','r')

for i=1:length(t)
    scatter(p0(1)+v(1)*t(i), p0(2)+v(2)*t(i),'s','r')
end

axis equal
set(gca,'FontSize',16)
xlabel('$x_1$','Interpreter','latex')
ylabel('$x_2$','Interpreter','latex')

```

```
% Save graphic as pdf
set(gcf,'Units','Inches');
pathFigPos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches',...
    'PaperSize',[pathFigPos(3), pathFigPos(4)])
print(gcf,'data/graphicClosestLocalization','-dpdf','-r0')
hold off

% Print solutions
p0
v
p
d

% Save solutions as matlab matrix file
mkdir('data')
save('data/closest_point_possible','p0','p','v','d')
```

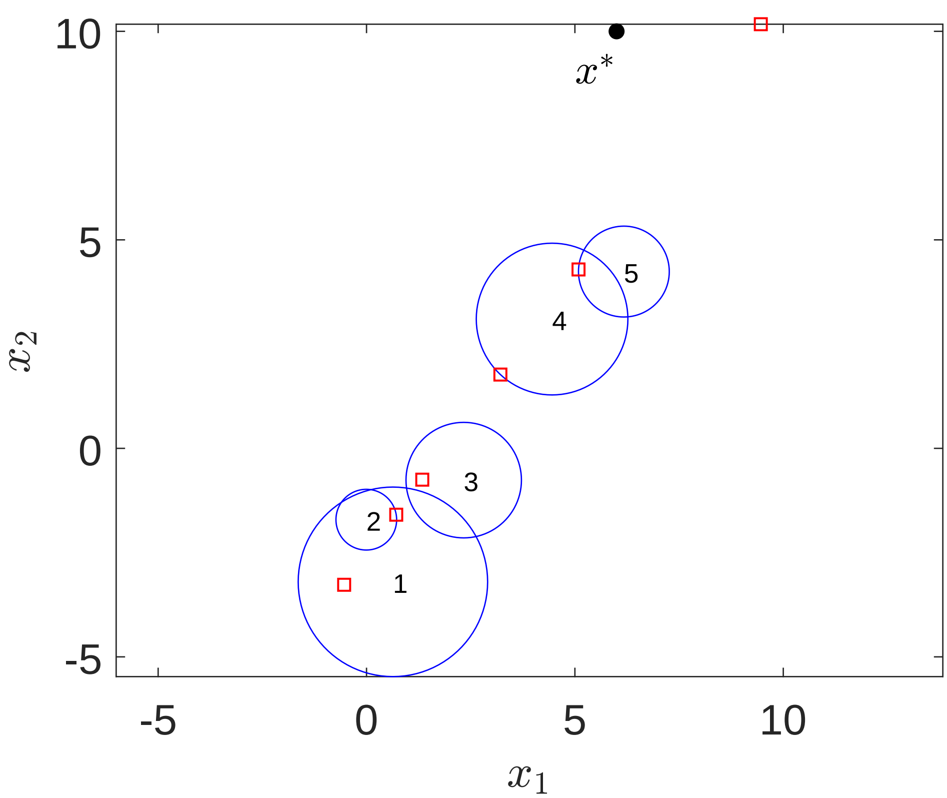


Figure 24: Closest possible location of the target at time t^*.

1.6 Task 6

In this task, the goal is to find the smallest enclosing rectangle of the position of the target at time t^* defined as

$$R\left(a_1, a_2, b_1 . b_2\right)=\left\{\left(x_1, x_2\right) \in \mathbf{R}^2: a_1 \leq x_1 \leq a_2, b_1 \leq x_2 \leq b_2\right\},$$

where $a_1 \in \mathbf{R}$, $a_2 \in \mathbf{R}$, $b_1 \in \mathbf{R}$, and $b_2 \in \mathbf{R}$. In the first place, in order to find the optimization problem that translates this problem, it is important to notice that this definition only considers rectangles aligned with the axes and non-empty sets can only be obtained if $a_1 \leq a_2$ and $b_1 \leq b_2$. In the second place, disk information is also provided, which corresponds to a constraint (9), for each disk $1, ..., K$. In addition, the problem must also be constrained in a way such that the obtained rectangle contains all the possible positions for the target at time t^*. Considering again $p \in \mathbf{R}^2$ to be the position of the target at time t, the last consideration translates into $a_1 \leq [1\ 0]p \leq a_2$ and $b_1 \leq [1\ 0]p \leq b_2$, for all possible values of p. Finally, it is necessary to mathematically specify the meaning of "smallest". In this report, it was assumed that the most reasonable choice would be to consider the rectangle with the smallest area, $(a_2-a_1)(b_2-b_1)$. The previous considerations lead to the formulation

$$\begin{array}{ll}
\underset{(a_1,a_2,b_1,b_2,p_0,v)\in\mathbf{R}^4\times\mathbf{R}^2\times\mathbf{R}^2}{\operatorname{minimize}} & \left(a_2-a_1\right)\left(b_2-b_1\right) \\
\text{subject to} & a_1 \leq a_2 \\
& b_1 \leq b_2 \\
& \begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right) \leq a_2 \\
& \begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right) \geq a_1 \\
& \begin{bmatrix}0 & 1\end{bmatrix}\left(p_0+t^* v\right) \leq b_2 \\
& \begin{bmatrix}0 & 1\end{bmatrix}\left(p_0+t^* v\right) \geq b_1 \\
& \left\|p_0+t_k v-c_k\right\| \leq R_k, \quad k=1, \ldots, K .
\end{array} \tag{10}$$

Given $a_1 \leq a_2$ and $b_1 \leq b_2$, it is obvious that the solution of the problem occurs for a_2 minimum, a_1 maximum, b_2 minimum, and b_1 maximum. Given the constraints for a_1, it is obvious that its maximum occurs for $a_1=\min\left\{\begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right)\right\}$. For a_2, its minimum occurs for $a_2=\max\left\{\begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right)\right\}=\min\left\{-\begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right)\right\}$, where it was taken into account that a maximization problem can be turned into a minimization problem by multiplying the objective function by -1. The same reasoning goes for b_1 and b_2. Therefore, it is necessary to find the solutions of the four optimization problems,

$$\begin{array}{ll}
\underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\operatorname{minimize}} & \begin{bmatrix}1 & 0\end{bmatrix}\left(p_0+t^* v\right) \\
\text{subject to} & \left\|p_0+t_k v-c_k\right\| \leq R_k, \quad k=1, \ldots, K .
\end{array} \tag{11}$$

$$\begin{array}{ll}
\underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\operatorname{minimize}} & \begin{bmatrix}-1 & 0\end{bmatrix}\left(p_0+t^* v\right) \\
\text{subject to} & \left\|p_0+t_k v-c_k\right\| \leq R_k, \quad k=1, \ldots, K,
\end{array} \tag{12}$$

$$\begin{array}{ll}
\underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\operatorname{minimize}} & \begin{bmatrix}0 & 1\end{bmatrix}\left(p_0+t^* v\right) \\
\text{subject to} & \left\|p_0+t_k v-c_k\right\| \leq R_k, \quad k=1, \ldots, K,
\end{array} \tag{13}$$

$$
\begin{array}{ll}
\underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}0 & -1\end{bmatrix}(p_0+t^*v) \\
\text{subject to} & ||p_0+t_kv-c_k|| \leq R_k, \quad k=1,...,K.
\end{array} \tag{14}
$$

For each set of solutions, p_0 and v, for (11) up to (14), it can be obtained respectively $a_1 = \begin{bmatrix}1 & 0\end{bmatrix}(p_0+t^*v)$, $a_2 = \begin{bmatrix}1 & 0\end{bmatrix}(p_0+t^*v)$, $b_1 = \begin{bmatrix}0 & 1\end{bmatrix}(p_0+t^*v)$, and $b_2 = \begin{bmatrix}0 & 1\end{bmatrix}(p_0+t^*v)$. Taking this into consideration, the following code was written to obtain the solutions $a_1 = 9.4582$, $a_2 = 14.1902$, $b_1 = 7.4763$, and $b_2 = 12.9690$ and the Fig. 25 represents them.

```
% To ensure independent executions
clear
close all

% Exercise data
tStar = 8;
xStar = [6;10];
t = [0,1,1.5,3,4.5];
c = [[0.6332;-3.2012], [-0.0054;-1.7104], [2.3322;-0.7620], ...
    [4.4526;3.1001], [6.1752;4.2391]];
R = [2.2727,0.7281,1.3851,1.8191,1.0895];

% Optimization problem
E = [1 0;
    -1 0;
    0 1;
    0 -1];
rectangleCorners = zeros(4,1);
for j=1:4
    cvx_begin quiet
        variable p0(2,1)
        variable v(2,1)

        minimize(E(j,:)*(p0+tStar*v))

        norms(p0*ones(1,length(t))+v*t-c) <= R
    cvx_end

    rectangleCorners(j) = abs(E(j,:))*(p0+tStar*v);
end

% Draw graphic
N = 100;
th = 0:2*pi/N:2*pi;

figure();
for i=1:size(c,2)
    plot(c(1,i) + R(i)*cos(th),c(2,i) + R(i)*sin(th),'b')
    text(c(1,i),c(2,i),num2str(i))
    hold on
end
```

```
axis equal
set(gca,'FontSize',16)
xlabel('$x_1$','Interpreter','latex')
ylabel('$x_2$','Interpreter','latex')
rectangle('Position',[rectangleCorners(1) rectangleCorners(3)...
    rectangleCorners(2)-rectangleCorners(1) ...
    rectangleCorners(4)-rectangleCorners(3)],'FaceColor',...
    [225/255 225/255 225/255],'LineStyle','--')
hold off

% Save graphic as pdf
set(gcf,'Units','Inches');
pathFigPos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches',...
    'PaperSize',[pathFigPos(3), pathFigPos(4)])
print(gcf,'data/graphicSmallestEnclosingRectangle','-dpdf','-r0')

% Print solutions
rectangleCorners

% Save solutions as matlab matrix file
mkdir('data')
save('data/smallest_enclosing_rectangle','rectangleCorners')
```

1.7 Smallest enclosing region

1.7.1 Algorithm

In addition, one can also observe that this method can be applied to any rectangle and not only the ones oriented with the axes. Consider a rectangle rotated with an angle ϕ with respect to the positive x_1 axis. In a reference frame rotated of ϕ in relation to the reference frame that has been considered, this rectangle would be aligned with the axes. In conclusion, for every rectangle not aligned with the axes being considered there is always another reference frame rotated of some angle ϕ from the previous one in which the rectangle is aligned with the axis and the optimization problem (10) can be solved. Considering a generic vector whose coordinates in the first reference frame are $u_1 \in \mathbf{R}^2$, its coordinates, $u_2 \in \mathbf{R}^2$, in the second reference frame, which was obtained from a rotation of the first one, can be written as

$$u_2 = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} u_1.$$

Therefore, the optimization problems (11) up to (14) can be rewritten for a rectangle rotated with an angle ϕ in relation to the x_1 axis of the first reference frame as

$$\begin{array}{ll} \underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}\cos\phi & \sin\phi\end{bmatrix}(p_0 + t^* v s) \\ \text{subject to} & ||p_0 + t_k v - c_k|| \leq R_k, \quad k = 1, ..., K. \end{array}$$

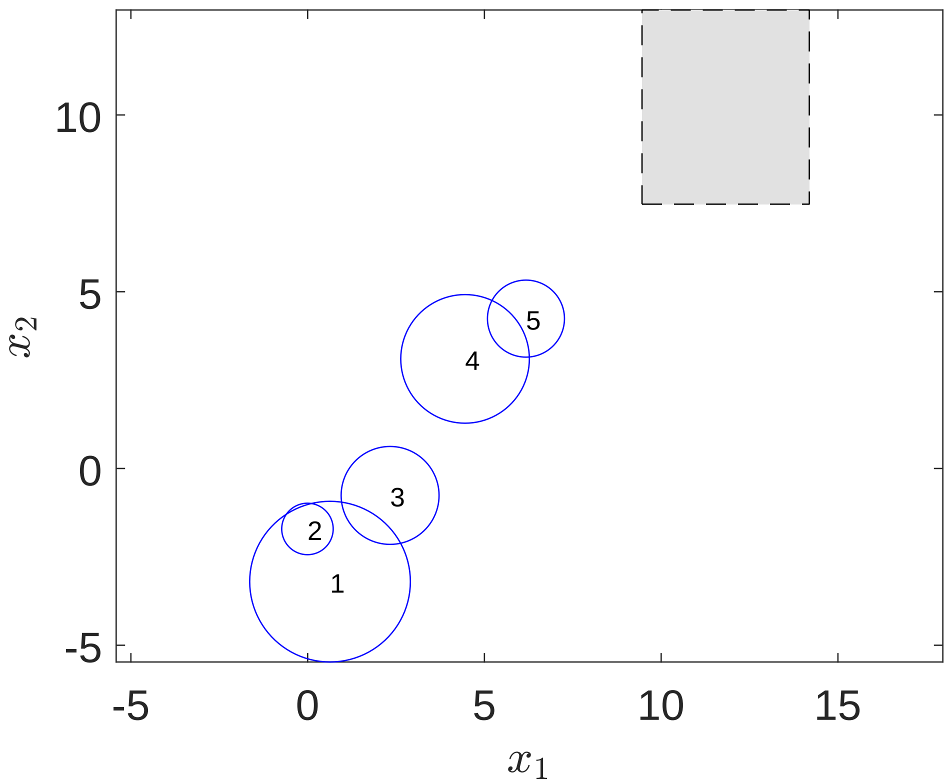


Figure 25: Smallest enclosing rectangle for the target position at time t^*.

$$\begin{array}{ll} \underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}-\cos\phi & -\sin\phi\end{bmatrix}(p_0+t^*v) \\ \text{subject to} & ||p_0+t_kv-c_k||\leq R_k, \quad k=1,...,K, \end{array}$$

$$\begin{array}{ll} \underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}-\sin\phi & \cos\phi\end{bmatrix}(p_0+t^*v) \\ \text{subject to} & ||p_0+t_kv-c_k||\leq R_k, \quad k=1,...,K, \end{array}$$

$$\begin{array}{ll} \underset{(p_0,v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}\sin\phi & -\cos\phi\end{bmatrix}(p_0+t^*v) \\ \text{subject to} & ||p_0+t_kv-c_k||\leq R_k, \quad k=1,...,K. \end{array}$$

However, these problems only give a_1, a_2, b_1, and b_2 in the second reference frame. In order to find the solution rectangle in the first one its vertices have to be computed and afterwards transformed back to the first reference frame. Since this procedure can be applied to every rotation angle, an approximation to the set of possible solutions for condition (9) can be obtained if various inclinations are chosen. One of those approximations can be observed in Fig. 26 for 10 equally spaced values of rotation angle in $[0, \pi/2[$. This approximation was obtained with the code below. It is important to notice that these approximations are always convex polygons. Therefore, not existing guarantees that when N tends to infinity, the approximation tends to the smallest enclosing region.

```
% smallest_enclosing_region.m
% To ensure independent executions
clear
close all

% Exercise data
tStar = 8;
xStar = [6;10];
t = [0,1,1.5,3,4.5];
c = [[0.6332;-3.2012], [-0.0054;-1.7104], [2.3322;-0.7620], ...
    [4.4526;3.1001], [6.1752;4.2391]];
R = [2.2727,0.7281,1.3851,1.8191,1.0895];

% Plots the data first
N = 100;
th = 0:2*pi/N:2*pi;
figure();
for i=1:size(c,2)
    plot(c(1,i) + R(i)*cos(th),c(2,i) + R(i)*sin(th),'b')
    text(c(1,i),c(2,i),num2str(i))
    hold on
end
axis equal
set(gca,'FontSize',16)
xlabel('$x_1$','Interpreter','latex')
ylabel('$x_2$','Interpreter','latex')

% Optimization problem
E = [1 0;
    -1 0;
    0 1;
    0 -1];
N = 10;
% a1,a2,b1,b2
rectangleSides = zeros(1,4);
% corners coordinates in clockwise rotation from top left with first
% coordinate being x1
rectangleCorners = zeros(2,5);

% Rotation angle between tries
phi = 0:pi/2/N:pi/2;

for i=1:N
    % Updates F
    F = [cos(phi(i)) sin(phi(i));
         -sin(phi(i)) cos(phi(i))];

    for j=1:4
        cvx_begin quiet
            variable p0(2,1)
            variable v(2,1)
```

```
            minimize(E(j,:)*F*(p0+tStar*v))

            norms(p0*ones(1,length(t))+v*t-c) <= R
        cvx_end

        % In fact, this computes sides
        rectangleSides(j) = abs(E(j,:))*F*(p0+tStar*v);
    end

    % Computes corners from sides
    rectangleCorners(1,1) = rectangleSides(1);
    rectangleCorners(2,1) = rectangleSides(4);

    rectangleCorners(1,2) = rectangleSides(2);
    rectangleCorners(2,2) = rectangleSides(4);

    rectangleCorners(1,3) = rectangleSides(2);
    rectangleCorners(2,3) = rectangleSides(3);

    rectangleCorners(1,4) = rectangleSides(1);
    rectangleCorners(2,4) = rectangleSides(3);

    % To close the rectangle
    rectangleCorners(1,end) = rectangleCorners(1,1);
    rectangleCorners(2,end) = rectangleCorners(2,1);

    % In the actual reference frame, it is rotated by F
    rectangleCorners = F'*rectangleCorners;

    % Draws the rectangle
    plot(rectangleCorners(1,:),rectangleCorners(2,:),'k')
end
hold off

% Save graphic as pdf
set(gcf,'Units','Inches');
pathFigPos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches',...
    'PaperSize',[pathFigPos(3), pathFigPos(4)])
print(gcf,'data/graphicSmallestEnclosingPolygon','-dpdf','-r0')
```

1.7.2 Convergence guarantees

The problem of finding the smallest enclosing region $\mathcal{D} \in \mathbb{R}^2$ of the possible positions at time t^* can be written as

$$\begin{aligned} &\underset{\mathcal{D}\in\mathbf{R}^2}{\text{minimize}} && \text{Area}(\mathcal{D}) \\ &\text{subject to} && (\mathbf{p_0} + t^*\mathbf{v}) \in \mathcal{D},\ \forall\ \mathbf{p}_0, \mathbf{v} \in \mathbb{R}^2 : \|\mathbf{p_0} + t_k\mathbf{v} - \mathbf{c_k}\| \leq R_k, \quad k = 1, ..., K\ . \end{aligned} \tag{15}$$

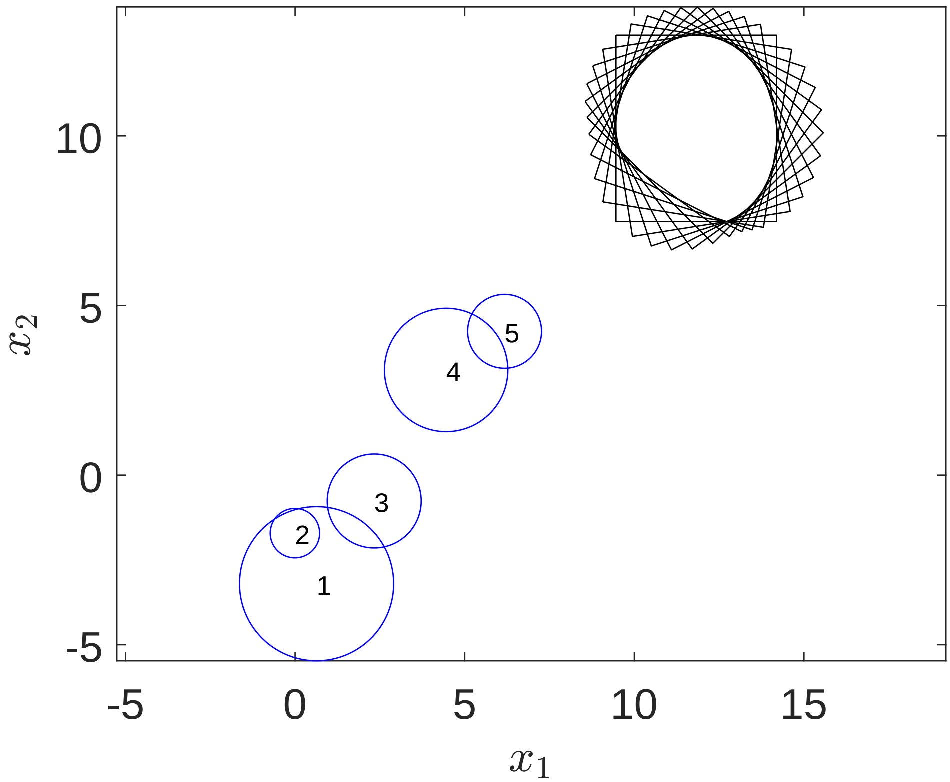


Figure 26: Smallest enclosing region for $N = 10$ equally spaced rotations.

which is clearly nonconvex. Previously the solution to this problem has been approximated with the solution to $4N$ convex problems of the form

$$
\begin{array}{ll}
\underset{(\mathbf{p_0}, v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}\cos\phi & \sin\phi\end{bmatrix}(\mathbf{p_0} + t^*\mathbf{v}) \\
\text{subject to} & ||\mathbf{p_0} + t_k\mathbf{v} - \mathbf{c_k}|| \le R_k, \quad k = 1, ..., K. \\
 & \\
\underset{(\mathbf{p_0}, \mathbf{v})\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}-\cos\phi & -\sin\phi\end{bmatrix}(\mathbf{p_0} + t^*\mathbf{v}) \\
\text{subject to} & ||\mathbf{p_0} + t_k\mathbf{v} - \mathbf{c_k}|| \le R_k, \quad k = 1, ..., K, \\
 & \\
\underset{(\mathbf{p_0}, v)\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}-\sin\phi & \cos\phi\end{bmatrix}(\mathbf{p_0} + t^*\mathbf{v}) \\
\text{subject to} & ||\mathbf{p_0} + t_k\mathbf{v} - \mathbf{c_k}|| \le R_k, \quad k = 1, ..., K, \\
 & \\
\underset{(\mathbf{p_0}, \mathbf{v})\in\mathbf{R}^2\times\mathbf{R}^2}{\text{minimize}} & \begin{bmatrix}\sin\phi & -\cos\phi\end{bmatrix}(\mathbf{p_0} + t^*\mathbf{v}) \\
\text{subject to} & ||\mathbf{p_0} + t_k\mathbf{v} - \mathbf{c_k}|| \le R_k, \quad k = 1, ..., K.
\end{array}
\tag{16}
$$

This technique is called convex relaxation. It is possible to prove, in this case, that it does, in fact, converge to the optimal nonconvex solution, as detailed in the following result.

Theorem 1.1. *Consider N rectangles, $\mathcal{R}_1, \ldots, \mathcal{R}_N$, each obtained by solving 4 convex optimization problems* (16)*, for $\phi \in \Phi$ with*

$$\Phi = \left\{ \phi \in \mathbb{R} : \phi = \frac{\pi}{2}((n-1)/N) \,; n = 1, \ldots, N \right\} .$$

Then the intersection of the N rectangles, converges to the solution of the nonconvex problem (15) *as $N \to \infty$, i.e.*

$$\lim_{N\to\infty} \cap_{i=1}^{N} \mathcal{R}_i = \mathcal{D} \,,$$

where $\mathcal{D}$ is the solution to (15)*.*

Proof. First, consider the set $\mathcal{S} \in \mathbb{R}^4$ defined by

$$\mathcal{S} := \left\{ \mathbf{s} = \mathrm{col}(\mathbf{p_0}, \mathbf{v}) \in \mathbb{R}^4 : \|\mathbf{p_0} + t_k \mathbf{v} - \mathbf{c_k}\| \leq R_k, \quad k = 1, ..., K \right\} ,$$

or equivalently

$$\mathcal{S} := \left\{ \mathbf{s} \in \mathbb{R}^4 : \left\| \mathbf{h}_k^T s - \mathbf{c_k} \right\| \leq R_k, \quad k = 1, ..., K \right\} ,$$

with

$$\mathbf{h}_k = \begin{bmatrix} \mathbf{I}_{2\times2} \\ t_k \mathbf{I}_{2\times2} \end{bmatrix} .$$

For $\mathbf{s}_1 \in \mathcal{S}$ and $\mathbf{s}_2 \in \mathcal{S}$ consider $\mathbf{s}(\gamma) = \mathbf{s}_1(1-\gamma) + \mathbf{s}_2\gamma$. Note that $\mathbf{s}(\gamma)$ corresponds to a one-dimensional parameterization that connects $\mathbf{s}_1 \in \mathbb{R}^4$ and $\mathbf{s}_2 \in \mathbb{R}^4$, if $\gamma \in [0, 1]$. It is possible to write

$$\begin{aligned}
&\left\| \mathbf{h}_k^T \mathbf{s}(\gamma) - \mathbf{c}_k \right\| \\
=&\left\| \mathbf{h}_k^T (\mathbf{s}_1(1-\gamma) + \mathbf{s}_2\gamma) - \mathbf{c}_k \right\| \\
=&\left\| (1-\gamma)\mathbf{h}_k^T \mathbf{s}_1 + \gamma \mathbf{h}_k^T \mathbf{s}_2 - (1-\gamma)\mathbf{c}_k - \gamma \mathbf{c}_k \right\| \\
=&\left\| (1-\gamma)(\mathbf{h}_k^T \mathbf{s}_1 - \mathbf{c}_k) + \gamma(\mathbf{h}_k^T \mathbf{s}_2 - \mathbf{c}_k) \right\| \\
\leq&(1-\gamma) \left\| \mathbf{h}_k^T \mathbf{s}_1 - \mathbf{c}_k \right\| + \gamma \left\| \mathbf{h}_k^T \mathbf{s}_2 - \mathbf{c}_k \right\| \\
\leq&(1-\gamma)R_k + \gamma R_k \\
=&R_k \,,
\end{aligned}$$

where the triangular inequality was used. Therefore, $\mathbf{s}(\gamma) \in \mathcal{S}$. Then, by the definition of a convex set, if $\mathbf{s}_1 \in \mathcal{S}$ and $\mathbf{s}_2 \in \mathcal{S} \implies \mathbf{s}(\gamma) = \mathbf{s}_1(1-\gamma) + \mathbf{s}_2\gamma \in \mathcal{S}$ for $\gamma \in [0, 1]$, then $\mathcal{S}$ is convex. Furthermore, consider

$$\mathcal{D} := \left\{ \mathbf{x} \in \mathbb{R}^2 : \mathbf{x} = \mathbf{h}_*^T \mathbf{s}, \mathbf{s} \in \mathcal{S} \right\} ,$$

which one can easily say is the solution to (15). Note that $\mathcal{D}$ is the result of a linear transformation of $\mathcal{S}$, which is convex, thus $\mathcal{D}$ is convex. Furthermore, note that the solution of (16) for ϕ_i is the smallest enclosing rectangle of $\mathcal{D}$, with orientation ϕ_i, which is known as an oriented minimum bounding box. In fact, it it easily shown that

$$\lim_{N\to\infty} \cap_{i=1}^{N} \mathcal{R}_i = \mathrm{C}\left(\mathcal{D}\right) ,$$

where $\mathrm{C}(\mathcal{D})$ denotes the convex hull of $\mathcal{D}$. As it was previously shown, $\mathcal{D}$ is convex, therefore $\mathrm{C}(\mathcal{D}) = \mathcal{D}$. It then follows

$$\lim_{N\to\infty} \cap_{i=1}^{N} \mathcal{R}_i = \mathcal{D} .$$

□

2 Part 2

2.1 Task 1

In this Part, the goal is to solve the optimization problem

$$\underset{(s,r)\in\mathbf{R}^n\times\mathbf{R}}{\text{minimize}} \quad \tfrac{1}{K}\textstyle\sum_{k=1}^{K}\left(\log\left(1+\exp\left(s^T x_k - r\right)\right) - y_k\left(s^T x_k - r\right)\right). \tag{17}$$

From (17), the objective function is $f : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$

$$f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\left(\log\left(1+\exp\left(s^T x_k - r\right)\right) - y_k\left(s^T x_k - r\right)\right), \tag{18}$$

which can be written as

$$f(s,r) = \sum_{k=1}^{K}\frac{1}{K}h_k(s,r) + \sum_{k=1}^{K}\frac{1}{K}l_k(s,r), \tag{19}$$

where, for $k = 1, ..., K$, $h_k : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$

$$h_k(s,r) = \log\left(1+\exp\left(s^T x_k - r\right)\right), \tag{20}$$

and $l_k : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$

$$l_k(s,r) = -y_k\left(s^T x_k - r\right). \tag{21}$$

From (21), the functions l_k can be written as

$$l_k(s,r) = \begin{bmatrix} -y_k x_k \\ y_k \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix} + 0, \tag{22}$$

Therefore, the functions l_k are affine. Since l_k are affine, they are also convex. In addition, from (20), the functions h_k can be written as

$$h_k(s,r) = (t_k \circ q_k)(s,r), \tag{23}$$

where, for $k = 1, ..., K$, $t_k : \mathbf{R} \to \mathbf{R}$

$$t_k(z) = \log(1+\exp(z)) \tag{24}$$

and $q_k : \mathbf{R}^n \times \mathbf{R} \to \mathbf{R}$

$$q_k(s,r) = s^T x_k - r. \tag{25}$$

From (24), it can be observed that the functions t_k are logistic functions and, therefore, convex. The functions q_k, on the other hand, can be written from (25) as

$$q_k(s,r) = \begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix} + 0,$$

being, therefore, affine. Since the functions q_k are affine and t_k are convex, it is know from (23) that each function h_k can be written as the composition of an affine function with a convex function, which means the functions h_k are all convex. Since all functions h_k and l_k are convex, it is also known from (18) that the objective function f is the sum with positive coefficients of convex functions, which means f is convex.

2.2 Task 2

In this Task, the gradient descent algorithm will be used to solve (17). To use this algorithm, it is necessary to know the gradient of f. From (19), one can write

$$\nabla f(s,r) = \frac{1}{K} \sum_{k=1}^{K} (\nabla h_k(s,r) + \nabla l_k(s,r)). \tag{26}$$

From (21), the gradient of l_k can be written as

$$\nabla l_k(s,r) = \begin{bmatrix} -y_k x_k \\ y_k \end{bmatrix} = -y_k \begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{27}$$

From (23), it can be written that

$$\nabla h_k(s,r) = \nabla t_k(q_k(s,r)) \nabla q_k(s,r), \tag{28}$$

where, from (25),

$$\nabla q_k(s,r) = \begin{bmatrix} x_k \\ -1 \end{bmatrix} \tag{29}$$

and, from (24),

$$\nabla t_k(z) = \frac{dt_k}{dz}(z) = \frac{\exp(z)}{1+\exp(z)}. \tag{30}$$

Combining (25), (28), (29), and (30) leads to

$$\nabla h_k(s,r) = \left(1 - \frac{1}{\exp\left(\begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix} \right) + 1} \right) \begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{31}$$

Finally, combining (26), (27), and (31), one can write

$$\nabla f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\left(1 - y_k - \frac{1}{\exp\left(\begin{bmatrix} x_k \\ -1 \end{bmatrix}^T \begin{bmatrix} s \\ r \end{bmatrix}\right) + 1}\right)\begin{bmatrix} x_k \\ -1 \end{bmatrix}. \tag{32}$$

In order to apply the gradient method, the following MATLAB script was written. This script returns the results of the method for each of the datasets given. This script implements the Gradient Descent algorithm through the MATLAB function `gradientDescent` also given below. For Task 2, the results obtained were $s = (1.3495, 1.0540)$ and $r = 4.8815$. The dataset 1 and the line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$ are represented in Fig. 27. In Fig. 28, the norm of the gradient along iterations is represented.

```
% GradientMethod.m
%% Initialization
clear;
clc;
NDataSets = 4;

%% Setup parameters
epsl = 1e-6; % stopping criterion
alpha_hat = 1; %initialization of alpha_k for the backtracking routine
gamma = 1e-4; % gamma of backtraking routine
beta = 0.5; % beta of backtraking routine
maxIt = [1e4; 1e4; 1e4; 1e5]; % maximum number of iterations

%% GD for each data set
for i = 1:NDataSets
    %% Upload data
    load(sprintf("./data%d.mat",i),'X','Y'); % upload data set
    K = length(Y);
    n = size(X,1);

    %% Set up x0 (note that x = [s;r])
    x0 = [-ones(n,1); 0];

    %% Setup objetive function and gradient
    h = [X;-ones(1,K)];
    F = @(x) (1/K)*...
        sum(log(1+exp((h'*x)'))-Y.*(h'*x)');
    gradF = @(x) (1/K)*sum((exp((h'*x)')./...
        (1+exp((h'*x)'))-Y).*h,2);

    %% Run GD
    fprintf("Running gradient descent for dataset %d (n = %d | K = %d).\n",...
        i,n,K);
    tic
```

```
    [xGD,ItGD,normGradGD] = gradientDescent(F,gradF,x0,epsl,...
        alpha_hat,gamma,beta,maxIt(i));
    elapsedTimeGD = toc;
    if ¬isnan(xGD)
        fprintf("Gradient descent for dataset %d"+...
        " converged in %d iterations.\n",i,ItGD);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeGD);
        if i≤2
            fprintf("s = [%g; %g] | r = %g.\n",xGD(1),xGD(2),xGD(3));
        end
    else
        fprintf("Gradient descent for dataset %d "+...
            "exceeded the maximum number of iterations.\n",i);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeGD);
    end
    save(sprintf("./DATA/GradientDescent/GDsolDataset%d.mat",i),...
        'xGD','ItGD','normGradGD','elapsedTimeGD');

    %% Plot result
    plotResults = false;
    if plotResults
    if i≤ 2
        figure('units','normalized','outerposition',[0 0 1 1]);
        set(gca,'FontSize',35);
        hold on;
        ax = gca;
        ax.XGrid = 'on';
        ax.YGrid = 'on';
        axis equal;
        for k = 1:K
            if Y(k)
                scatter(X(1,k),X(2,k),200,'o','b','LineWidth',3);
            else
                scatter(X(1,k),X(2,k),200,'o','r','LineWidth',3);
            end
        end
        %ylim([-4 8]);
        %xlim([-4 8]);
        title(sprintf("Dataset %d",i));
        ylabel('$x_2$','Interpreter','latex');
        xlabel('$x_1$','Interpreter','latex');
        x1 = (min(X(1,:)):(max(X(1,:)-min(X(1,:))))/100:max(X(1,:)));
        plot(x1,(xGD(3)-xGD(1)*x1)/xGD(2),'--g','LineWidth',4);
        saveas(gcf,sprintf("./DATA/GradientDescent/GDsolDataset%d.fig",i));
        close(gcf);
        hold off;
    end

    figure('units','normalized','outerposition',[0 0 1 1]);
    plot(0:ItGD,normGradGD,'LineWidth',3);
    hold on;
```

```
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title(sprintf("Gradient method | Dataset %d",i));
    ylabel('$||\Delta f (s_k,r_k)||$','Interpreter','latex');
    xlabel('$k$','Interpreter','latex');
    set(gca, 'YScale', 'log');
    saveas(gcf,sprintf("./DATA/GradientDescent/GDNormGradDataset%d.fig",i));
    close(gcf);
    hold off;
    end

end
```

```
% gradientDescent.m
function [xk,k,normGk] = gradientDescent(F,gradF,x0,epsl,...
    alpha_hat,gamma,beta,maxIt)
    %% Description
    % Inputs: 1. F: objective function (as a function handle)
    %         2. gradF: gradient of teh objective function (as a function
    %         handle)
    %         3. x0: initialization
    %         4. epsl: stopping criterion
    %         5. alpha_hat: initialization of alpha_k for the backtracking
    %         routine
    %         6. gamma: gamma of backtraking routine
    %         7. beta: beta of backtraking routine
    %         8. maxIt: maximum number of iterations
    % Outputs: 1. x: output of the gradient descent method (returns NaN if
    %          stopping criterion not met after the maximum number of
    %          iterations chosen
    %          2. k: number of iterations required for convergence if a
    %          solution was found
    %          3. normGk: norm of the gradient of the objective function
    %% Gradient descent routine
    k = 0;
    xk = x0;
    normGk = zeros(maxIt,1);
    while k < maxIt
        gk = gradF(xk); % Compute gradient at xk
        normGk(k+1) = norm(gk);
        if normGk(k+1) < epsl % Stopping criterion
           break;
        end
        % ---------  backtracking routine  ---------
        alpha_k = alpha_hat;
        % It is guaranteed that there is convergence, no maximum number of
        % iterations needed (obviously for beta < 0)
        while true
```

```
            % check if F(alpha_k) < phi(0)+gamma*phi_dot(0)+alpha_k
            if F(xk-alpha_k*gk) < F(xk)-gamma*alpha_k*(gk'*gk)
                break; % alpha_k found
            else
                alpha_k = beta*alpha_k; % Update alpha_k
            end
        end
        xk = xk - alpha_k*gk; % update xk
        % -------  End backtracking routine  -------
        k = k + 1; % Increment iteration count
    end
    if k == maxIt
        % No solution found within the maximum number of iterations
        xk = NaN;
    else
        normGk = normGk(1:k+1);
    end
end
```

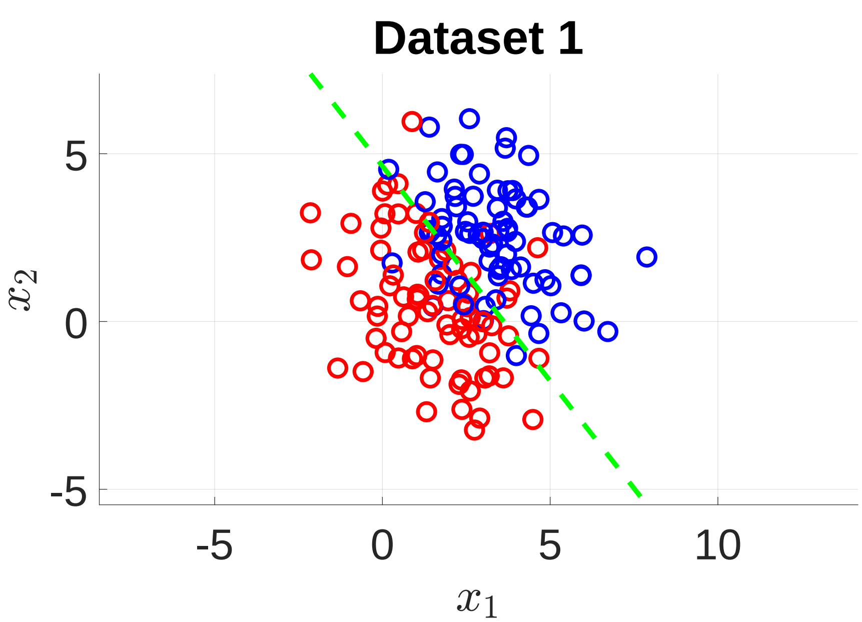


Figure 27: Dataset 1 and the corresponding line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$.

2.3 Task 3

In this Task, the code used was the one presented in the previous section. The results can be observed in Fig. 29 and 30 and the values obtained for s and r were $s = (0.7402, 2.3577)$ and $r = 4.5553$.

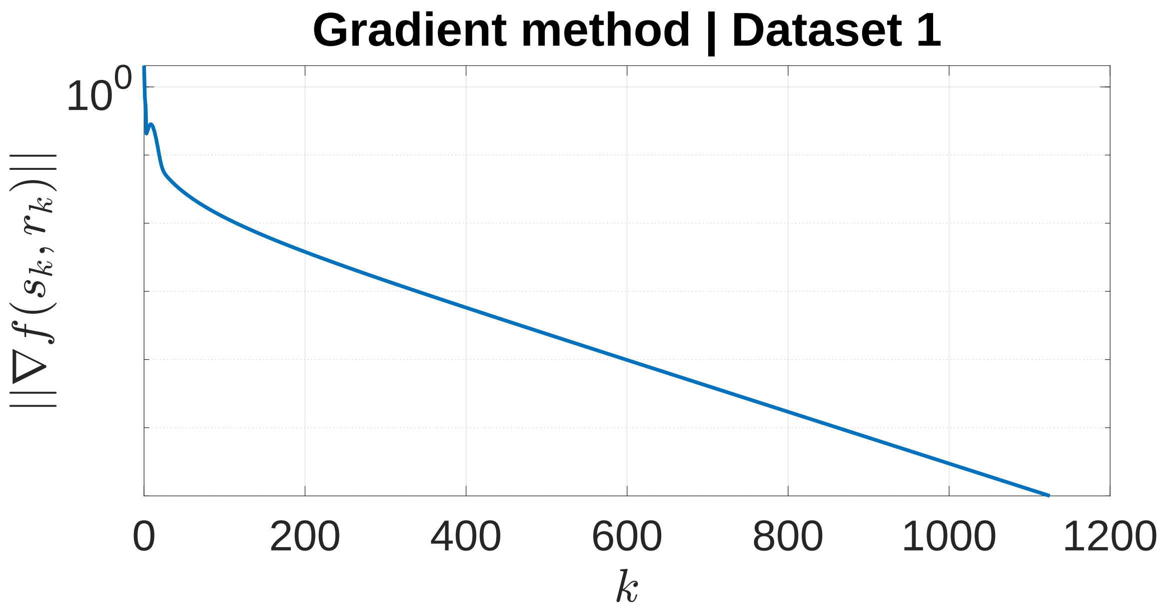


Figure 28: Norm of the gradient along iterations for the dataset 1.

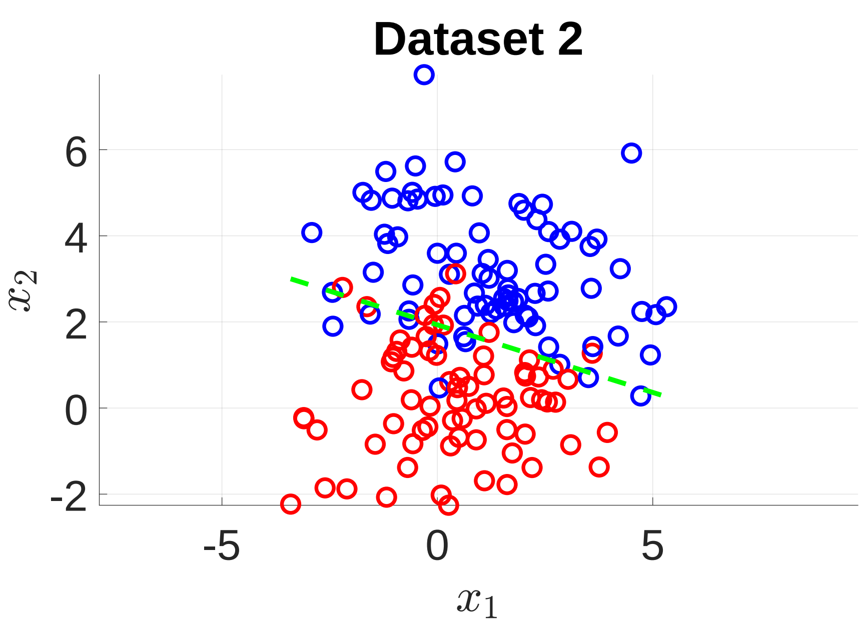


Figure 29: Dataset 2 and the corresponding line defined by $\{x \in \mathbf{R}^2 : s^T x = r\}$.

2.4 Task 4

In this Task, the gradient method was applied to two different datasets. However, in these datasets the points are no longer two-dimensional. In the dataset 3, $x_k \in \mathbf{R}^{30}$, for $k = 1, ..., 500$, and in the dataset 4, $x_k \in \mathbf{R}^{100}$, for $k = 1, ..., 8000$, which means it is no longer

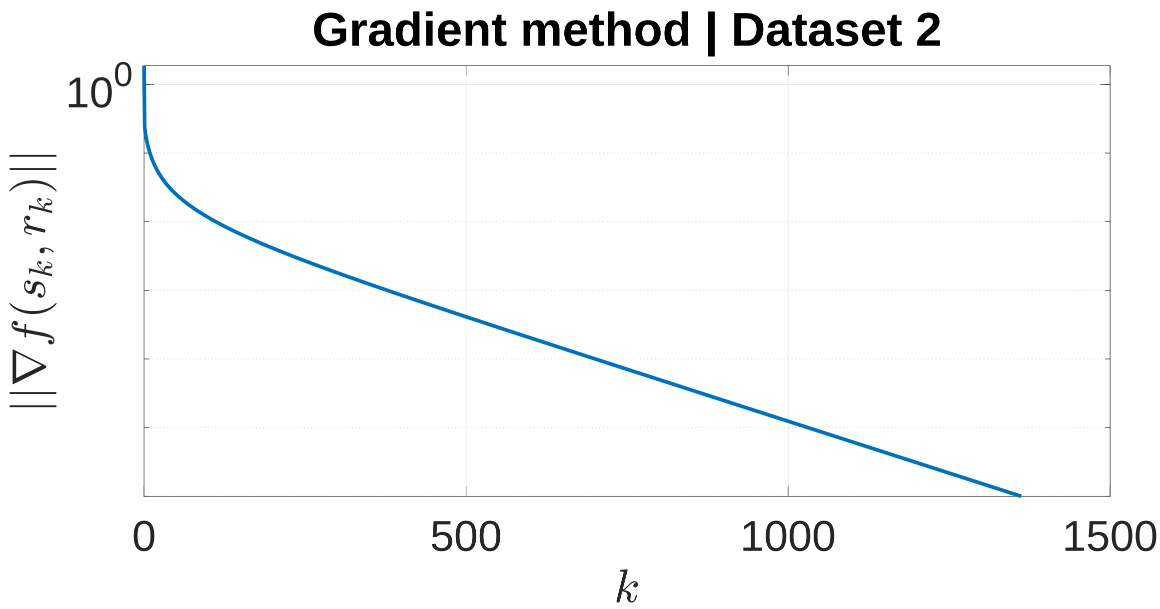


Figure 30: Norm of the gradient along iterations for the dataset 2.

possible to represent the datasets. Therefore, only the global minimizers, s and r, and the evolution of the norm of the gradient along iterations are presented. For dataset 3,

$$s = [-1.3082, 1.4078, 0.8049, -1.0024, 0.5548, -0.5489, -1.1997, 0.0792, -1.8279, -0.1484, 1.9241, -0.3586, -0.2900, 0.1925, 1.0614, 0.2107, -0.0929, 1.0476, -1.1248, -1.3311, 0.7661, -0.2729, -0.5349, 0.9996, -0.4191, -0.3133, 0.4075, -0.1965, -0.7379, -0.9814],$$

$r = 4.7984$, and the evolution of the norm of the gradient along iterations is represented in Fig. 31. For dataset 4,

$$s = [0.1098, -0.6423, 0.1019, 1.2428, -1.6431, 1.0244, 0.0512, 0.8271, 0.3136, 0.7449, -0.5858, 0.6267, 1.3611, 0.1534, 2.3234, -0.0840, -0.9489, 2.4699, -0.8678, -1.6516, 0.6460, -0.4779, 1.6397, 0.9034, -1.2293, -0.7587, -0.4887, 1.0306, 0.0888, -1.0917, -1.2717, -2.0333, -0.2505, -0.3518, -0.3486, -2.5610, -0.3132, -0.4902, 0.7258, 0.5774, -1.0528, 0.6400, 0.3759, -0.1547, 0.0298, 0.9547, -0.2863, 0.6364, 0.7859, 0.7584, 0.2880, 0.1648, 0.6776, 2.0550, 1.0996, 0.5261, -0.5770, 1.1454, -0.5617, 0.0065, 0.4768, -2.3677, -1.1561, -2.6619, 0.0622, 0.1037, -0.6237, 0.1913, 0.6672, -1.0493, -0.3240, -0.3207, -1.0904, -0.8293, -0.3104, -0.4879, -0.1060, -0.1646, 2.2683, -1.2380, -0.8575, -2.4781, -0.4158, 0.1660, 0.7931, 0.3685, -0.0524, -0.9997, -0.5732, 0.3971, 1.1911, 1.8318, -1.7287, 0.2329, -1.1921, 1.6558, 0.4612, -0.6431, 0.8295, 0.2975],$$

$r = 7.6701$, and the evolution of the norm of the gradient along iterations is represented in Fig. 32.

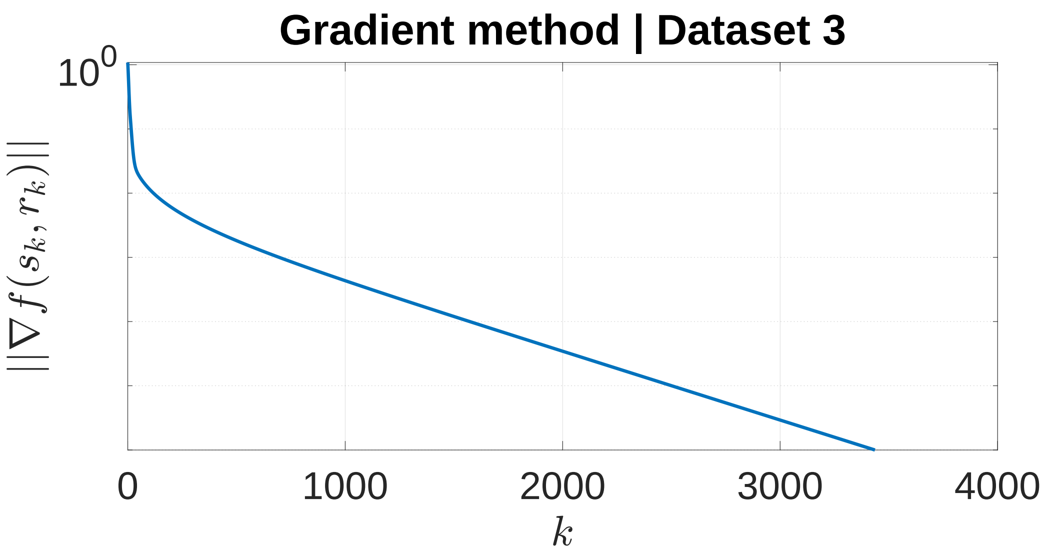


Figure 31: Norm of the gradient along iterations for the dataset 3.

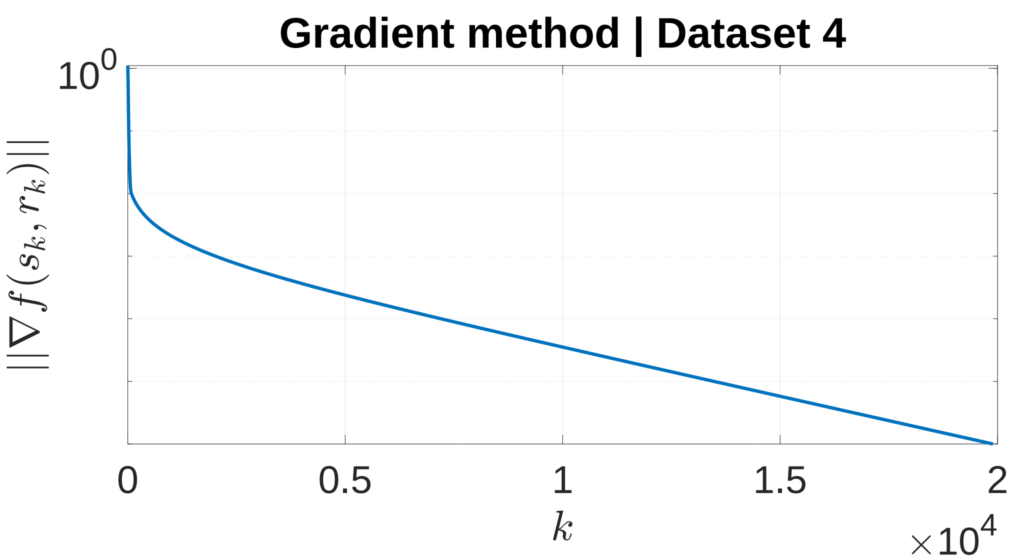


Figure 32: Norm of the gradient along iterations for the dataset 4.

2.5 Task 5

Letting $\phi : \mathbf{R} \rightarrow \mathbf{R}$ be a twice differentiable function and supposing $p : \mathbf{R}^3 \rightarrow \mathbf{R}$ is given by

$$p(x) = \sum_{k=1}^{K} \phi\left(a_k^T x\right), \tag{33}$$

where $a_k \in \mathbf{R}^3$ for $k = 1, ..., K$, one can write from the rule of the derivative of a composed function

$$\nabla p(x) = \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_k = Av, \tag{34}$$

where $A = [a_1 \quad a_2 \quad ... \quad a_K]$ and $v = [\phi(a_1^T x) \quad \phi(a_2^T x) \quad ... \quad \phi(a_K^T x)]^T$. In order to write the Hessian of p at x, one can define, for $k = 1, ..., K$, the functions $u_k : \mathbf{R} \rightarrow \mathbf{R}^3$

$$u_k(z) = z a_k, \tag{35}$$

which, from (34), lead to

$$\nabla p(x) = \sum_{k=1}^{K} u_k \left[\dot{\phi}(a_k^T x)\right]. \tag{36}$$

From (35) and (36), it is possible to write

$$\nabla^2 p(x) = \sum_{k=1}^{K} D u_k \left[\dot{\phi}(a_k^T x)\right] D \left[\dot{\phi}(a_k^T x)\right] D(a_k^T x) = \sum_{k=1}^{K} a_k \ddot{\phi}(a_k^T x) a_k^T = ADA^T, \tag{37}$$

where D is the diagonal matrix

$$D = \begin{bmatrix} \ddot{\phi}(a_1^T x) & & & \\ & \ddot{\phi}(a_2^T x) & & \\ & & ... & \\ & & & \ddot{\phi}(a_K^T x) \end{bmatrix} \tag{38}$$

2.6 Task 6

In this Task, the Newton method will be used to solve (17). To be able to use this method, it is necessary to know the gradient, which is given by (32), and the Hessian of the objective function. In order to write the Hessian, one starts by writing, from (18), (20), and (21),

$$f(x) = \sum_{k=1}^{K} \left(\phi\left(a_k^T x\right) + \frac{1}{K} \begin{bmatrix} -y_k x_k \\ y_k \end{bmatrix}^T x \right), \tag{39}$$

where $x = [s \quad r]^T$, $a_k = [x_k \quad -1]^T$ and $\phi : \mathbf{R} \rightarrow \mathbf{R}$

$$\phi(z) = \frac{1}{K} \log(1 + \exp(z)) \tag{40}$$

with second-derivative

$$\ddot{\phi}(z) = \frac{\exp(z)}{K\left[1+\exp(z)\right]^2}. \tag{41}$$

Taking into consideration that (39) is written in the form of (33) except for the sum of affine terms whose second-derivative is null, it may be written, from (37), (38), and (41),

$$\nabla^2 f(x) = \begin{bmatrix} a_1 & a_2 & \dots & a_K \end{bmatrix} \begin{bmatrix} \ddot{\phi}(a_1^T x) & & & \\ & \ddot{\phi}(a_2^T x) & & \\ & & \dots & \\ & & & \ddot{\phi}(a_K^T x) \end{bmatrix} \begin{bmatrix} a_1^T \\ a_2^T \\ \dots \\ a_K^T \end{bmatrix}. \tag{42}$$

Taking into consideration (32) and (42), the Newton method was implemented according to the script below. In it, the Newton algorithm is implemented through the MATLAB function `newtonAlgorithm` also presented below. In addition in Fig. 33, the evolutions of the norm of the gradient are presented for each dataset and the evolutions of the values of the stepsizes are presented for each dataset in Fig. 34.

```
%% Initialization
clear;
clc;
NDataSets = 4;

%% Setup parameters
epsl = 1e-6; % stopping criterion
alpha_hat = 1; %initialization of alpha_k for the backtracking routine
gamma = 1e-4; % gamma of backtraking routine
beta = 0.5; % beta of backtraking routine
maxIt = [15; 15; 15; 15]; % maximum number of iterations

%% Newton algorithm for each data set
for i = 1:NDataSets
    % Upload data
    load(sprintf("./data%d.mat",i),'X','Y'); % upload data set
    K = length(Y);
    n = size(X,1);

    % Set up x0 (note that x = [s;r])
    x0 = [-ones(n,1); 0];

    % Setup objetive function and gradient
    h = [X;-ones(1,K)];
    F = @(x) (1/K)*...
        sum(log(1+exp((h'*x)'))-Y.*(h'*x)');
    gradF = @(x) (1/K)*sum((exp((h'*x)')./...
        (1+exp((h'*x)'))-Y).*h,2);
    hessF = @(x)(1/K)*(h*diag(exp(h'*x)./((1+exp(h'*x)).^2))*h');

    % Run Newton algorithm
```

```
    fprintf("Running Newton algorithm for dataset %d (n = %d | K = %d).\n",...
        i,n,K);
    tic
    [xNA,ItNA,normGradNA,alphakNA] = newtonAlgorithm(F,gradF,hessF,x0,epsl,...
        alpha_hat,gamma,beta,maxIt(i));
    elapsedTimeNA = toc;
    if ¬isnan(xNA)
        fprintf("Newton algorithm for dataset %d"+...
        " converged in %d iterations.\n",i,ItNA);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeNA);
        if i≤2
            fprintf("s = [%g; %g] | r = %g.\n",xNA(1),xNA(2),xNA(3));
        end
    else
        fprintf("Newton algorithm for dataset %d "+...
            "exceeded the maximum number of iterations.\n",i);
        fprintf("Elapsed time is %f seconds.\n",elapsedTimeNA);
    end

    % Save data
    save(sprintf("./DATA/NewtonAlgorithm/NAsolDataset%d.mat",i),...
        'xNA','ItNA','normGradNA','alphakNA','elapsedTimeNA');
end

%%
figure('units','normalized','outerposition',[0 0 1 1]);

for i=1:4
    load(sprintf("./DATA/NewtonAlgorithm/NAsolDataset%d.mat",i),...
        'ItNA','normGradNA');

    plot(1:ItNA+1,normGradNA,'LineWidth',3);
    hold on;
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title("Newton Algorithm");
    legend("Dataset 1","Dataset 2","Dataset 3","Dataset 4",'location','best')
    ylabel('$||\nabla f (s_k,r_k)||$','Interpreter','latex');
    xlabel('$k$','Interpreter','latex');
    set(gca, 'YScale', 'log');

    hold on
end

set(gcf,'Units','Inches');
pathFigPos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches',...
'PaperSize',[pathFigPos(3), pathFigPos(4)])
print(gcf,"./DATA/NewtonAlgorithm/NANormGrad",'-dpdf','-r0')
```

```
hold off

%%
figure('units','normalized','outerposition',[0 0 1 1]);

for i=1:4
    load(sprintf("./DATA/NewtonAlgorithm/NAsolDataset%d.mat",i),...
        'ItNA','alphakNA');

    stem(1:ItNA,alphakNA,'LineWidth',3,'MarkerSize',12);
    hold on;
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title("Newton Algorithm");
    legend("Dataset 1","Dataset 2","Dataset 3","Dataset 4",'location','best')
    ylabel('$\alpha_k$','Interpreter','latex');
    xlabel('$k$','Interpreter','latex');

    hold on
end

set(gcf,'Units','Inches');
pathFigPos = get(gcf,'Position');
set(gcf,'PaperPositionMode','Auto','PaperUnits','Inches',...
'PaperSize',[pathFigPos(3), pathFigPos(4)])
print(gcf,"./DATA/NewtonAlgorithm/NAalphak",'-dpdf','-r0')
hold off
```

```
function [xk,k,normGk,alpha_k_found] = newtonAlgorithm(F,gradF,hessF,x0,epsl,...
    alpha_hat,gamma,beta,maxIt)
    %% Description
    % Inputs: 1. F: objective function (as a function handle)
    %         2. gradF: gradient of the objective function (as a function
    %         handle)
    %         3. hessF: hesssian of the objective function (as a function
    %         handle)
    %         4. x0: initialization
    %         5. epsl: stopping criterion
    %         6. alpha_hat: initialization of alpha_k for the backtracking
    %         routine
    %         7. gamma: gamma of backtraking routine
    %         8. beta: beta of backtraking routine
    %         9. maxIt: maximum number of iterations
    % Outputs: 1. x: output of the gradient descent method (returns NaN if
    %          stopping criterion not met after the maximum number of
    %          iterations chosen
    %          2. k: number of iterations required for convergence if a
    %          solution was found
```

```
    %          3. normGk: norm of the gradient of the objective function
    %% Newton method
    k = 0;
    xk = x0;
    normGk = zeros(maxIt,1);
    alpha_k_found = zeros(maxIt,1);
    while k < maxIt
        gk = gradF(xk); % Compute gradient at xk
        normGk(k+1) = norm(gk);
        if normGk(k+1) < epsl % Stopping criterion
           break;
        end
        dk = -hessF(xk)\gk;
        % ---------  backtracking routine  ---------
        alpha_k = alpha_hat;
        % It is guaranteed that there is convergence, no maximum number of
        % iterations needed (obviously for beta < 0)
        while true
            % check if F(alpha_k) < phi(0)+gamma*phi_dot(0)+alpha_k
            if F(xk+alpha_k*dk) < F(xk)+gamma*alpha_k*gk'*dk
                alpha_k_found(k+1) = alpha_k;
                break; % alpha_k found
            else
                alpha_k = beta*alpha_k; % Update alpha_k
            end
        end
        xk = xk + alpha_k*dk; % update xk
        % -------  End backtracking routine  -------
        k = k + 1; % Increment iteration count
    end
    if k == maxIt
        % No solution found within the maximum number of iterations
        xk = NaN;
    else
        normGk = normGk(1:k+1);
        alpha_k_found = alpha_k_found(1:k);
    end
end
```

2.7 Task 7

In this task, the two methods, gradient descent method and Newton method, are compared for each dataset. In particular, their number of iterations, examples of time elapsed for the execution of each algorithm in the same machine, and average time per iteration are compared.

Theoretically, it would be expected that the Newton method should present lower numbers of iterations and higher average time per iteration than the ones from the gradient descent method. In Table 5, the results obtained for a set of experiments on the same ma-

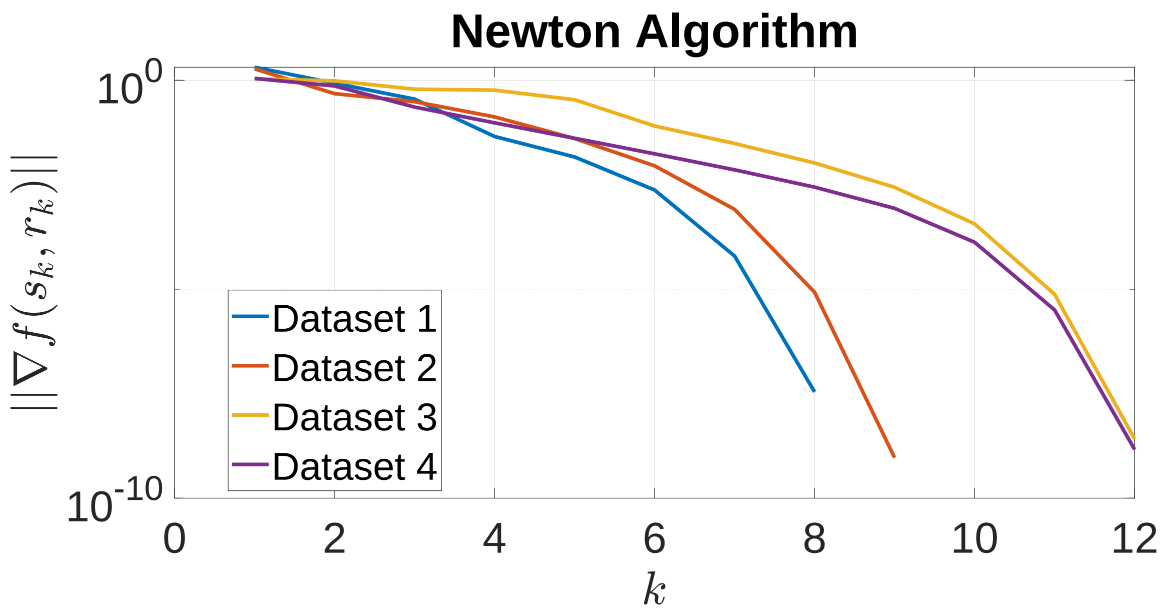


Figure 33: Norm of the gradient across iterations for every dataset.

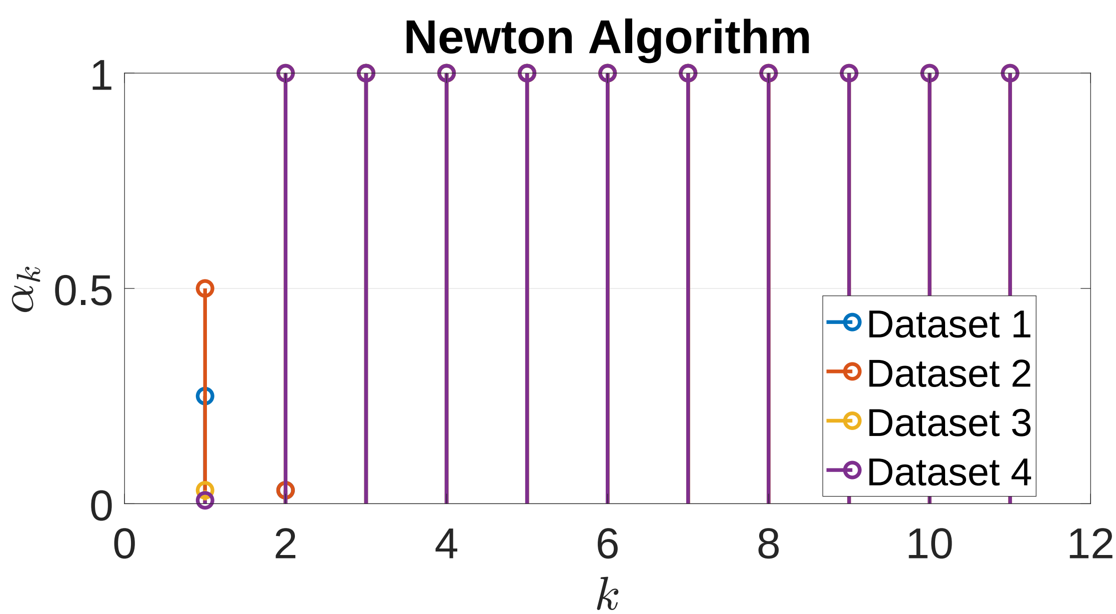


Figure 34: Value of the stepsize across iterations for every dataset.

chine are presented for each method and dataset. In this table, GD refers to the gradient descent algorithm and NA to the Newton algorithm. In Fig. 35 and 36, the solutions obtained with each method for datasets 1 and 2 are compared to show that both present the same results. From Table 5, it is verified that the gradient descent method takes a number of iterations much larger than the Newton algorithm. However, the average time per iteration, that was obtained by simply dividing the time elapsed by the number of iterations, is larger for the Newton algorithm which was also theoretically expected. This was expected since every iteration of the Newton algorithm requires computing the hessian of the objective function and solving a linear system. It is also important to notice that these times are just examples and depend on the used machine and even on its instantaneous state prior to execution. Therefore, they are only useful to verify that the Newton method requires less iterations but more time per iteration than the gradient descent method.

Table 5: Execution data for every dataset and method.

Method	Dataset	# Iterations	Time elapsed [s]	Avg. time/iteration [s]
GD	1	1125	0.078	6.90×10^{-5}
NA	1	7	0.041	2.30×10^{-3}
GD	2	1362	0.082	6.04×10^{-5}
NA	2	8	0.014	3.16×10^{-3}
GD	3	3436	1.00	29.1×10^{-5}
NA	3	11	0.051	465×10^{-3}
GD	4	19892	169	851×10^{-5}
NA	4	11	8.51	774×10^{-3}

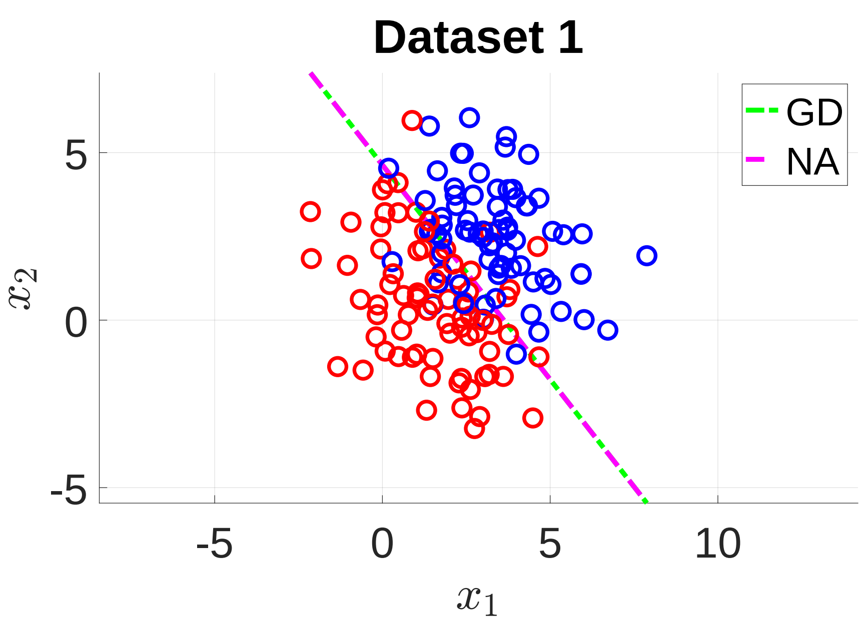


Figure 35: Comparison of the solutions for dataset 1 obtained with both methods.

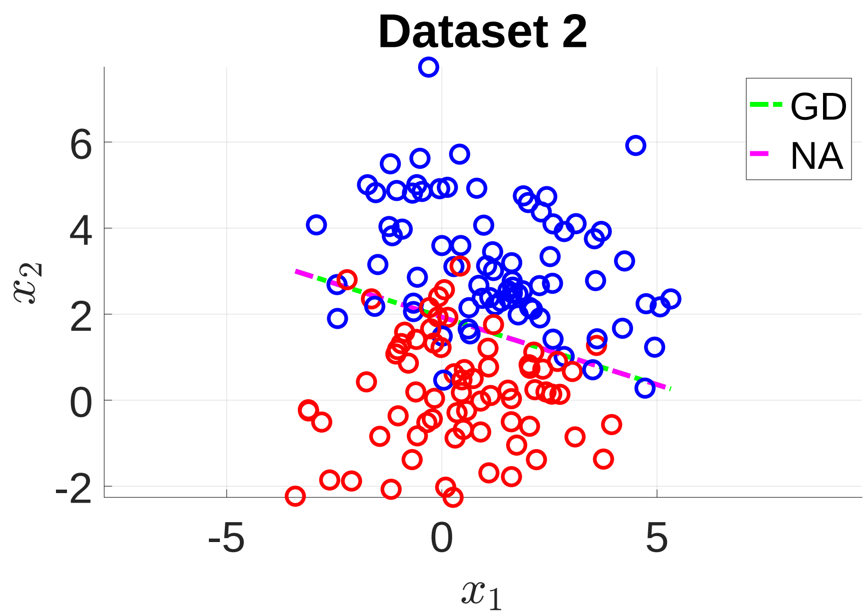


Figure 36: Comparison of the solutions for dataset 2 obtained with both methods.

3 Part 3

In this part, the goal is to solve

$$\underset{\mathbf{y}\in\mathbf{R}^{Nk}}{\text{minimize}} \quad \sum_{m=1}^{N}\sum_{n=m+1}^{N}\left(||\mathbf{y_m}-\mathbf{y_n}||_2 - D_{mn}\right)^2, \tag{43}$$

where $\mathbf{D} \in \mathbb{R}^{N\times N}$.

3.1 Task 1

The dataset in file `data_opt.csv` is loaded and the corresponding matrix D is computed according to $D_{mn} = ||\mathbf{x_m} - \mathbf{x_n}||_2$. The following MATLAB script solves task 1

```
%% Part 3 - Task 1 (part3task1.m)
%% Load dataset from file data_opt.csv
X = csvread("./data/data_opt.csv");
N = size(X,1); % Get number of datapoints
%% Compute matrix D
D = zeros(N); % Initialize matrix D
for m = 1:N % Four each off-diagonal pair of coordinates
    for n = m+1:N
       D(m,n) = norm(X(m,:)-X(n,:),2); % Compute D_{mn}
       D(n,m) = D(m,n); % D_{nm}=D_{mn}
    end
```

```
end
%% Check results
% Find maximum value of distance and repective indices
Dmax = max(max(D));
[mDmax,nDmax] = find(D==Dmax);
% Output results
fprintf("------------------------ Task 1 ------------------------\n");
fprintf("D(2,3) = %g | D(4,5) = %g.\n", D(2,3),D(4,5));
fprintf("max{D(m,n)} = %g for (m,n) = {(%d,%d),(%d,%d)}.\n",...
    Dmax,mDmax(1),nDmax(1),mDmax(2),nDmax(2))
%% Save data
save("./data/distancesTask1.mat",'D','Dmax','nDmax','mDmax','N');
```

obtaining

$$D_{2,3} = 5.8749, \quad D_{4,5} = 24.3769$$

and

$$\max(D_{mn}) = 83.003 \quad \text{for} \quad (m, n) \in \{(134, 33), (33, 134)\} .$$

3.2 Task 2

One has

$$f(\mathbf{y}) = \sum_{m=1}^{N} \sum_{n=m+1}^{N} (||\mathbf{y_m} - \mathbf{y_n}|| - D_{mn})^2 = \sum_{m=1}^{N} \sum_{n=m+1}^{N} f_{mn}(\mathbf{y})^2 , \tag{44}$$

where $\mathbf{y_m} \in \mathbb{R}^m$, k is the dimension of the target space, $\mathbf{y} = \mathrm{col}(\mathbf{y_1}, \ldots, \mathbf{y_N}) \in \mathbb{R}^{Nk}$ is the optimization variable, and

$$f_{mn}(\mathbf{y}) := ||\mathbf{y_{m-n}}|| - D_{mn} , \tag{45}$$

defining $\mathbf{y_{m-n}}$ as $\mathbf{y_{m-n}} := \mathbf{y_m} - \mathbf{y_n}$.

Note that one can write $\mathbf{y_m} = \mathbf{E_m y}$, where $\mathbf{E_m} \in \mathbb{R}^{k\times Nk}$ is defined as

$$\mathbf{E_m} := \begin{bmatrix} \mathbf{0}_{k\times k(m-1)} & \mathbf{I}_{k\times k} & \mathbf{0}_{k\times k(N-m)} \end{bmatrix} ,$$

thus, it is possible to rewrite (45) as

$$f_{mn}(\mathbf{y}) = ||\mathbf{E_m y} - \mathbf{E_n y}|| - D_{mn} = \sqrt{\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}} - D_{mn} . \tag{46}$$

Taking the jacobian of (46), one obtains

$$\begin{aligned} D_{\mathbf{y}} f_{m,n}(\mathbf{y}) &= D_u(\sqrt{u})\Big|_{u=\mathbf{y}^T(\mathbf{E_m}-\mathbf{E_n})^T(\mathbf{E_m}-\mathbf{E_n})\mathbf{y}} D_{\mathbf{y}}(\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}) \\ &= \mathbf{y}^T \frac{(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})}{\sqrt{\mathbf{y}^T(\mathbf{E_m} - \mathbf{E_n})^T(\mathbf{E_m} - \mathbf{E_n})\mathbf{y}}} \\ &= \frac{\begin{bmatrix} \mathbf{0}_{1\times(m-1)k} & \mathbf{y_{m-n}}^T & \mathbf{0}_{1\times(n-m-1)k} & -\mathbf{y_{m-n}}^T & \mathbf{0}_{1\times(N-n)k} \end{bmatrix}}{||\mathbf{y_{m-n}}||} \end{aligned} \tag{47}$$

therefore the gradient $\nabla_{\mathbf{y}} f_{mn}(\mathbf{y}) = (D_{\mathbf{y}} f_{mn}(\mathbf{y}))^T$.

Similarly taking the jacobian of (44), one obtains

$$D_{\mathbf{y}} f(\mathbf{y}) = \sum_{m=1}^{N} \sum_{n=m+1}^{N} D_u(u^2)\Big|_{u=f_{mn}(\mathbf{y})} D_{\mathbf{y}} f_{mn}(\mathbf{y}) = \sum_{m=1}^{N} \sum_{n=m+1}^{N} 2 f_{mn}(\mathbf{y}) D_{\mathbf{y}} f_{mn}(\mathbf{y})$$

therefore the gradient $\nabla_{\mathbf{y}} f(\mathbf{y}) = (D_{\mathbf{y}} f(\mathbf{y}))^T$ is given by

$$\nabla_{\mathbf{y}} f(\mathbf{y}) = \sum_{m=1}^{N} \sum_{n=m+1}^{N} 2 f_{mn}(\mathbf{y}) \nabla_{\mathbf{y}} f_{mn}(\mathbf{y}) \tag{48}$$

In conclusion, $f(\mathbf{y})$, $f_{mn}(\mathbf{y})$,$\nabla_{\mathbf{y}} f_{mn}(\mathbf{y})$, and $\nabla_{\mathbf{y}} f(\mathbf{y})$ can be computed making use of (44),(45), (47), and (48), respectively. Also note that each of these four quantities may be computed making use of differences of the optimization vector exclusively, *i.e.*, the $N(N/2-1)$ vectors $\mathbf{y_{m-n}}$. As it is explored herein this property allows for considerable optimization of the computational load required to solve the optimization problem.

For the implementation of the Levenberg-Marquardt (LM) method it is required to compute, for each new iteration, $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, matrix $\mathbf{A}$, and vector $\mathbf{b}$, defined by

$$\mathbf{A} := \begin{bmatrix} D_{\mathbf{y}} f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}} f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}} f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda} \mathbf{I}_{Nk \times Nk} \end{bmatrix} \quad \text{and} \quad \mathbf{b} := \begin{bmatrix} D_{\mathbf{y}} f_{1,1}(\mathbf{y})\mathbf{y} - f_{1,1}(\mathbf{y}) \\ D_{\mathbf{y}} f_{1,2}(\mathbf{y})\mathbf{y} - f_{1,2}(\mathbf{y}) \\ \vdots \\ D_{\mathbf{y}} f_{N-1,N}(\mathbf{y})\mathbf{y} - f_{N-1,N}(\mathbf{y}) \\ \sqrt{\lambda}\mathbf{y} \end{bmatrix} . \tag{49}$$

For that purpose a MATLAB function independent of the LM algorithm is devised. This allows to implement the LM algorithm separately, which can then be applied to any optimization problem of suitable form, and not being constrained to the problem at hand in this part. To allow for a computationally efficient algorithm to compute, at each iteration, the relevant quantities related to the objective function, first note that

$$D_{\mathbf{y}} f_{mn}(\mathbf{y})\mathbf{y} - f_{mn}(\mathbf{y}) = \frac{\mathbf{y_{m-n}}(\mathbf{y_m} - \mathbf{y_n})}{||\mathbf{y_{m-n}}||} - ||\mathbf{y_{m-n}}|| + D_{mn} = D_{mn} . \tag{50}$$

Thus, noticing that $\mathbf{b}$ in (49) results of the concatenation of terms of the form (50), $\mathbf{b}$ is computed according to

$$\mathbf{b} = \begin{bmatrix} D_{1,1} & D_{1,2} & \dots & D_{N-1,N} & \sqrt{\lambda}\mathbf{y} \end{bmatrix} , \tag{51}$$

which is very efficiently computed since it does not have to be carried out iteratively. Furthermore, a significant portion of $\mathbf{b}$ is constant, which has to be computed just once in the LM algorithm. Second, the computation of $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, and $\mathbf{A}$ is performed iteratively, running one iteration for each of the $N(N/2-1)$ vectors $\mathbf{y_{m-n}}$. Therefore, it is more efficient to compute all of these quantities at once. Third, λ is a variable of the LM method, which does not depend directly on the objective function, thus, it was chosen that the entries of $\mathbf{A}$ and $\mathbf{b}$ which are dependent on λ are computed in the LM algorithm function. Fourth, an effort was made so that there is not replication of computations. Given that the quantities computed depend essentially on each other, $\mathbf{y_{m-n}}$, or $||\mathbf{y_{m-n}}||$, it allows to reduce the computational load significantly.

Having the aforementioned optimization guidelines in mind the following MATLAB function was designed

```
function [costF,normG,A,b] = objectiveF(y)
%% objectiveF.m
% Input: y: vector at which the quatities are evaluated
% Ouput: costF: cost function value
%        normG: norm of the gradient of the cost function
%        A: matrix A for the application of the LM method (only the entries
%        that do not depend on lambda)
%        b: vetor b for the application of the LM method (only the entries
%        that do not depend on lambda)
%% Initialize cost function dataset
% Load dataset in the first call to this function
persistent b_ N k % do not have to be recomputed between calls
if isempty(k) ||isempty(b_) || isempty(N)
    % Load data:
    % D: Distance matrix
    % N: Number of data points
    % k: Dimension of the target space
    load("./data/objectiveFData.mat",'D','N','k');
    % Compute the portion of b that is constant
    b_ = nonzeros(tril(D,-1));
    fprintf("Initializing dataset.\n");
end
%% Compute quantities
costF = 0; % Initialize costF
gradf = zeros(1,N*k); % Initialize gradf
A = zeros((N^2-N)/2+N*k,N*k); % Initialize A
b = [b_;zeros(N*k,1)]; % Compute entries of b that do not depend on lambda
count = 1; % Iteration count
for i = 1:N
   for j = i+1:N
        dy = (y((i-1)*k+1:(i-1)*k+k)-y((j-1)*k+1:(j-1)*k+k))'; % y_{m-n}
        normdy = norm(dy); % ||y_{m-n}||
        faux = normdy-b_(count); % f_{mn} = ||y_{m-n}||-D_{mn}
        gradaux_ = [zeros(1,(i-1)*k) dy zeros(1,k*(j-i-1))...
            -dy zeros(1,(N-j)*k)]/(normdy); % D_y(f{mn}(y))
        costF = costF + faux^2; % f(y) += f_{mn}(y)^2
        % D_y(f(y))(y) += 2*f_{mn}(y)*D_y(f{mn}(y))
        gradf = gradf + 2*faux*gradaux_;
        A(count,:) = gradaux_; % A(<->) = D_y(f{mn}(y))
        count = count+1; % increment iteration count
```

```
    end
end
normG = norm(gradf); % compute norm of D_y(f(y))(y)
end
```

This implementation allows for a decrease of computation time of roughly two orders of magnitude compared with a first naive implementation.

3.3 Task 3

In this task the optimization problem is solved for the dataset loaded in Task 1, for $k \in \{2, 3\}$, using the LM algorithm. For that reason, a generic implementation of this algorithm was implemented in MATLAB

```
function [xk,k,costF,normGk] = LMAlgorithm(lambda0,x0,epsl,maxIt)
%% LMAlgorithm.m
% Input: lambda0: initialization lambda of the LM algorithm
%          x0: initialization solution estimate
%          epsl: stopping criterion
%          maxIt: maximum number of iterations
% Output: xk: output of the gradient descent method (returns NaN if
%          stopping criterion not met after the maximum number of
%          iterations chosen
%          k: number of iterations required for convergence if a
%          solution was found
%          costF: vector of objective function value for each iteration
%          normGk: vector of the norm of the gradient of the objective
%          function for each iteration
%% LM algorithm
% Initialize variables
costF = zeros(maxIt,1); % vector of objective function value
normGk = zeros(maxIt,1); % vector of gradient norms of the objective f
% Initialize LM algorithm
k = 0; % Initialize iteration count
xk = x0; % Initialization solution estimate
lambdak = lambda0; % Initialization lambda
fprintf("Running LM.\n");
% ---------------------------  LM algorithm  ---------------------------
while k < maxIt % iterate up to a limit of maxIt iterations
    % -----------  Compute objective function and gradients  -----------
    if k % store previous A and b, which are used if the step is invalid
        Aprev = A;
        bprev = b;
    end
    % Compute objective f value, norm of the gradient, A and b for xk
    % note that only the entries of A and b that do not depend on lambda
    % are computed
    [costF(k+1),normGk(k+1),A,b] = objectiveF(xk);
    % -------------------  check stopping criterion  --------------------
```

```
    if normGk(k+1) < epsl % Stopping criterion
            break;
    end
    %  ----------------  Check validity of last step  ------------------
    if k && costF(k+1)<costF(k) % if step is valid
        lambdak = 0.7*lambdak; % decrease lambda
    elseif k % if step is invalid
        xk = xprev; % redo step
        A = Aprev;
        b = bprev;
        normGk(k+1) = normGk(k);
        costF(k+1) = costF(k);
        lambdak = 2*lambdak; % decrease lambda
    end
    % ------------  New step (solve least squares problem)  ------------
    % store previous estimate, which is used if the step is invalid
    xprev = xk;
    % Entries of A and b that depend on lambda must be computed
    A(end-size(x0,1)+1:end,:) = sqrt(lambdak)*eye(size(x0,1));
    b(end-size(x0,1)+1:end,1) = sqrt(lambdak)*xk;
    xk = ((A'*A)\A')*b; % solve least squares problem Ax=b

    % ------------------  Increment iteration count  -------------------
    % Display LM status
    if k fprintf("Iteration: %d | cost = %g.\n",k,costF(k+1)); end
    k = k+1; % increment iteration count
end
if k == maxIt
    % No solution found within the maximum number of iterations
    xk = NaN; % Output invalid estimate
else
    % Output only relevant data
    costF = costF(2:k+1);
    normGk = normGk(2:k+1);
end
end
```

Note that this implementation relies on the fact that $f(\mathbf{y})$, $||\nabla_{\mathbf{y}} f(\mathbf{y})||$, matrix $\mathbf{A}$, and vector $\mathbf{b}$ are computed at once. It is important to remark, however, that each time a step is invalid, the new $\mathbf{A}$ and $\mathbf{b}$ that were computed are useless, which represents a waste of computational power. It was verified that, for this particular optimization problem, the reduction that is achieved computing all quantities at once is greater than that obtained if $\mathbf{A}$ and $\mathbf{b}$ are only computed when necessary.

The following MATLAB script was then run to solve the optimization problem

```
%% Part 3 - Task 3 (part3task3.m)
%% Initialize cost function dataset
% Load distances matrix of the dataset of task 1
load("./data/distancesTask1.mat",'D','N');
```

```matlab
% Initialize variables to hold the solution and status parameters of the LM
% algorithm for K = 2,3
solLM = cell(2,1); % solution of the optimization problem
itLM = zeros(2,1); % number of iterations ran
elapsedTimeLM = zeros(2,1); % time elapsed running LM
costLM = cell(2,1); % vector of cost function values for each iteration
% vector of gradient norm of the cost function for each iteration
normGradLM = cell(2,1);

%% Solve optimization problem for k = 2,3
for k = 2:3 % target space dimension
    % Set up parameters
    maxIt = 200; % maximum number of iterations
    lambda0 = 1; % initial value for lambda of the LM method
    epsl = k*1e-2; % stopping criterion
    % set up data for the compuattion of the quatities related to the
    % objective function in objectiveF(y)
    save("./data/objectiveFData.mat",'D','N','k');
    y0 = csvread(sprintf("./data/yinit%d.csv",k)); % initialization of LM
    clear objectiveF; % clear persistent variables in objectiveF
    fprintf("------------------------ Task 3 ------------------------\n");
    tic; % start counting LM time
    % run LM method
    [solLM{k-1,1},itLM(k-1,1),costLM{k-1,1},normGradLM{k-1,1}] =...
        LMAlgorithm(lambda0,y0,epsl,maxIt);
    elapsedTimeLM(k-1,1) = toc; % save elapsed time
    if ¬isnan(solLM{k-1,1}) % if a solution was found
        fprintf("Solution found for dataset of task 1 with k = %d "+...
            "using LM algorithm.\n",k);
        fprintf("- Objective function value: %g.\n",costLM{k-1,1}(end,1));
        fprintf("- Elapsed time: %g s.\n", elapsedTimeLM(k-1,1));
    else % if a solution was not found
        fprintf("Solution could not be found for dataset of task 1 "+...
            "with k = %d using\n LM algorithm with the provided "+...
            "stopping criterion and maximum number of iterations.\n",k);
    end
end
% Save solutions
save("./data/solTask3.mat",...
    'solLM','itLM','elapsedTimeLM','costLM','normGradLM');
%% Plot results
for k = 2:3
    figure('units','normalized','outerposition',[0 0 1 1]);
    yyaxis left
    plot(0:itLM(k-1,1)-1,costLM{k-1,1},'LineWidth',3);
    hold on;
    ylabel('$f(y)$','Interpreter','latex');
    set(gca, 'YScale', 'log');
    yyaxis right
    plot(0:itLM(k-1,1)-1,normGradLM{k-1,1},'LineWidth',3);
    ylabel('$||\nabla f (y)||$','Interpreter','latex');
```

```
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
    set(gca, 'YScale', 'log');
    xlabel('$k$','Interpreter','latex');
    saveas(gcf,sprintf("./data/task3_LM_k_%d.fig",k));
    hold off;
    y = reshape(solLM{k-1},[k,N]);
    figure('units','normalized','outerposition',[0 0 1 1]);
    if k == 2
        scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,...
            'MarkerFaceColor','flat');
    else
        scatter3(y(1,:),y(2,:),y(3,:),100,'o','b','LineWidth',1,...
            'MarkerFaceColor','flat');
    end
    hold on;
    set(gca,'FontSize',35);
    ax = gca;
    ax.XGrid = 'on';
    ax.YGrid = 'on';
    title(sprintf("LM algorithm | Dataset task 1 | k = %d",k));
    saveas(gcf,sprintf("./data/task3_lowerDim_k_%d.fig",k));
    hold off;
end
```

The results obtained for $k = 2$ are shown in Figs. 37 and 38, and for $k = 3$ in Figs. 39 and 40.

First, it is noticeable that the algorithm converges and reaches the expected solution for both values of k. Second, the value of the objective function of both solutions is shown in Table 6. It is visible that the value of the cost function for $k = 3$ decreases by a factor of roughly 2.7 in relation to the solution with $k = 2$. Thus, $k = 3$ fits much better to the dataset. Third, note that the solution using $k = 3$ requires more iterations of the LM algorithm than what is presented in the provided results. In fact, observing the evolution of the norm of the gradient of the objective function, visible in Fig. 39, it is possible to detect the presence of numerical error starting at the 70-th iteration, which arises using MATLAB 2018a. For this reason, although the solution is identical, it takes more iterations to reach the stopping criterion.

Table 6: Value of the objective function of both solutions.

k	$f(\mathbf{y})$
2	7486.6
3	2779.2

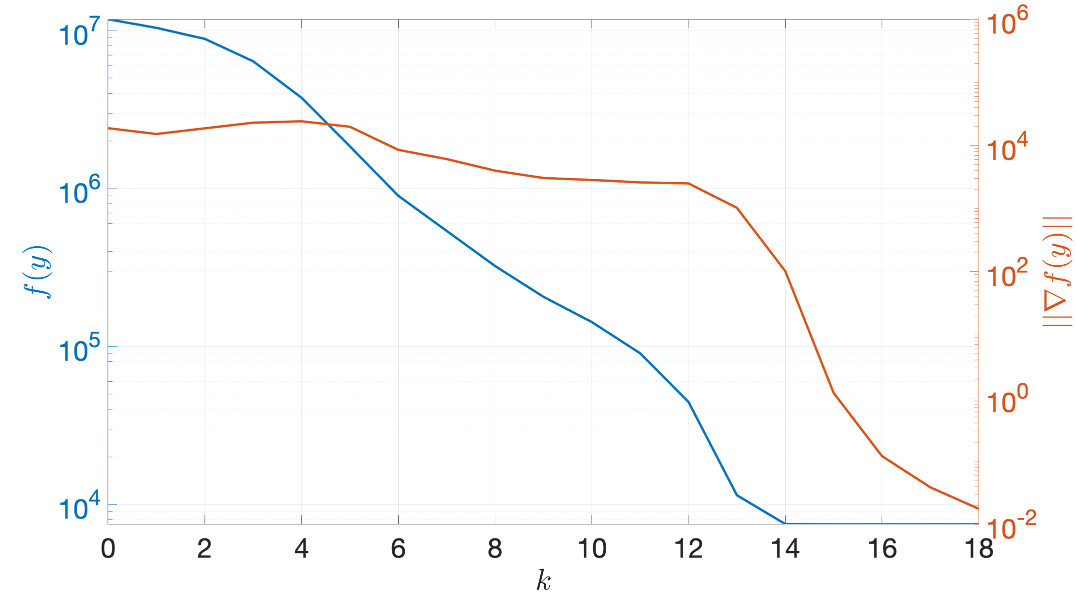


Figure 37: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

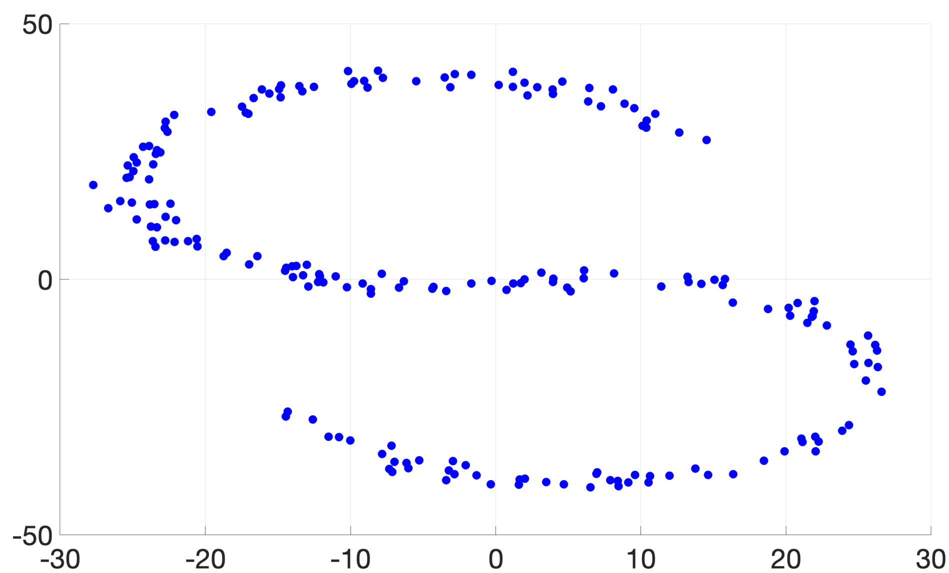


Figure 38: Solution to the optimization problem using the LM algorithm for $k = 2$.

3.4 Task 4

The LM method is now applied to dataset `dataProj.csv`, and we are not provided with an initialization $\mathbf{y}_0$. There is no guarantee that a solution found by the LM method is the

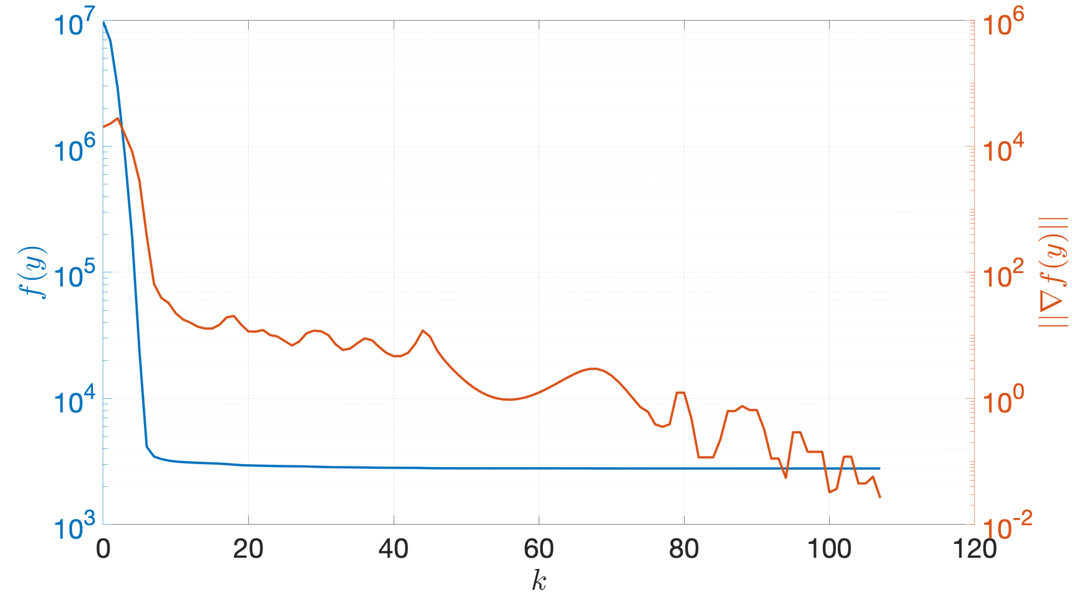


Figure 39: Objective function value and gradient norm throughout the iterations of the LM algorithm for $k = 2$.

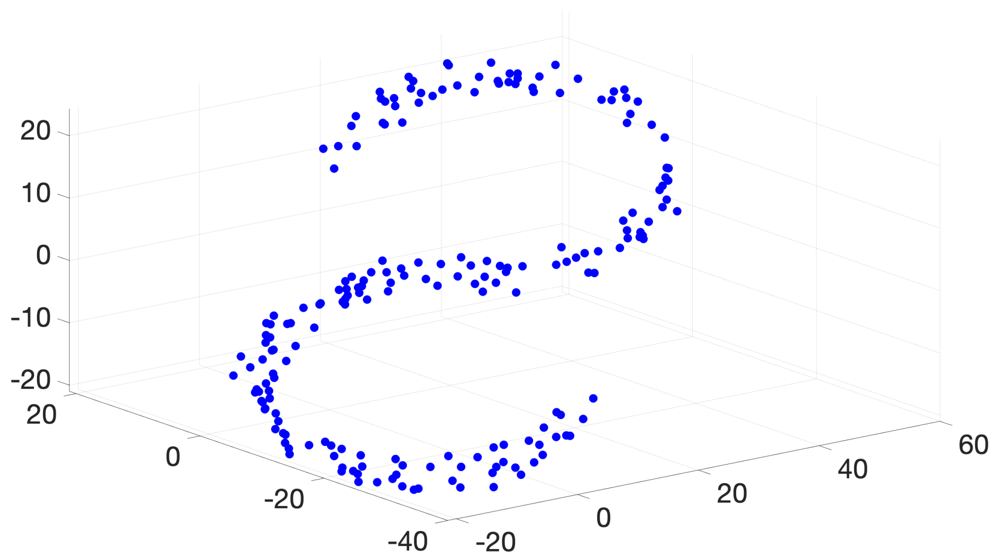


Figure 40: Solution to the optimization problem using the LM algorithm for $k = 2$.

global solution, since the objective function is not convex. For this reason, to find a good suboptimal solution, the method has to be run several times for different randomly generated initializations. The solutions are then sorted according to the value of the objective function, of which the best is chosen. It is very important to remark that the computation of each of the solutions can be run in a parallel manner, using the *Parallel toolbox* in MATLAB, for instance. The following MATLAB script solves task 4.

```
%% Part 3 - Task 4 (part3task4.m)
% Various runs can be performed. In each run the LM algorithm is run NIts
% times all for randmly generated initializations
%% Set parameters
RUN = 1; % run number
NRuns = 1; % number of runs so far
NIts = 12*2; % number of times the LM algorith is called in a run
%% Load or compute data
computedD = false; % if D is already computed just load it
if ¬computedD % C was not computed -> compute it now
    X = csvread("./data/dataProj.csv");
    N = size(X,1);
    D = zeros(N);
    for m = 1:N
        for n = m+1:N
           D(m,n) = norm(X(m,:)-X(n,:),2);
           D(n,m) = D(m,n);
        end
    end
    % Save data
    save("./data/distancesTask4.mat",'D','N');
else % D was already computed -> compute it now
    load("./data/distancesTask4.mat",'D','N');
end
%% Run various times LM for random initializations
% Set up parameters
k = 2; % target space dimension
lambda0 = 1; % initial value for lambda of the LM method
maxIt = 200; % maximum number of iterations
epsl = k*1e-4; % stopping criterion
% Initialize variables to hold the solution and status parameters of the LM
% algorithm for the NIts LM calls
solLM = cell(NIts,1); % solution of the optimization problem
itLM = zeros(NIts,1); % number of iterations ran
elapsedTimeLM = zeros(NIts,1); % time elapsed running LM
costLM = cell(NIts,1); % vector of cost function values for each iteration
% vector of gradient norm of the cost function for each iteration
normGradLM = cell(NIts,1);
fprintf("----------------------- Task 4 -----------------------\n");
clear objectiveF; % clear persistent variables in objectiveF
save("./data/objectiveFData.mat",'D','N','k');
parfor it = 1:NIts % calls of LM can be run in parallel
    % each entry of y is randomly generated from an uniform distribution
    % between -200 and 200
    y0 = 200*2*(rand()-0.5)*2*(rand(N*k,1)-0.5);
    tic; % start counting LM time
    fprintf("----------------- RUN %02d - Attempt %02d ----------------\n",...
```

```
        RUN,it);
     % run LM method
    [solLM{it,1},itLM(it,1),costLM{it,1},normGradLM{it,1}] =...
        LMAlgorithm(lambda0,y0,epsl,maxIt);
    elapsedTimeLM(it,1) = toc; % save elapsed time
end
% Save whole run
save(sprintf("./data/RunsTask4/solRUN%02d.mat",RUN),...
    'solLM','itLM','elapsedTimeLM','costLM','normGradLM');

%% Sort solutions found in all runs
solSorted = zeros(NRuns*NIts,3); % sorted list of all solutions
count = 0; % count number of solutions
for i = 1:NRuns
   data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",i));
   for j = 1:NIts
       count = count+1;
       solSorted(count,1) = i; % 1st column has run number
       solSorted(count,2) = j; % 2nd column has attempt number within run
       solSorted(count,3) = data.costLM{j,1}(end,1);  % 2nd column has cost
   end
end
solSorted = sortrows(solSorted,3); % sort rows ascending cost
save("./data/solSortedTask4.mat",'solSorted'); % save sorted solutions

% Best solution
% Load best solution run
data = load(sprintf("./data/RunsTask4/solRUN%02d.mat",solSorted(1,1)));
% Get best solution data and save it
solLM = data.solLM{solSorted(1,2),1};
itLM = data.itLM(solSorted(1,2),1);
elapsedTimeLM = data.elapsedTimeLM(solSorted(1,2),1);
costLM = data.costLM{solSorted(1,2),1};
normGradLM = data.normGradLM{solSorted(1,2),1};
% Save best solution data
save("./data/solBestTask4.mat",...
     'solLM','itLM','elapsedTimeLM','costLM','normGradLM');

%% Plot best solution
figure('units','normalized','outerposition',[0 0 1 1]);
yyaxis left
plot(0:itLM(k-1,1)-1,costLM,'LineWidth',3);
hold on;
ylabel('$f(y)$','Interpreter','latex');
set(gca, 'YScale', 'log');
yyaxis right
plot(0:itLM(k-1,1)-1,normGradLM,'LineWidth',3);
ylabel('$||\nabla f (y)||$','Interpreter','latex');
set(gca,'FontSize',35);
ax = gca;
ax.XGrid = 'on';
```

```
ax.YGrid = 'on';
title(sprintf("LM algorithm | Dataset task 4 | k = %d",k));
set(gca, 'YScale', 'log');
xlabel('$k$','Interpreter','latex');
saveas(gcf,"./data/task4_LM.png");
hold off;
y = reshape(solLM,[k,N]);
figure('units','normalized','outerposition',[0 0 1 1]);
scatter(y(1,:),y(2,:),100,'o','b','LineWidth',1,'MarkerFaceColor','flat');
hold on;
set(gca,'FontSize',35);
ax = gca;
ax.XGrid = 'on';
ax.YGrid = 'on';
title(sprintf("LM algorithm | Dataset task 4 | k = %d",k));
saveas(gcf,sprintf("./data/task4_sol.png"));
hold off;
```

The LM algorithm was run for 24 different randomly generated initialization vectors, in a parallel manner. The best solution, obtained for $k = 2$, is shown in Figs. 41 and 42, achieving an objective function value of $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$. First, the parallel computation of the various solutions allowed for a significantly faster computation. Second, it was verified that all solutions obtained have identical objective function values, which on its own does not imply that it is the global solution. Notice that the objective function is nonnegative. Furthermore, it is verified that if the stopping criteria parameter is lowered then this method yields an objective function value that it closer to zero. Also, the order of magnitude of the entries of D is substantially greater than the order of magnitude of the objective function value at the solutions. For these reasons, even though this claim is not theoretically correct, it is possible to assume in a practical sense that one approximation very close to a global minimizer was found.

It is now important to analyze the uniqueness of the solutions. In fact consider

$$\bar{\mathbf{y}} = \operatorname{col}(\mathcal{T}\mathbf{y_1} + \mathbf{w}, \dots, \mathcal{T}\mathbf{y_N} + \mathbf{w})\,, \tag{52}$$

where $\mathcal{T} \in \mathbb{R}^{k\times k}$ is a rotation matrix and $\mathbf{w} \in \mathbb{R}^k$. It is easily verified that

$$||\bar{\mathbf{y}}_\mathbf{m} - \bar{\mathbf{y}}_\mathbf{n}|| = ||\mathbf{y_m} - \mathbf{y_n}||$$

for every pair $(m, n) : m \in \{1, \dots, N\}, n \in \{1, \dots, N\}$. Therefore,

$$f(\bar{\mathbf{y}}) = f(\mathbf{y})\,.$$

It is, then, evident that if a global minimum is found, there are infinitely many other global minimums obtained via a rotation and translation of every $\mathbf{y_i}$, $i \in \{1, \dots, N\}$. For this reason the previously shown solution, conjectured to be an estimate of a global solution, is not unique. In fact, the solution of the second best objective function value found, out of the 24 computations, is shown in Fig. 43. In fact, even though both solutions achieve practically

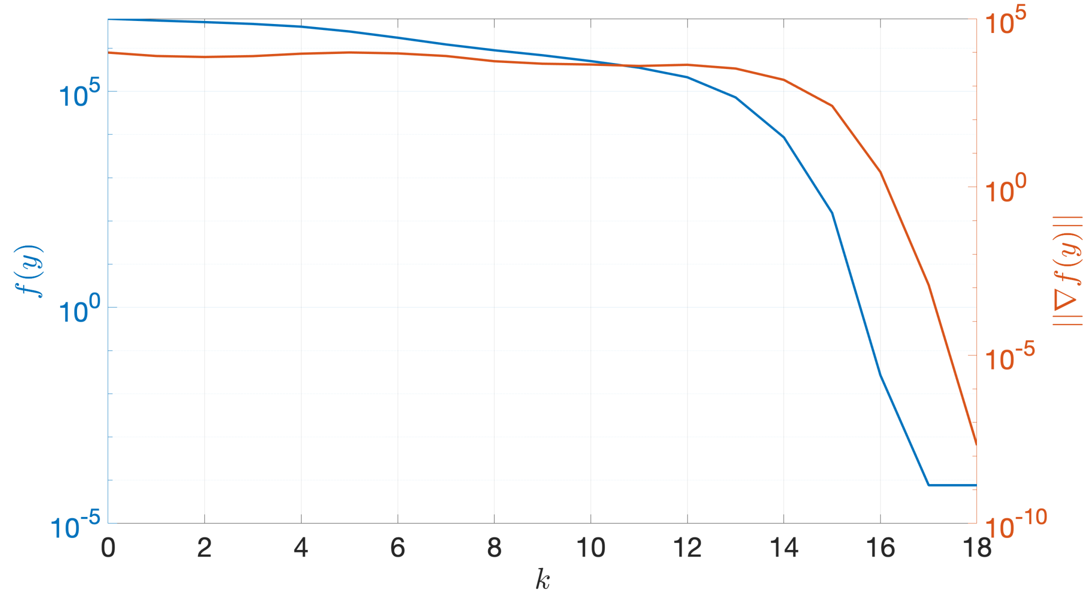


Figure 41: Objective function value and gradient norm throughout the iterations of the LM algorithm for the dataset of task 4 and $k = 2$.

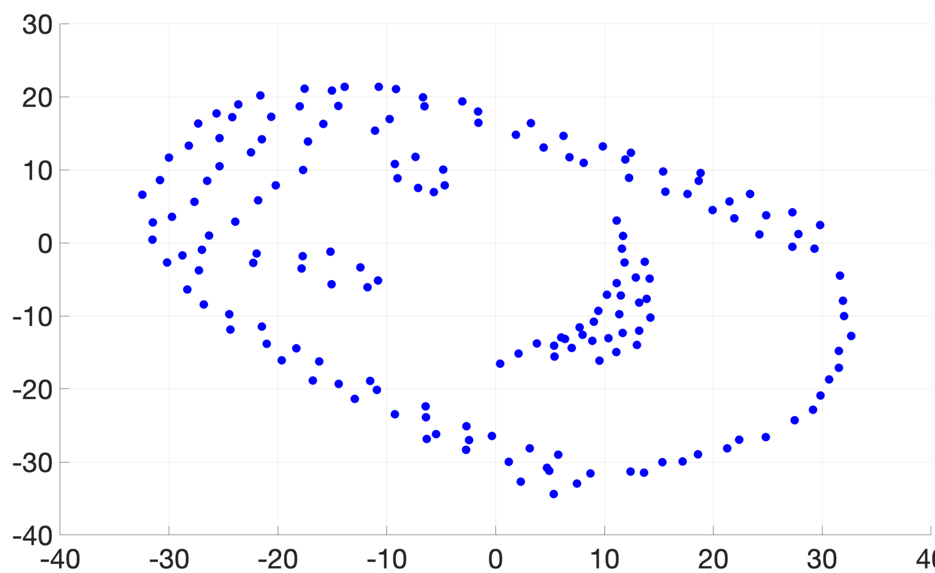


Figure 42: Best solution to the optimization problem using the LM algorithm for the dataset of task 4 and $k = 2$, obtaining $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$.

the same objective function value $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$, they are the result of a rotation and translation of each other.

The following question now arises naturally: if the additional degrees of freedom (d.o.f.)

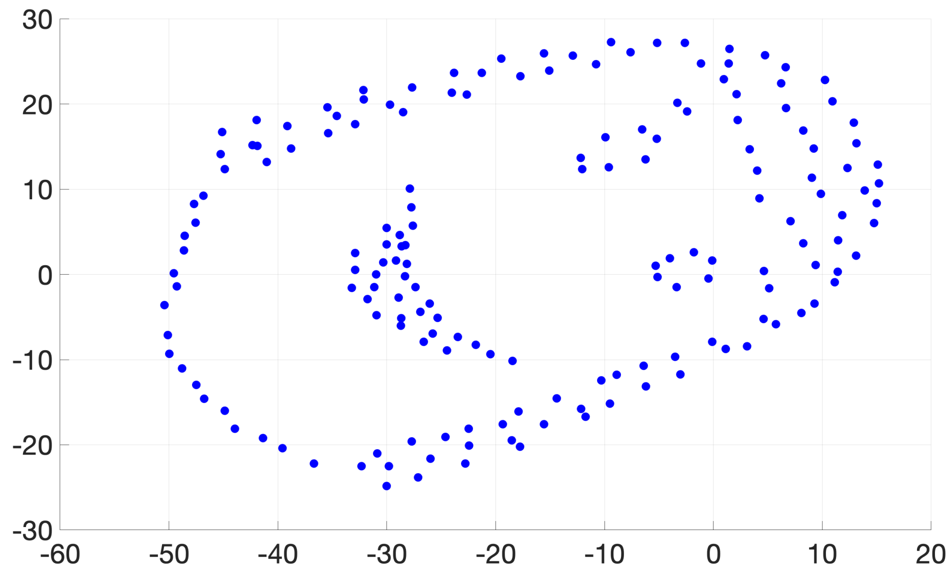


Figure 43: Second best solution to the optimization problem using the LM algorithm for the dataset of task 4 and $k = 2$, obtaining $f(\mathbf{y}_{sol}) = 7.6430 \times 10^{-5}$.

of the rotation ($k-1$ d.o.f.) and of the translation (k d.o.f) are suppressed, constraining the optimization problem, is a solution obtained unique, *i.e.*, are all the solutions of the optimization problem of the form (52)? First, the optimization problem can be constrained imposing

$$\mathbf{y_1} = \mathbf{0}_{k\times 1} \quad \text{and} \quad \mathbf{y_2} = \alpha \text{col}(1, \mathbf{0}_{(k-1)\times 1})\,,$$

with $\alpha \in \mathbb{R}$. The optimization problem becomes

$$\begin{array}{ll} \underset{(\alpha, \mathbf{y_3}, \ldots, \mathbf{y_N}) \in \mathbb{R}\times\mathbf{R}^k \times \ldots \times \mathbf{R}^k}{\text{minimize}} & (\alpha - D_{1,2})^2 + \sum_{n=3}^{N} (||\mathbf{y_n}||_2 - D_{1,n})^2 \\ & + \sum_{n=3}^{N} \left(||\alpha \text{col}(1, \mathbf{0}_{(k-1)\times 1}) - \mathbf{y_n}||_2 - D_{2,n}\right)^2 \\ & + \sum_{m=3}^{N} \sum_{n=m+1}^{N} (||\mathbf{y_m} - \mathbf{y_n}||_2 - D_{mn})^2 \end{array}, \tag{53}$$

with $\mathbf{y} = \text{col}\left(\mathbf{0}_{k\times 1}, \alpha\text{col}(1, \mathbf{0}_{(k-1)\times 1}), \mathbf{y_3} \ldots, \mathbf{y_N}\right) \in \mathbb{R}^{Nk}$, which is still nonconvex. Furthermore, it is easily proven that if a global solution is found, it is not necessarily unique. As a matter of fact, considering

$$\mathbf{D} = \begin{bmatrix} 0 & 4 & \sqrt{5} \\ 4 & 0 & 5 \\ \sqrt{5} & 5 & 0 \end{bmatrix}$$

both

$$\mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \\ 1 \\ 4 \end{bmatrix} \quad \text{and} \quad \mathbf{y} = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \\ 1 \\ -4 \end{bmatrix}$$

are solutions to (53), with $f(\mathbf{y}) = 0$. Notice that they can not be obtained from each other via a rotation and translation. Therefore, even if the additional $2k-1$ degrees of freedom of the solutions to the original problem (43) are suppressed, there is no guarantee that if a global solution is found, it is unique. In fact, inspired by this example it is not difficult to notice that if every $\mathbf{y_m}$ is reflected on the axis corresponding to the first coordinate, then the objective function value remains unchanged. Even if this d.o.f. is suppressed via an additional constraint, we could not arrive at any uniqueness guarantees.

In conclusion, given the thorough analysis conducted, the solution found is not unique. In fact, infinitely many solutions can be found via a translation, rotation, and/or reflection on the axis corresponding to the first coordinate.

3.5 Task 4 - cMDS

The problem at hand is, in fact, very well-studied. In fact, the classical metric Multidimensional Scaling (cMDS) problem is well-known and very easily solved. It is detailed in [CC08], and [Seb09], for instance. In fact, it is shown that metric cMDS is easily solved using singular value decomposition. The solution obtained for the method put forward in [Jac19] is shown in Fig. 44. The algorithm proposed in [Jac19] is implemented in MATLAB in the following script.

```
%% Part 3 - Task 4 extra (part3task4_extra.m)
%% Set parameters
clear;
k = 2; % target dimension

%% Load data
X = csvread("./data/dataProj.csv");
N = size(X,1);
D = zeros(N);
for m = 1:N
    for n = m+1:N
        D(m,n) = norm(X(m,:)-X(n,:),2);
        D(n,m) = D(m,n);
    end
end

%% Compute cMDS solution
Q = -(1/2)*D.^2; % Define Q
```

```
H = eye(N,N) - (1/N)*ones(N,N); % Compute centering matrix
B = H*Q*H; % Centered distances
[eigVec,eigVal] = eig(B); % Computeeigen values and eigenvectors
[eigValSorted, inds] = sort(diag(eigVal)); % Sort eigenvalues
inds = flipud(inds);
eigValSorted = flipud(eigValSorted);
eigVec = eigVec(:,inds); % Get corresponding eigenvectors
eigVal = diag(eigValSorted);
Y = eigVec(:,1:k)*diag(sqrt(eigValSorted(1:k)));

%% Plot
figure('units','normalized','outerposition',[0 0 1 1]);
scatter(Y(:,1),Y(:,2),100,'o','b','LineWidth',1,'MarkerFaceColor','flat');
hold on;
set(gca,'FontSize',35);
ax = gca;
ax.XGrid = 'on';
ax.YGrid = 'on';
title(sprintf("cMDS | Dataset task 4 | k = %d",k));
saveas(gcf,sprintf("./data/task4_extra_sol.png"));
hold off;
```

There are a few facts that are interesting to point out. First, as expected, the solution depicted in Fig. 44 is similar to the two shown in Figs. 42 and 43, solved using the LM method. It corresponds, in fact, to a rotation and centering translation of the two previous solutions. Second, even though the LM method was optimized to be computationally efficient, and the solution presented in this subsection requires eigenvalue decomposition of a large matrix, it was possible to conclude that the latter takes significantly less time to be found. The iterative nature of the LM algorithm does not allow for a time efficient computation compared to eigenvalue decomposition. Third, in the proof of the algorithm implemented to solve the cMDS problem [Jac19], it is also evident the presence of additional translational and rotational degrees of freedom, which confirms the analysis undergone in the previous subsection.

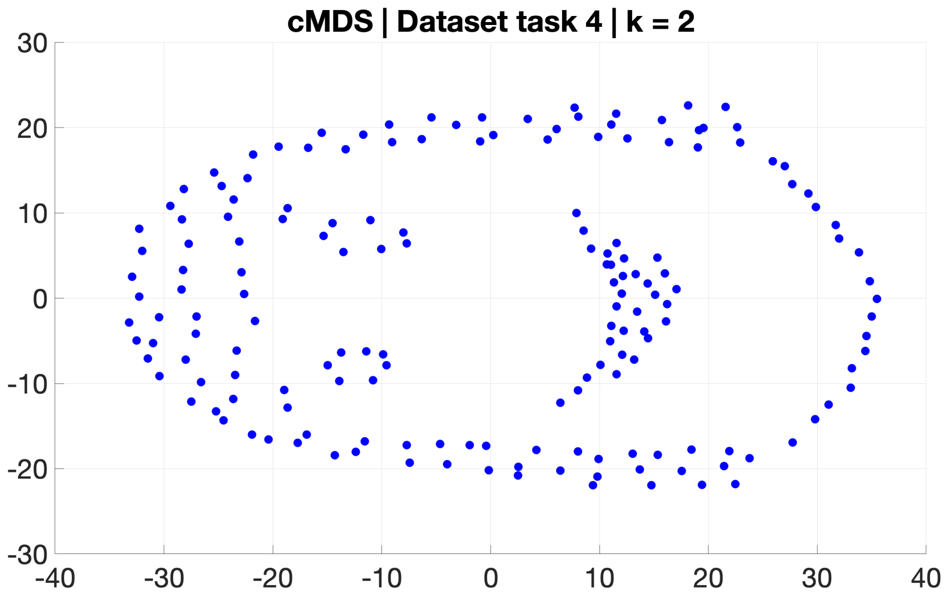


Figure 44: Solution to cMDS for the dataset of task 4 and $k = 2$

References

[CC08] Michael AA Cox and Trevor F Cox. Multidimensional scaling. In *Handbook of data visualization*, pages 315–347. Springer, 2008.

[Jac19] David Jacobs. *Multidimensional Scaling: More complete proof and some insights not mentioned in class.* Department of Computer Science and UMIACS, University of Maryland, 2019.

[Seb09] George AF Seber. *Multivariate observations*, volume 252. John Wiley & Sons, 2009.