Optimization and Algorithms
Project report

Group 24

Duarte Honrado 90058, Joao Maria Janeiro 90105, and Pedro Guerreiro 90161

Contents

1 Part 1

The goal of this part was to give control signals to a robot, in order for him to pass as near as possible to given intermediate points along its journey. To accomplish this 3 different regularizers were used, ℓ_2^2, ℓ_2 and ℓ_1, that were multiplied by our control signal, this way its influence over the path could be controled.

The following code creates the necessary setup and calls all tasks:

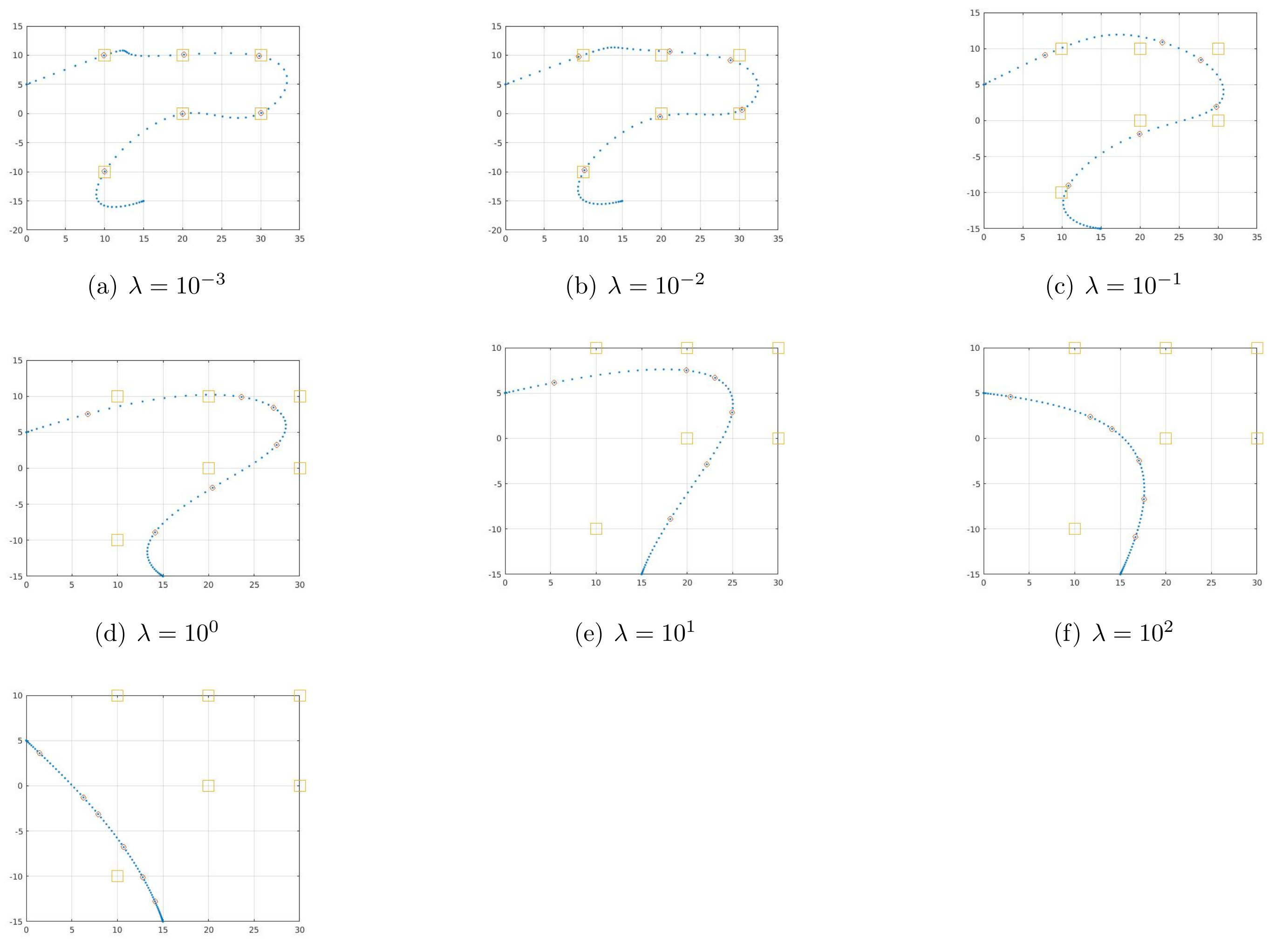
```
A = [
        1 0 0.1 0;
        0 1 0 0.1;
        0 0 0.9 0;
        0 0 0 0.9
    ];
B = [
        0 0;
        0 0;
        0.1 0;
        0 0.1
    ];

T = 80;
p_initial = [0, 5];
p_final = [15, -15];
K = 6;
w = [
        10 10;
        20 10;
        30 10;
        30 0;
        20 0;
        10 -10
    ];
tau = [10 25 30 40 50 60] + 1;
Umax = 100;
t = 1:1:T;
E = [
        1 0 0 0;
        0 1 0 0
    ];

task1(A, B, T, p_initial, p_final, w, tau, Umax, E, K, 10^(3));
task2(A, B, T, p_initial, p_final, w, tau, Umax, E, K, 10^(-1));
task3(A, B, T, p_initial, p_final, w, tau, Umax, E, K, 10^(-1));

K = 5;
tk = [0 1 1.5 3 4.5];
ck = [
    0.6332, -3.2012;
    -0.0054, -1.7104;
    2.3322, -0.7620;
    4.4526 3.1001;
    6.1752, 4.2391
];
```

```
Rk = [2.2727 0.7281 1.3851 1.8191 1.0895];
x_star = [6 10];
t_star = 8;
task5(tk, ck, Rk, x_star, t_star, K);
task6(tk, ck, Rk, t_star, K);
```

1.1 Task 1

a) Ploting the optimal positions of the robot from t=0 to t=T

In Figure 1 we plot the optimal positions of the robot from t=0 to t=T, the target positions as well as the robot's position at the appointed times τ_k for different values of λ while using ℓ_2^2 regularizer in the cost function to penalize deviations from the wishes (transfer and bounded control).



(a) $\lambda = 10^{-3}$ (b) $\lambda = 10^{-2}$ (c) $\lambda = 10^{-1}$

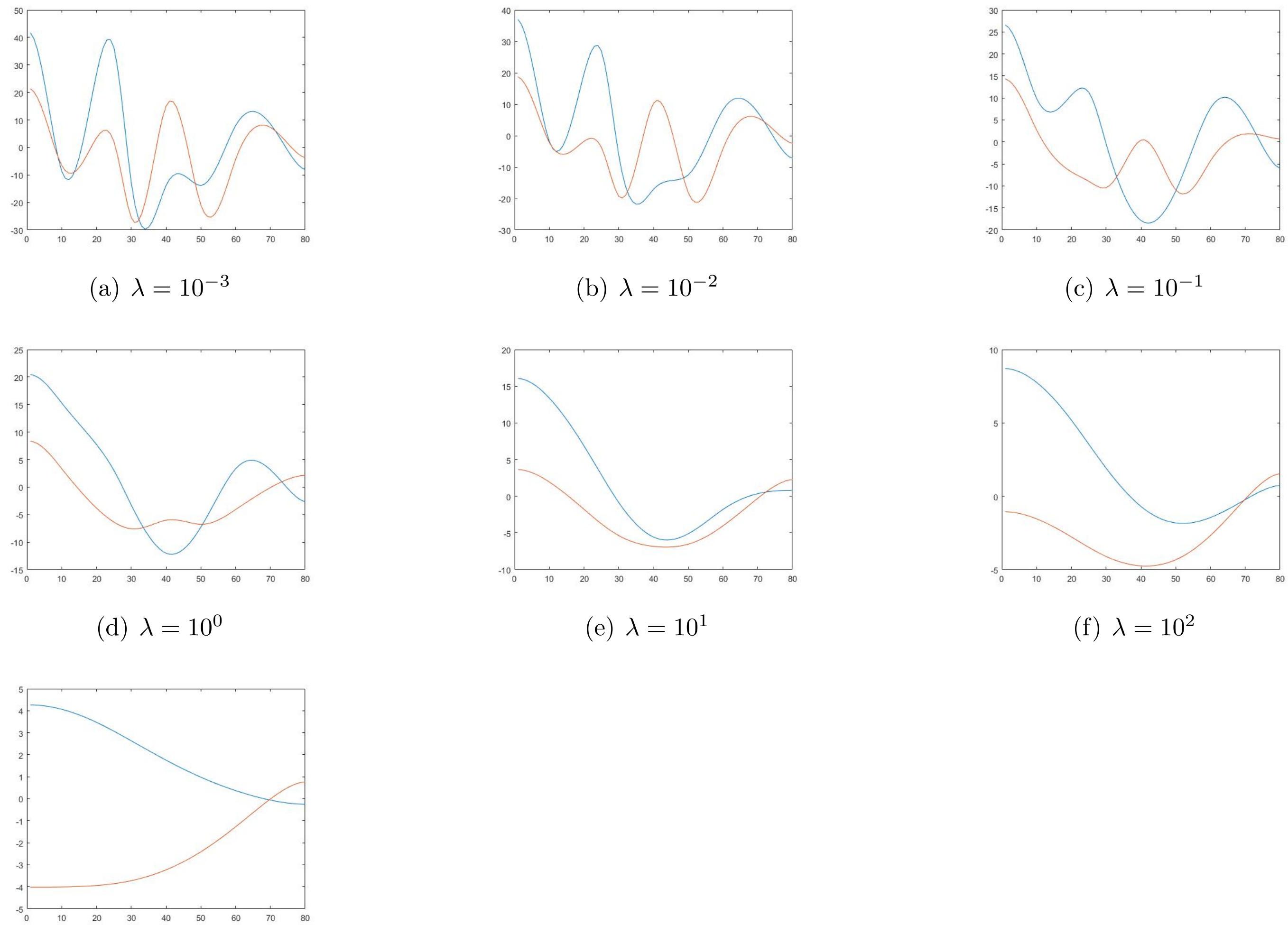
(d) $\lambda = 10^{0}$ (e) $\lambda = 10^{1}$ (f) $\lambda = 10^{2}$

(g) $\lambda = 10^{3}$

Figure 1: Optimal positions of the robot from t=0 to t=T for different values of λ while using ℓ_2^2 regularizer

b) Ploting the optimal control signal $u(t)$, from t=0 to t=T

In Figure 2 we plot the the optimal positions of the robot from t=0 to t=T,the target positions as well as the robot's position at the appointed times τ_k for $\lambda = 10^{-1}$ using the ℓ_2 regularizer.



(a) $\lambda = 10^{-3}$ (b) $\lambda = 10^{-2}$ (c) $\lambda = 10^{-1}$

(d) $\lambda = 10^{0}$ (e) $\lambda = 10^{1}$ (f) $\lambda = 10^{2}$

(g) $\lambda = 10^{3}$

Figure 2: Optimal control signal from t=0 to t=T for different values of λ while using ℓ_2^2 regularizer

c) Reporting the number of times the optimal signal changes, from t=1 to t=T

Every time the inequality 1 was true, a change in the optimal control signal was reported. So, the table 1 was filled with the number of times this happened, from $t = 0$ to $t = T$, for different values of λ.

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

Table 1: Number of changes in the optimal control signal from t=*1* to t=*T-1*

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
79	79	79	79	79	79	79

d) Reporting the mean deviation from the waypoints

By using the equation 2, it is possible to determine the mean deviations from the waypoints, shown in table 2, for different values of λ.

$$\frac{1}{K}\sum_{k=1}^{K}||Ex(\tau_k) - \omega_k||_2 \tag{2}$$

Table 2: Mean deviation from waypoints

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
0.1257	0.8242	2.1958	3.6826	5.6317	10.9042	15.3304

e) Matlab Code

To implement all the questions in task1, the following code was developed:

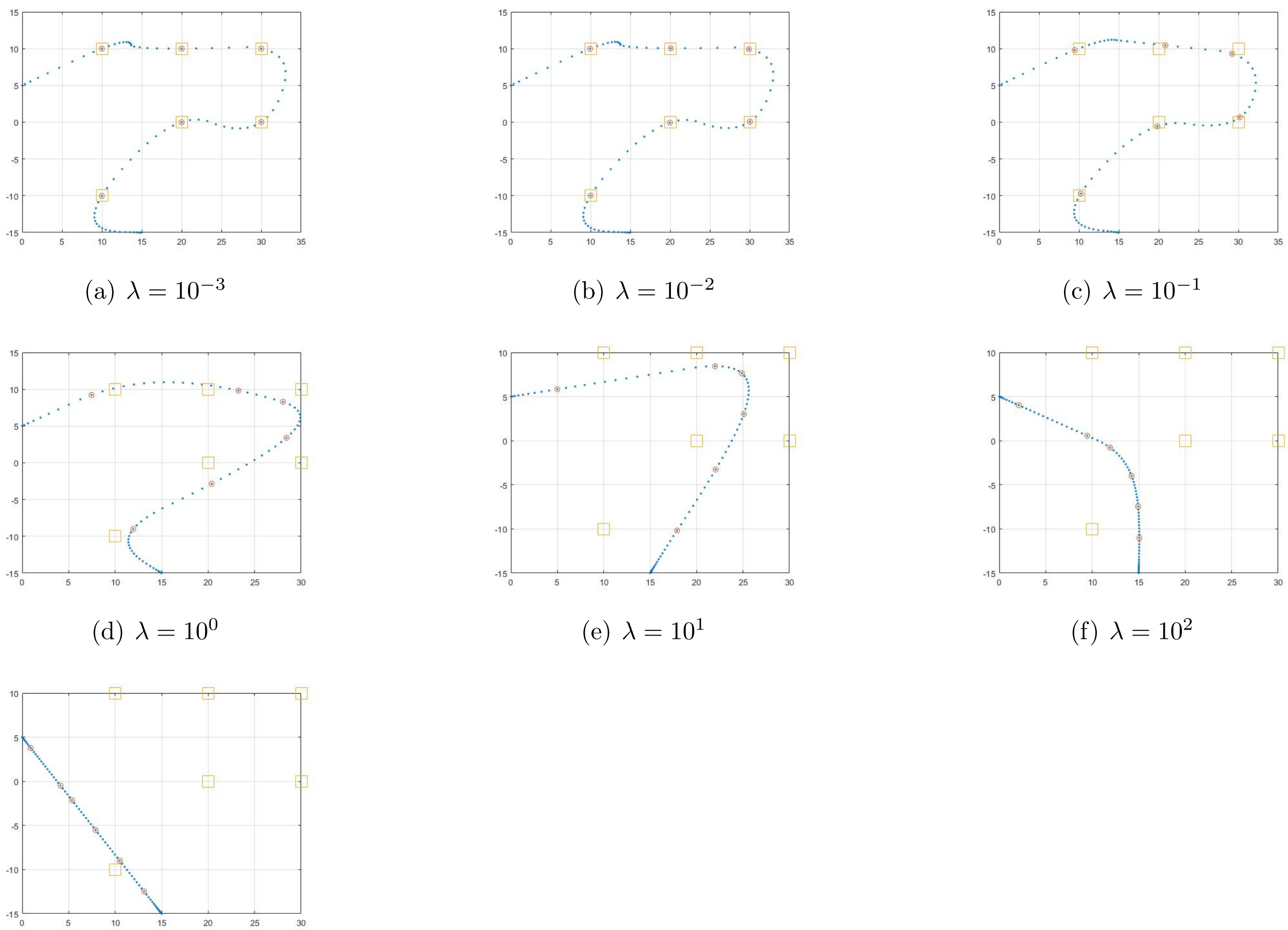
```
function task1(A, B, T, p_initial, p_final, w, tau, Umax, E, K, lambda)

cvx_begin quiet
    variable x(4, T+1)
    variable u(2,T)
    t = 2:1:T;

    first_term = sum(square_pos(norms(E*x(:,tau) - w', 2, 1)));
    second_term = sum(square_pos(norms(u(:, t) - u(:, t-1), 2, 1)));

    minimize(first_term + lambda * second_term);

    subject to
        x(1, 1) == p_initial(1);
        x(2, 1) == p_initial(2);
        x(3, 1) == 0;
        x(4, 1) == 0;
        x(1, T+1) == p_final(1);
        x(2, T+1) == p_final(2);
        x(3, T+1) == 0;
        x(4, T+1) == 0;
        norms(u, 2, 1) <= Umax;
        x(:, 2:T+1) == A*x(:, 1:T) + B*u(:, 1:T);

cvx_end;


% a)
plot(x(1,:), x(2,:), '.');
grid on;
hold on
plot(x(1,tau), x(2,tau), 'o');
hold on
plot(w(:, 1), w(:, 2), 's', 'MarkerSize',17);
```

```
hold off

% b)
i = 1:1:T;
plot(i, u(1,:), '-');
hold on
plot(i, u(2,:), '-');

% c)
counter = 0;
for j=2 : T
     if norms(u(:,j) - u(:,j-1), 2, 1) > 10^(-4)
        counter = counter + 1;
    end
end
counter

% d)
sum(norms(E*x(:,tau) - w', 2, 1)) / K

end
```

1.2 Task 2

a) Ploting the optimal positions of the robot from t=0 to t=T

In Figure 3 we plot the optimal positions of the robot from t=0 to t=T, the target positions as well as the robot's position at the appointed times τ_k for different values of λ while using ℓ_2 regularizer in the cost function to penalize deviations from the wishes (transfer and bounded control).



(a) $\lambda = 10^{-3}$ (b) $\lambda = 10^{-2}$ (c) $\lambda = 10^{-1}$

(d) $\lambda = 10^{0}$ (e) $\lambda = 10^{1}$ (f) $\lambda = 10^{2}$

(g) $\lambda = 10^{3}$

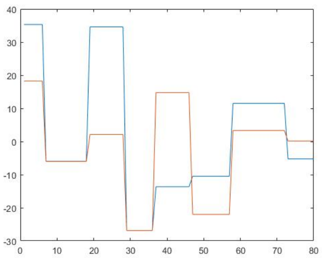
Figure 3: Optimal positions of the robot from t=0 to t=T for different values of λ while using ℓ_2 regularizer

b) Ploting the optimal control signal $u(t)$, from t=0 to t=T-1

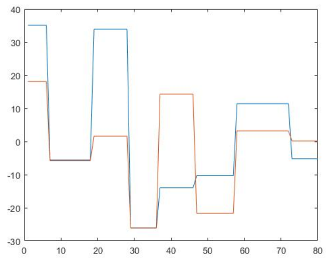
In Figure 4 we plot the the optimal positions of the robot from t=0 to t=T 1,the target positions as well as the robot's position at the appointed times τ_k for $\lambda = 10^{-1}$ using the ℓ_2 regularizer.

c) Reporting the number of times the optimal signal changes, from t=1 to t=T-1

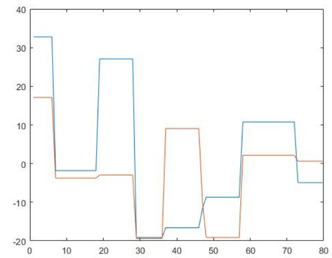
Every time the inequality 1 was true, a change in the optimal control signal was reported. So, the table 3 was filled with the number of times this happened, from t=0 to t=T-1, for different values of λ.



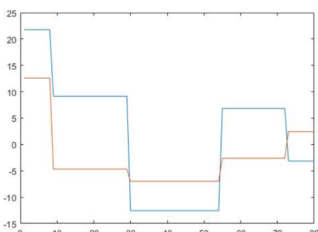
(a) $\lambda = 10^{-3}$



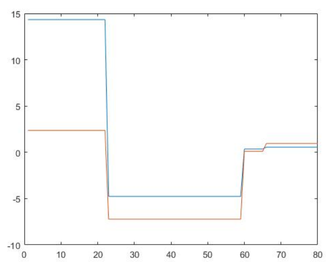
(b) $\lambda = 10^{-2}$



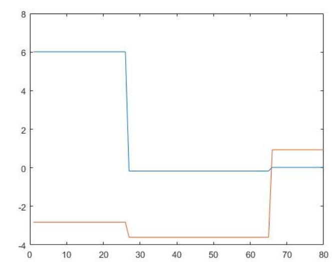
(c) $\lambda = 10^{-1}$



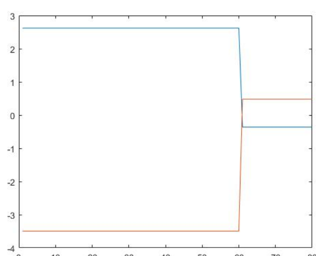
(d) $\lambda = 10^{0}$



(e) $\lambda = 10^{1}$



(f) $\lambda = 10^{2}$



(g) $\lambda = 10^{3}$

Figure 4: Optimal control signal from t=0 to t=T for different values of λ while using ℓ_2 regularizer

Table 3: Number of changes in the optimal control signal from t=*1* to t=*T-1*

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
7	7	8	4	3	2	1

d) Reporting the mean deviation from the waypoints

By using the equation 3, it is possible to determine the mean deviations from the waypoints, shown in table 4, for different values of λ.

$$\frac{1}{K}\sum_{k=1}^{K} ||Ex(\tau_k) - \omega_k||_2 \tag{3}$$

Table 4: Mean deviation from waypoints

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
0.0075	0.0747	0.7021	2.8876	5.3689	12.5914	16.2266

e) Matlab Code

To implement all the questions in task2, the following code was developed:

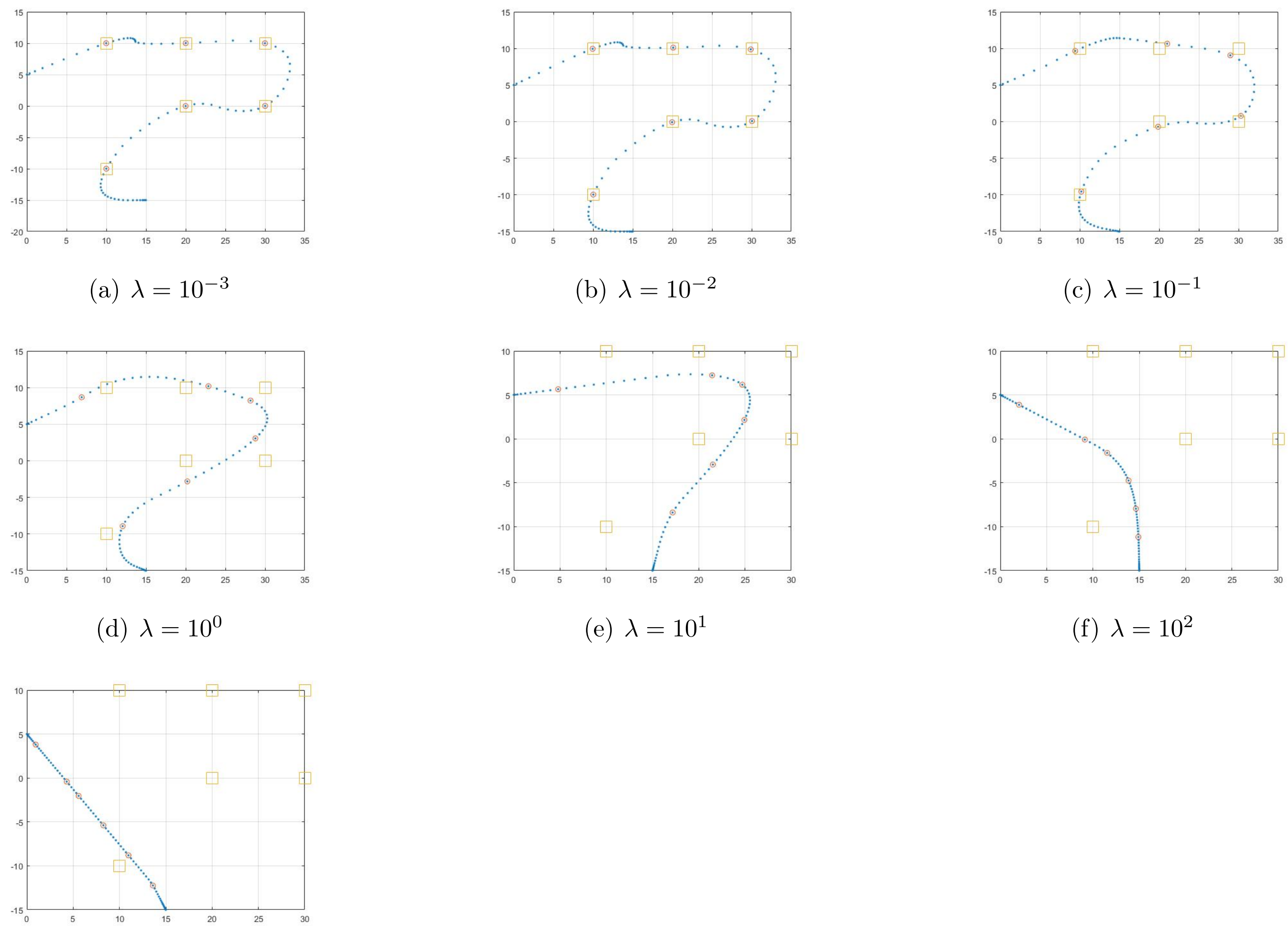
```
function task2(A, B, T, p_initial, p_final, w, tau, Umax, E, K, lambda)

cvx_begin quiet
    variable x(4, T+1)
    variable u(2,T)
    t = 2:1:T;

    first_term = sum(square_pos(norms(E*x(:,tau) - w', 2, 1)));
    second_term = sum(norms(u(:, t) - u(:, t-1), 2, 1));

    minimize(first_term + lambda * second_term);

    subject to
        x(1, 1) == p_initial(1);
        x(2, 1) == p_initial(2);
        x(3, 1) == 0;
        x(4, 1) == 0;
        x(1, T+1) == p_final(1);
        x(2, T+1) == p_final(2);
        x(3, T+1) == 0;
        x(4, T+1) == 0;
        norms(u, 2, 1) ≤ Umax;
        x(:, 2:T+1) == A*x(:, 1:T) + B*u(:, 1:T);

cvx_end;


% a)
```

```
figure()
plot(x(1,:), x(2,:), '.');
grid on;
hold on
plot(x(1,tau), x(2,tau), 'o');
hold on
plot(w(:, 1), w(:, 2), 's', 'MarkerSize',17);
hold off

% b)
figure()
i = 1:1:T;
plot(i, u(1,:), '-');
hold on
plot(i, u(2,:), '-');

% c)
counter = 0;
for j=2 : T
     if norms(u(:,j) - u(:,j-1), 2, 1) > 10^(-4)
        counter = counter + 1;
    end
end
counter

% d)
sum(norms(E*x(:,tau) - w', 2, 1)) / K

end
```

1.3 Task 3

a) Ploting the optimal positions of the robot from t=0 to t=T

In Figure 5 we plot the optimal positions of the robot from t=0 to t=T, the target positions as well as the robot's position at the appointed times τ_k for different values of λ while using ℓ_1 regularizer in the cost function to penalize deviations from the wishes (transfer and bounded control).



(a) $\lambda = 10^{-3}$ (b) $\lambda = 10^{-2}$ (c) $\lambda = 10^{-1}$

(d) $\lambda = 10^{0}$ (e) $\lambda = 10^{1}$ (f) $\lambda = 10^{2}$

(g) $\lambda = 10^{3}$

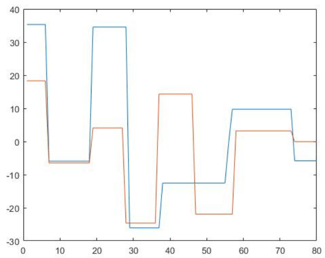
Figure 5: Optimal positions of the robot from t=0 to t=T for different values of λ while using ℓ_1 regularizer

b) Ploting the optimal control signal $u(t)$, from t=0 to t=T-1

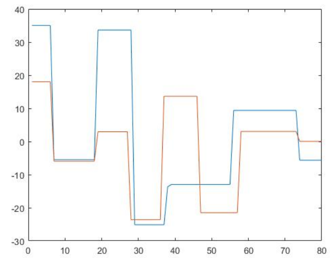
In Figure 6 we plot the the optimal positions of the robot from t=0 to t=T 1,the target positions as well as the robot's position at the appointed times τ_k for $\lambda = 10^{-1}$ using the ℓ_1 regularizer.

c) Reporting the number of times the optimal signal changes, from t=1 to t=T-1

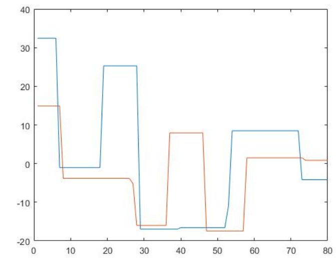
Every time the inequality 4 was true, a change in the optimal control signal was reported. So, the table 5 was filled with the number of times this happened, from t=0 to t=T-1, for different values of λ.



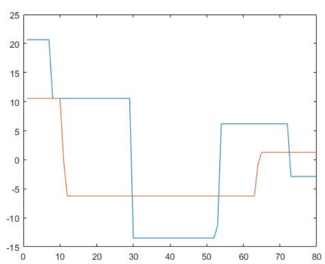
(a) $\lambda = 10^{-3}$



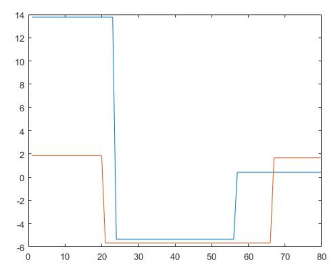
(b) $\lambda = 10^{-2}$



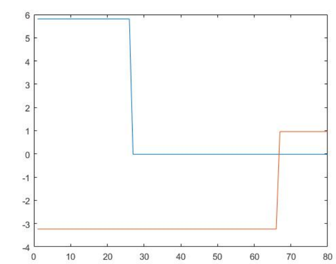
(c) $\lambda = 10^{-1}$



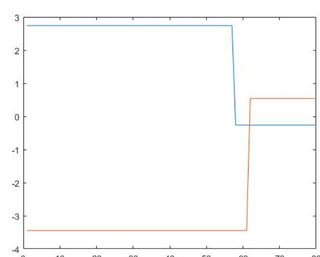
(d) $\lambda = 10^{0}$



(e) $\lambda = 10^{1}$



(f) $\lambda = 10^{2}$



(g) $\lambda = 10^{3}$

Figure 6: Optimal control signal from t=0 to t=T for different values of λ while using ℓ_1 regularizer

$$||u(t) - u(t-1)||_1 > 10^{-4} \tag{4}$$

Table 5: Number of changes in the optimal control signal from *t=1* to *t=T-1*

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
11	11	14	9	4	2	2

d) Reporting the mean deviation from the waypoints

By using the equation 5, it is possible to determine the mean deviations from the waypoints, shown in table 6, for different values of λ.

$$\frac{1}{K}\sum_{k=1}^{K} ||Ex(\tau_k) - \omega_k||_1 \tag{5}$$

Table 6: Mean deviation from waypoints

$\lambda = 10^{-3}$	$\lambda = 10^{-2}$	$\lambda = 10^{-1}$	$\lambda = 10^{0}$	$\lambda = 10^{1}$	$\lambda = 10^{2}$	$\lambda = 10^{3}$
0.0107	0.1055	0.8863	2.8732	5.4361	13.0273	16.0463

e) Matlab Code

To implement all the questions in task3 the following code was developed:

```
function task3(A, B, T, p_initial, p_final, w, tau, Umax, E, K, lambda)

cvx_begin quiet
    variable x(4, T+1)
    variable u(2,T)
    t = 2:1:T;

    first_term = sum(square_pos(norms(E*x(:,tau) - w', 2, 1)));
    second_term = sum(norms(u(:, t) - u(:, t-1), 1, 1));

    minimize(first_term + lambda * second_term);

    subject to
        x(1, 1) == p_initial(1);
        x(2, 1) == p_initial(2);
        x(3, 1) == 0;
        x(4, 1) == 0;
        x(1, T+1) == p_final(1);
        x(2, T+1) == p_final(2);
        x(3, T+1) == 0;
        x(4, T+1) == 0;
        norms(u, 2, 1) <= Umax;
        x(:, 2:T+1) == A*x(:, 1:T) + B*u(:, 1:T);
```

```
cvx_end;


% a)
figure()
plot(x(1,:), x(2,:), '.');
grid on;
hold on
plot(x(1,tau), x(2,tau), 'o');
hold on
plot(w(:, 1), w(:, 2), 's', 'MarkerSize',17);
hold off

% b)
figure()
i = 1:1:T;
plot(i, u(1,:), '-');
hold on
plot(i, u(2,:), '-');

% c)
counter = 0;
for j=2 : T
     if norms(u(:,j) - u(:,j-1), 2, 1) > 10^(-4)
        counter = counter + 1;
    end
end
counter

% d)
sum(norms(E*x(:,tau) - w', 2, 1)) / K

end
```

1.4 Task 4

Our cost function is defined as

$$\sum_{k=1}^{K} ||Ex(\tau_k - w_k)||_2^2 + \lambda \sum_{t=1}^{T-1} ||u(t) - u(t-1)||_a^p \tag{6}$$

In equation 6, the second term is denominated regularizer, where p is the ℓ norm function that is used. In task 1, $p = a = 2$, this defined the ℓ_2^2 regularizer. In task 2, $p = 2$ and $a = 1$, in this case we used the ℓ_2 regularizer. Finally, in task 3, $p = a = 1$, the ℓ_1 regularizer.

The first term is used to penalize deviations between the positions of the robot and the target positions, the waypoints. The second term is the regularizer which is used to penalize changes of the control signal at time t from its previous value at time t-1.

λ is used to give more or less weight to the second term of the equation, this means that when λ is increased, the robot will give more attention to the variations in the control signal, maybe even more than reaching the waypoints in the desired time. This means that by increasing λ, the mean deviation from the waypoints will also increase. Conclusion that can be easily verified by observing Table 1, Table 3 and Table 5.

When λ is set as a big value, its influence is notorious, making the control signal very constant, which makes it quite difficult for the robot to cross all the waypoints, which can be verified in Figure 1, Figure 3 and Figure 5. For lower values of λ, the robot fits the waypoints better, at the cost of a more complex control signal which can verified by looking at 2, Figure 4 and Figure 6.

The optimizer will attempt to minimize the values of the regularizer, which is, again, the difference between consecutive values. When the ℓ_2^2 regularizer is used, the minimum reached is never actually zero, which is the reason why the control signal changes for every instant, check Table 1. With the ℓ_2 and the ℓ_1 norms, some values are actually minimized to zero, originating a relatively small number of changes in the control signal, check Figure 4 and Figure 6, which are piecewise constant signals as desired ($u(t) = u(t-1)$ for most values of t).

The derivative at a point of the square of the Euclidean norm decreases when the algorithm is converging to zero. So, when using the ℓ_2^2 regularizer, the algorithm will never reach zero. However, comparing to the other regularizers, this one is faster and more efficient to compute.

When the ℓ_2 regularizer is used, the optimizer can minimize the second term to actual zero. The derivative is constant and, during the minimization, the algorithm decrements always the same value. The minimization can easily reach values below the threshold which explains the small number of changes.

When the ℓ_1 regularizer is used, we have a subtle increase in the number of changes in the control signal. This regularizer can also minimize and actually reach zero but, this increase in number of changes is due to the fact that using *norm1*, both x and y of each signal u(t) are minimized individually. So, even if x is maintained, if y changes, this leads to a change in the control signal.

The ℓ_1 regularizer is not as computational efficient as the rest because the *norm1* can have more than one solution, instead of the Euclidean norm that has a single solution.

The first wish (wish of transfer) and the second wish (bounded control) are always respected because they are the constraints of the optimization. The third wish (passing close to the waypoints) is highly dependent on the value of λ. The lower the value, the better it fits the waypoints. The forth wish (simpler control) is only satisfied for norms ℓ_1 and ℓ_2 and the higher the λ, the better it is satisfied.

1.5 Task 5

Let's take p(t) as the position of the target t, p_0 as the initial position and v as its velocity, we have:

$$p(t) = p_0 + tv \tag{7}$$

We want to know how close a target can be to a given critical point x^* at a given future time t^*. So our goal is to minimize the following equation:

$$\sum_{k=1}^{K} ||(p_{0k} + t^* v_k) - x^*||_2^2 \tag{8}$$

The constraint of this problem is that at every instance t_k the target must be inside the circle (c_k, R_k). For this problem we obtained $p_0 = [-0.5368; -3.2715]$ and $v = [1.2497; 1.6803]$.

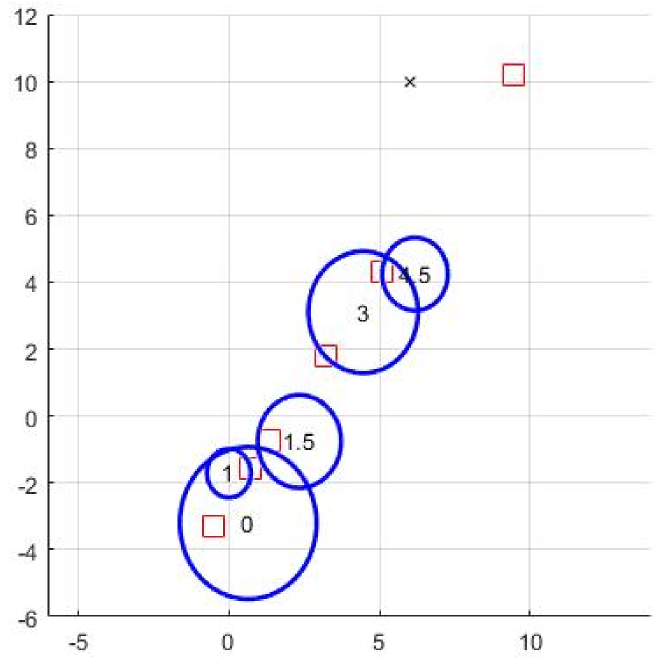


Figure 7: The red squares inside the circles are the positions at times t_k. The red square in the top right is the position of that target at time t^*

```
function task5(tk, ck, Rk, x_star, t_star, K)

cvx_begin quiet
    variable p0(2, 1)
    variable v(2, 1)
    k = 1:1:K;

    first_term = sum(square_pos(norms(p0 +t_star*v - x_star', 2, 1)));

    minimize(first_term);

    subject to
        norms([p0 p0 p0 p0 p0] + v*tk - ck', 2, 1) ≤ Rk;

cvx_end;

p0
v

figure
grid on


xlim([-6 14])
ylim([-6 12])

axis square

```

```
hold on

plot(p0(1) + v(1)*tk, p0(2) + v(2)*tk, 's', 'MarkerSize',12, 'Color', 'r');

hold on

plot(x_star(1),x_star(2),'x', 'Color', 'k');

hold on

viscircles(ck,Rk,'Color','b');
text(ck(:,1),ck(:,2),strcat(string(tk)), 'HorizontalAlignment', 'center',...
    'FontName', 'Arial',...
    'FontSize', 10);   % plots the text inside each circle
hold on

plot(p0(1) + v(1)*t_star, p0(2) + v(2)*t_star, 's', 'MarkerSize',12,...
    'Color', 'r');

hold on



end
```

1.6 Task 6

The goal of this task was to find the smallest rectangle that contains all possible positions for a target that moves as in (7) and that was in disk $D(C_k, R_k)$ at time t_k.

To find the edges of the rectangle 4 optimization problems were solved. To find the value of a_1 we solved the the optimization problem:

$$\begin{aligned} \min_{p_x,t} \quad & p_x + t^* \times v_x \\ \text{s.t.} \quad & ||p + v \times t_k - c_k|| \leq R_k \end{aligned} \tag{9}$$

To find a_2 the following optimization problem was solved:

$$\begin{aligned} \max_{p_x,t} \quad & p_x + t^* \times v_x \\ \text{s.t.} \quad & ||p + v \times t_k - c_k|| \leq R_k \end{aligned} \tag{10}$$

To find $b1$ we needed to solve:

$$\begin{aligned} \min_{p_y,t} \quad & p_y + t^* \times v_y \\ \text{s.t.} \quad & ||p + v \times t_k - c_k|| \leq R_k \end{aligned} \tag{11}$$

To find $b2$ we needed to solve:

$$\begin{aligned} \max_{p_y,t} \quad & p_y + t^* \times v_y \\ \text{s.t.} \quad & ||p + v \times t_k - c_k|| \leq R_k \end{aligned} \tag{12}$$

Table 7: Chosen Values for a_1, a_2, b_1 and b_2

a_1	a_2	b_1	b_2
9.4582	14.1902	7.4763	12.9690

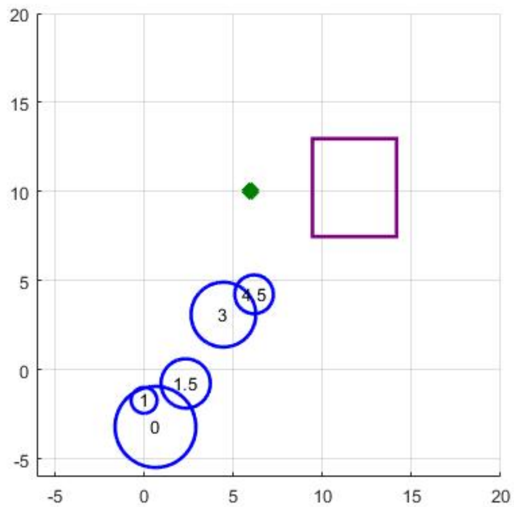


Figure 8: The purple square near the point x is the smallest rectangle that contains all possible positions for a target that moves as in the previous task at time $t^* = 8$.

a) Matlab Code

```
function task6(tk, ck, Rk,x_star, t_star, K)

cvx_begin quiet
    variable p0(2, 1)
    variable v(2, 1)
    k = 1:1:K;

    minimize(p0(1) + t_star*v(1));

    subject to
        norms([p0 p0 p0 p0 p0] + v*tk - ck', 2, 1) ≤ Rk;

cvx_end;

a1 = p0(1) + t_star*v(1)

cvx_begin quiet
    variable p0(2, 1)
    variable v(2, 1)
    k = 1:1:K;

    minimize(p0(2) + t_star*v(2));

    subject to
        norms([p0 p0 p0 p0 p0] + v*tk - ck', 2, 1) ≤ Rk;
```

```

cvx_end;

b1 = p0(2) + t_star*v(2)

cvx_begin quiet
    variable p0(2, 1)
    variable v(2, 1)
    k = 1:1:K;

    maximize(p0(1) + t_star*v(1));

    subject to
        norms([p0 p0 p0 p0 p0] + v*tk - ck', 2, 1) ≤ Rk;

cvx_end;

a2 = p0(1) + t_star*v(1)

cvx_begin quiet
    variable p0(2, 1)
    variable v(2, 1)
    k = 1:1:K;

    maximize(p0(2) + t_star*v(2));

    subject to
        norms([p0 p0 p0 p0 p0] + v*tk - ck', 2, 1) ≤ Rk;

cvx_end;

b2 = p0(2) + t_star*v(2)

figure
grid on


xlim([-6 20])
ylim([-6 20])

axis square

hold on

plot(x_star(1),x_star(2),'x', 'Color',  [0 0.5 0],'Linewidth',8);

hold on

viscircles(ck,Rk,'Color','b');
text(ck(:,1),ck(:,2),strcat(string(tk)), 'HorizontalAlignment',
        'center', 'FontName',
        'Arial', 'FontSize', 10);   % plots the text inside each circle
hold on


hold on
```

```

rectangle('Position', [a1 b1 a2-a1 b2-b1],
         'EdgeColor', [.5 0 .5], 'Linewidth',2)


end
```

2 Part 2

The goal in this part is to create a model that can do automatic prediction of a given task. A dataset is taken, a model is developed and then predictions can be done on unseen data. In order to simplify the computation of the function we want to minimize, its gradient and the hessian, a change in variables is performed. The variable s_r is defined as follows

$$s\_r = \begin{bmatrix} s \\ r \end{bmatrix} \tag{13}$$

By using this substitution the model expression for a point is given by a vector multiplication, as follows:

$$s^T x_k - r = \dot{x}_k^T s\_r$$

Then we also introduce

$$A = \begin{bmatrix} x_k \\ -1 \end{bmatrix} \tag{14}$$

We can now rewrite the optimization problem as:

$$\min_{s\_r} \quad \frac{1}{K} \sum_{k=1}^{K} (log(1 + exp(A^T s\_r) - y^T(A^T s\_r))) \tag{15}$$

These changes allow for the vectorization of the code, which will run much faster than if we were using for loops.

2.1 Task 1

We want to prove the convexity of the following equation

$$\frac{1}{K} \sum_{k=1}^{K} log(1 + e^{s^T x_k - r}) - y_k(s^T x_k - r)) \tag{16}$$

We will start by desconstructing this function into smaller, convex pieces, like shown in Figure 9.

The function g_2 is affine so now all we have to do is prove the convexity of g_1. In order to do this we will check its derivatives.

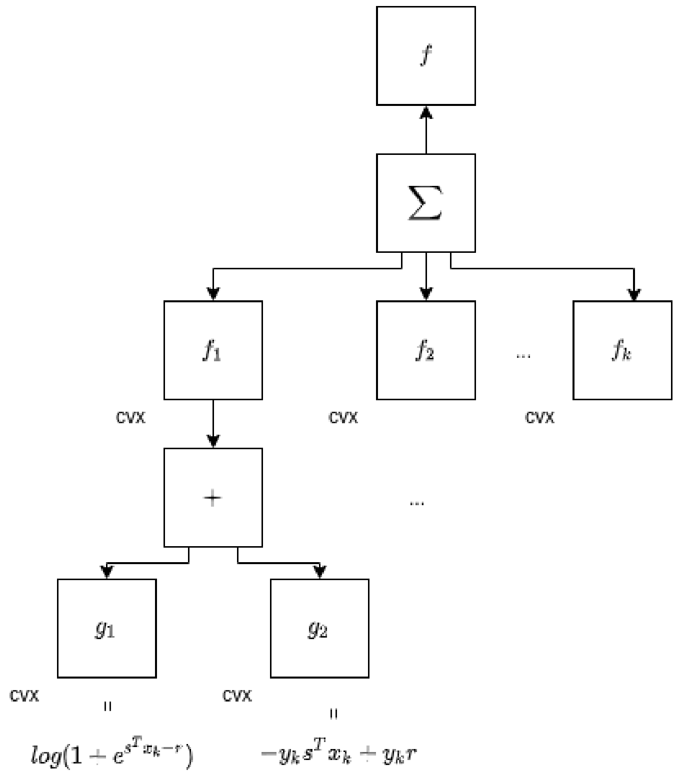


Figure 9: Recursion tree proving f convexity

$$g_1^{'} = \frac{s^T e^{s^T x_k - r}}{1 + e^{s^T x_k - r}} \tag{17}$$

$$g_1^{''} = \frac{s^2 e^{s^T x_k - r}}{(e^{s^T x_k} + e^r)^2} \tag{18}$$

A function f is said to be convex at an interval t if, for all pairs of points of $f(x)$, the line segment that connects these two points is above the $f(x)$ curve.

A twice differentiable function is convex on an interval if and only if its second derivative is non-negative. Visually speaking, a twice differentiable convex function has an upwards curve, without any bends downwards.

We can take a look at the numerator of equation 18 which is a number squared times an exponential, this is always $>= 0$ and the denominator is a squared sum which is always $>= 0$ as well.

2.2 Task 2

a) Problem Resolution

The goal of this task was to minimize the function previously given in Equation 15 by using the gradient method. The result of our algorithm is exhibited in the following figures.

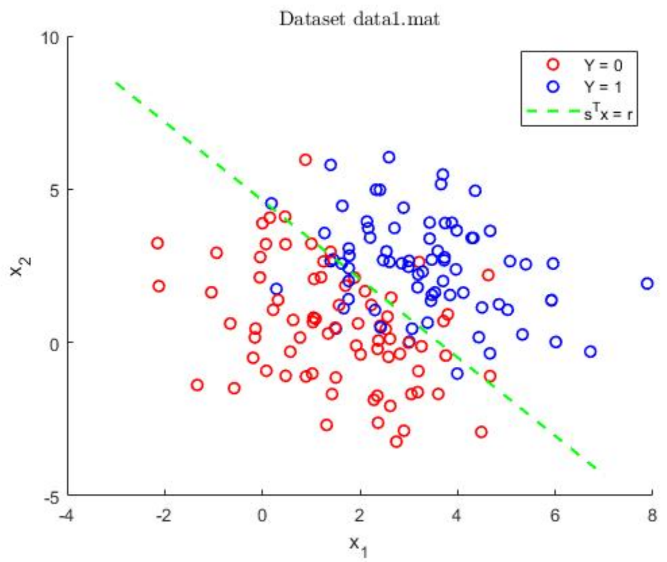


Figure 10: The dataset data1.mat with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed.

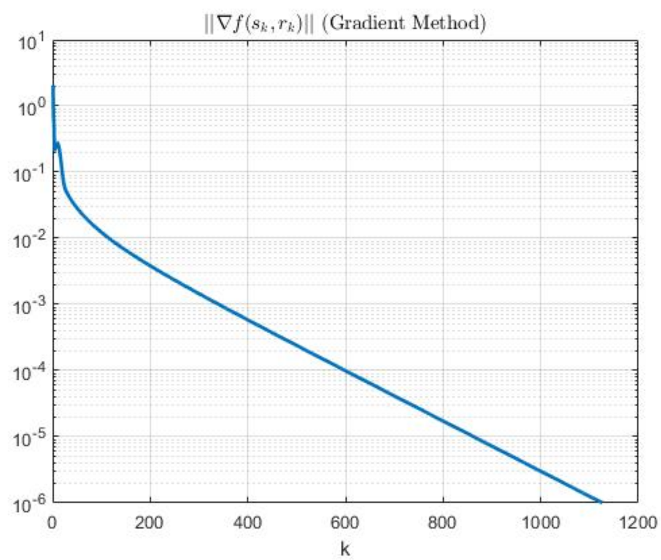


Figure 11: The norm of the gradient ($||\nabla f(s_k, r_k||$) along the iterations of the gradient method in the previous dataset.

In this case, the values retrieved for s, r and the number of iterations were:

- s = (1.3495, 1.0540)
- r = 4.8815
- Number of iterations = 1126

b) Matlab Code

```
function [f_function] = f(s_r, A ,Y,K)
    f_function = (1/K)*sum(log(1 + exp(A'*s_r)) - (Y.').*(A'*s_r));
end
```

```
function [gradient_k] = gradient_function(s_r, A ,Y,K)
    gradient_k = (1/K) * A*(exp(A'*s_r)./(1 + exp(A'*s_r)) - Y');


end
```

```
load('data1.mat');

%Amouns_rof inpus_rfeatures%
[n,K] = size(X);

%Stopping Criteria%
s0 = -ones(1,n);
r0 = 0;
epsilon = 10^(-6);
s_r0 = [s0 r0]';
A = [X; -ones(K,1)'];


%For the Backtracking Subroutine%
alpha0 = 1;
gama = 10^(-4);
beta = 0.5;
gradients=[];


s_r= s_r0;
alpha = alpha0;

while (1)
    g_k = gradient_function(s_r, A ,Y,K);
    gradients = [gradients norm(g_k)];
    if norm(g_k) < epsilon
        break;
    end
    d = -g_k;
    alpha = alpha0;
    while minimize_function(s_r+ alpha.*d, A, Y, K) ≥ ...
        minimize_function(s_r, A ,Y,K) + (gama.*g_k'*(alpha.*d))
        alpha = beta .* alpha;
    end
    s_r= s_r+ (alpha .* d);


end

```

```

s = s_r(1:length(s_r)-1)
r = s_r(length(s_r))
iterations = length(gradients)

%Plot Graph of The Norm of The Gradient
figure('NumberTitle', 'off', 'Name', 'Task_2_Norm of The Gradient');
semilogy(gradients, 'LineWidth',2);
grid on;
title('$$||\nabla f(s_{k},r_{k})||$$ (Gradient Method)','interpreter','latex')
xlabel('k')

%Plot Scatter
figure('NumberTitle', 'off', 'Name', 'Task_2_Dataset');
for i=1:K
   if Y(i) == 0
        a = scatter(X(1, i), X(2, i), [], 'red','LineWidth',1.25);
        hold on
    else
        b = scatter(X(1, i), X(2, i), [], 'blue','LineWidth',1.25);
        hold on
    end

end

x = linspace(-3,7);
y = s_r(3)/s_r(2) - x*(s_r(1)/s_r(2));
c = plot(x, y, '--y','Color','g', 'LineWidth', 1.5);
legend([a(1) b(1) c(1)], 'Y = 0', 'Y = 1', 's^Tx = r')
title('Dataset data1.mat','interpreter','latex')
xlabel('x_1');
ylabel('x_2');
```

c) Task 3

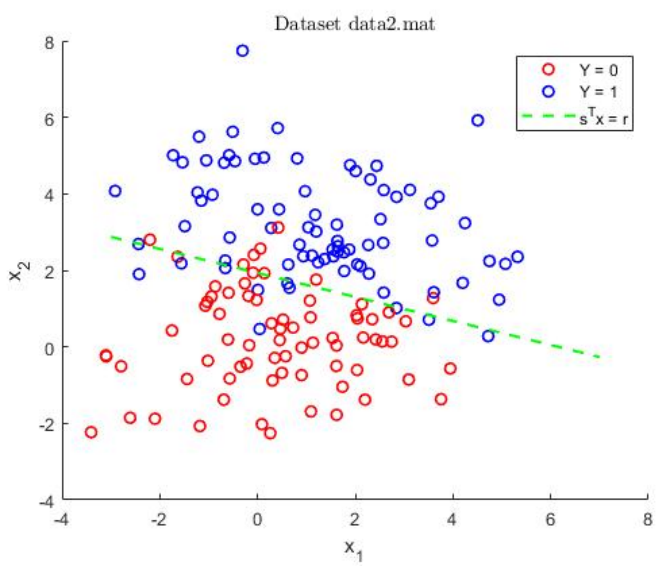


Figure 12: The dataset **data2.mat** with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed.

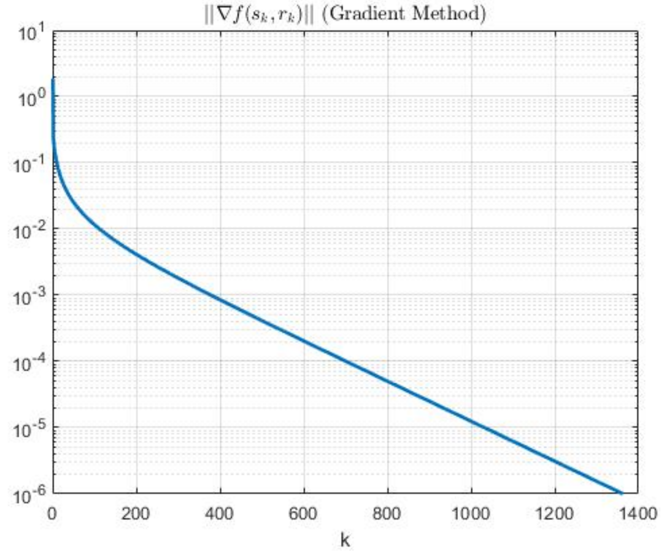


Figure 13: The norm of the gradient ($||\nabla f(s_k, r_k)||$) along the iterations of the gradient method in the previous dataset.

Redoing Task 2, now for the dataset available in **data2.mat**, the values obtained were:

- s = (0.7402, 2.3577)
- r = 4.5553
- Number of iterations = 1363

d) Task 4

For **data3.mat** we get the following results:

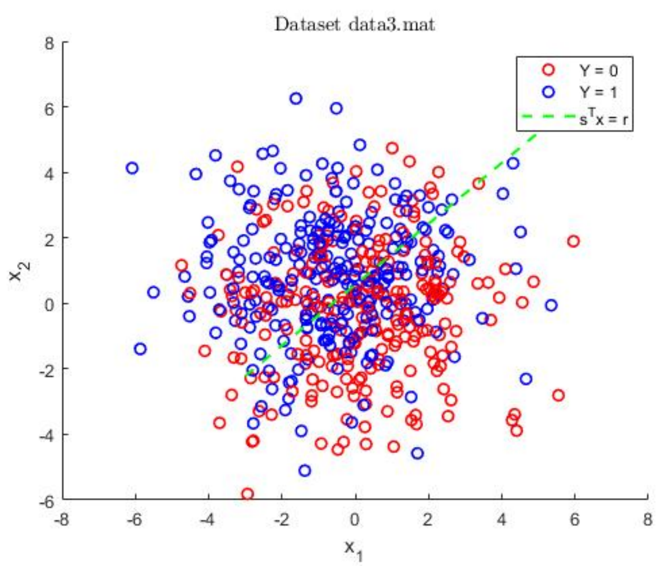


Figure 14: The dataset **data3.mat** with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed.

Which is generated from:

- $s =$ (-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)
- $r = 4.7984$
- Number of iterations = 3437

Where the norm of the gradient changes as in figure 15.

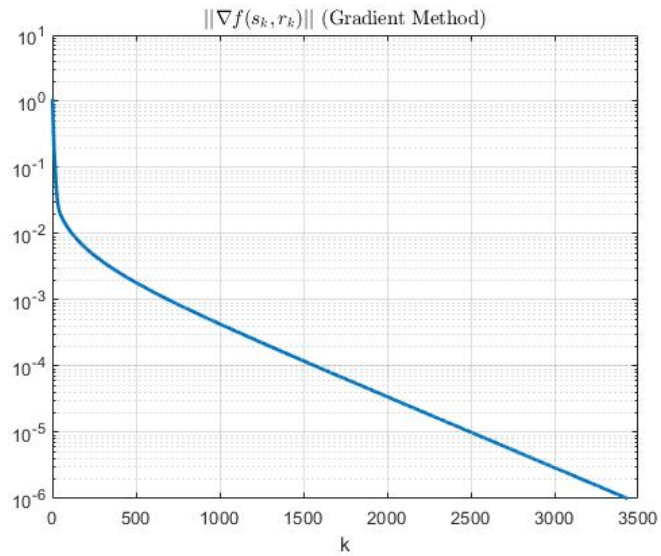


Figure 15: The norm of the gradient ($||\nabla f(s_k, r_k)||$) along the iterations of the gradient method in the data3.m

For **data4.mat** we get the following results:

- 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
- Number of iterations = 19893

Where the norm of the gradient changes as in figure 16.

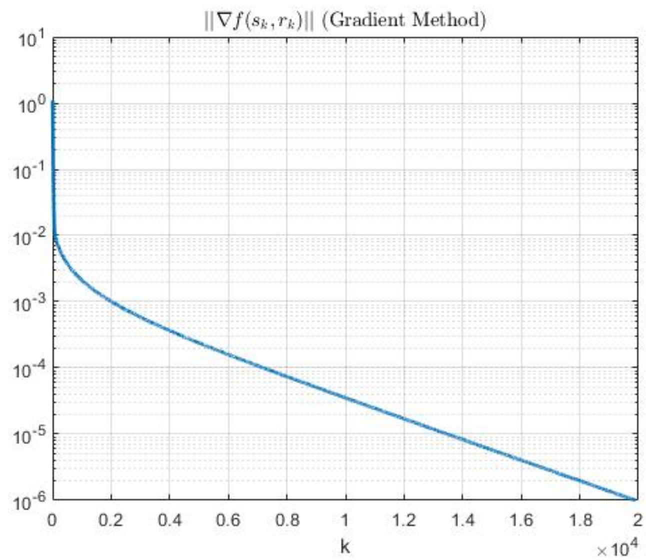


Figure 16: The norm of the gradient ($||\nabla f(s_k, r_k)||$) along the iterations of the gradient method in the data4.m

The code necessary for this part is the following:

```
load('data3.mat');


%Amount_of input_features%
[n,K] = size(X);

%Stopping Criteria%
s0 = -ones(1,n);
r0 = 0;
epsilon = 10^(-6);
s_r0 = [s0 r0]';
A = [X; -ones(K,1)'];

gradients = []

%For the Backtracking Subroutine%
alpha0 = 1;
gama = 10^(-4);
beta = 0.5;

s_r= s_r0;
alpha = alpha0;
while (1)
    g_k = gradient_function(s_r, A ,Y,K);
    gradients = [gradients norm(g_k)];
    if norm(g_k) < epsilon
        break;
    end
    d = -g_k;
    alpha = alpha0;
    while minimize_function(s_r+ alpha.*d, A, Y, K) ≥ ...
        minimize_function(s_r, A ,Y,K) + (gama.*g_k'*(alpha.*d))
        alpha = beta .* alpha;
    end
    s_r= s_r+ (alpha .* d);

end


s = s_r(1:length(s_r)-1)
r = s_r(length(s_r))
iterations = length(gradients)

%Plot Graph of The Norm of The Gradient
figure('NumberTitle', 'off', 'Name', 'Task_4_Norm of The Gradient');
semilogy(gradients, 'LineWidth',2);
grid on;
title('$$||\nabla f(s_{k},r_{k})||$$ (Gradient Method)','interpreter','latex')
xlabel('k')


figure('NumberTitle', 'off', 'Name', 'Task_4_Dataset3');
for i=1:K
```

```
    if Y(i) == 0
          a = scatter(X(1, i), X(2, i), [], 'red','LineWidth',1.25);
          hold on
     else
          b = scatter(X(1, i), X(2, i), [], 'blue','LineWidth',1.25);
          hold on
     end

end

x = linspace(-3,7);
y = s_r(3)/s_r(2) - x*(s_r(1)/s_r(2));
c = plot(x, y, '--y','Color','g', 'LineWidth', 1.5);
legend([a(1) b(1) c(1)], 'Y = 0', 'Y = 1', 's^Tx = r')
title('Dataset data3.mat','interpreter','latex')
xlabel('x_1');
ylabel('x_2');
```

2.3 Task 5

Being $\phi : \mathbb{R} \to \mathbb{R}$ a twice-differential function and $p : \mathbb{R}^3 \to \mathbb{R}$ given by:

$$p(x) = \sum_{k=1}^{K} \phi(a_k^T x), \qquad a_k, x \in \mathbb{R}^3 \tag{19}$$

a) Gradient of p

$$\nabla p(x) = \begin{bmatrix} \frac{\partial p(x)}{\partial x_1} \\ \frac{\partial p(x)}{\partial x_2} \\ \frac{\partial p(x)}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{K} \frac{\partial}{\partial x_1} \phi(a_k^T x) \\ \sum_{k=1}^{K} \frac{\partial}{\partial x_2} \phi(a_k^T x) \\ \sum_{k=1}^{K} \frac{\partial}{\partial x_3} \phi(a_k^T x) \end{bmatrix}$$

Applying the chain rule, $\frac{\partial}{\partial x_i} \phi(a_k^T x) = \dot{\phi}(a_k^T x) a_{ki}$.

$$\nabla p(x) = \begin{bmatrix} \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_{k1} \\ \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_{k2} \\ \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_{k3} \end{bmatrix} = \sum_{k=1}^{K} \dot{\phi}(a_k^T x) \begin{bmatrix} a_{k1} \\ a_{k2} \\ a_{k3} \end{bmatrix} = \sum_{k=1}^{K} \dot{\phi}(a_k^T x) a_k$$

Expressing the previous equality as a matrix multiplication:

$$\nabla p(x) = \begin{bmatrix} a_1 & a_2 & \dots & a_k \end{bmatrix} \begin{bmatrix} \dot{\phi}(a_k^T x) \\ \dot{\phi}(a_k^T x) \\ \vdots \\ \dot{\phi}(a_k^T x) \end{bmatrix} = Av \tag{20}$$

b) Hessian of p

$$\nabla^2 p(x) = H_{p(x)} = \begin{bmatrix} \sum_{k=1}^{K} \ddot{\phi}(a_k^T x) a_{k1} a_{k1} & \dots & \sum_{k=1}^{K} \ddot{\phi}(a_k^T x) a_{k1} a_{k3} \\ \dots & \dots & \dots \\ \sum_{k=1}^{K} \ddot{\phi}(a_k^T x) a_{k3} a_{k1} & \dots & \sum_{k=1}^{K} \ddot{\phi}(a_k^T x) a_{k3} a_{k3} \end{bmatrix}$$

$$\Leftrightarrow H_{p(x)} = \sum_{k=1}^{K} a_k^{\ddot{T}} x \begin{bmatrix} a_{k1}a_{k1} & a_{k1}a_{k2} & a_{k1}a_{k3} \\ a_{k2}a_{k1} & a_{k2}a_{k2} & a_{k2}a_{k3} \\ a_{k3}a_{k1} & a_{k3}a_{k2} & a_{k3}a_{k3} \end{bmatrix} = \sum_{k=1}^{K} a_k^{\ddot{T}} x a_k a_k^T = \sum_{k=1}^{K} a_k a_k^{\ddot{T}} x a_k^T$$

Expressing the previous equation using matrix multiplication:

$$H_{p(x)} = [a_1 \quad a_2 \quad \dots \quad a_k] \begin{bmatrix} \ddot{\phi}(a_1^T x) & & 0 \\ & \dots & \\ 0 & & \ddot{\phi}(a_k^T x) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_k \end{bmatrix} = ADA^T \tag{21}$$

2.4 Task 6

a) Data1.m

The result of the Newton method for **data1.m** is present in figure

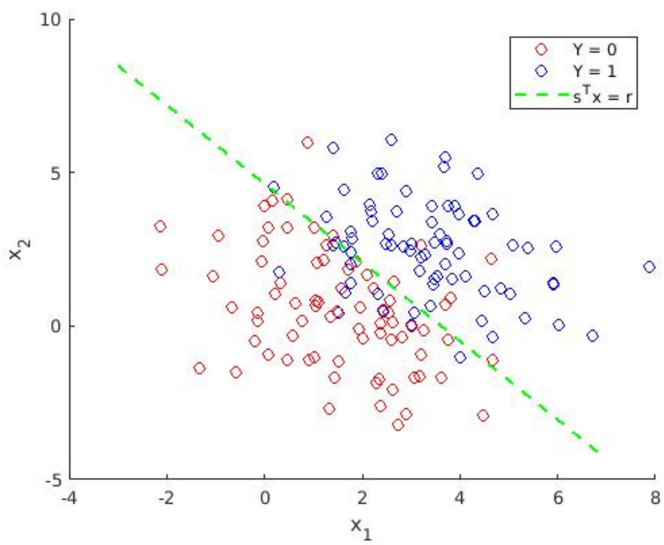


Figure 17: The dataset **data1.mat** with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed with the Newton method results.

The results are

- s = (1.3496, 1.0540)
- r = 4.8817
- Number of iterations = 8

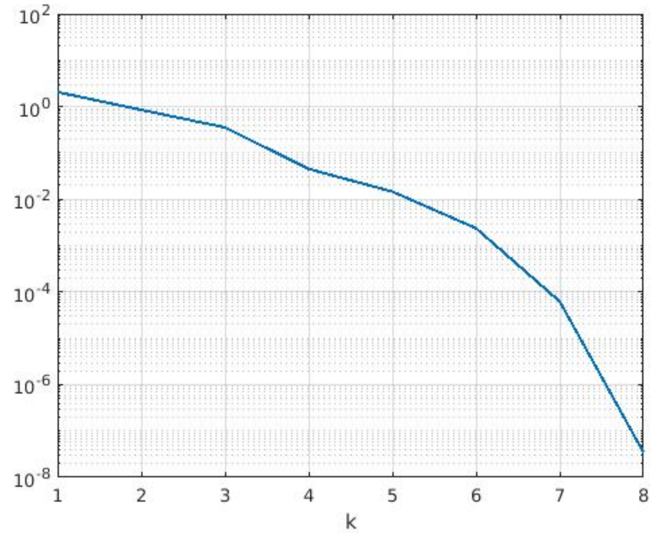


Figure 18: Norm of the gradient for **data1.mat**

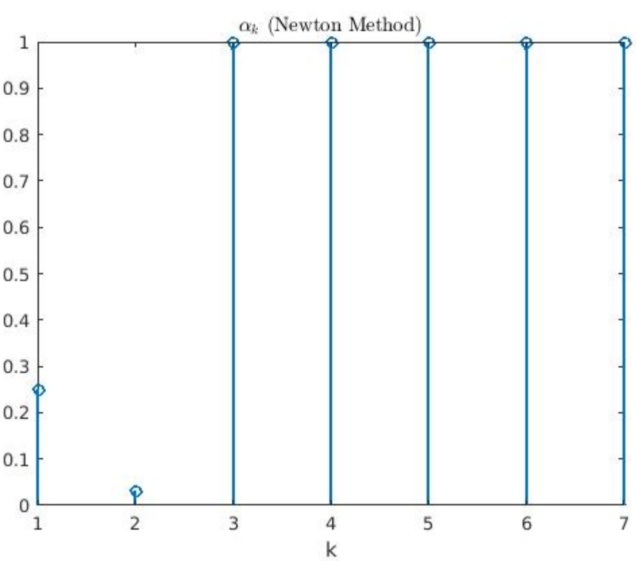


Figure 19: Value of α_k for **data1.mat**

b) Data2.m

The result of the Newton method for **data2.m** is present in figure

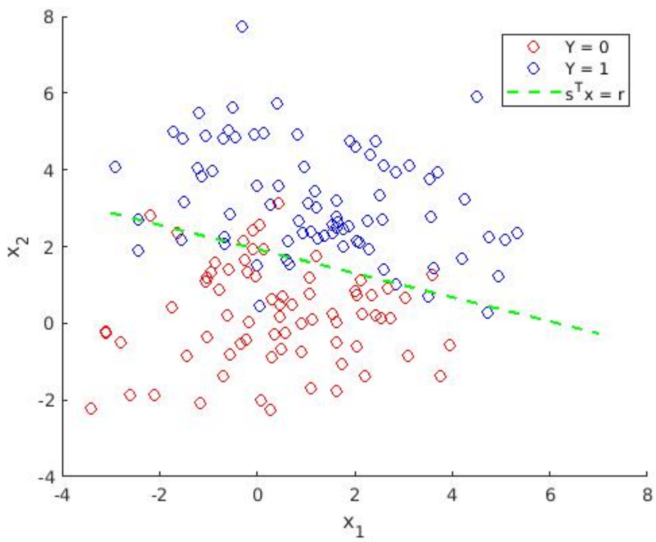


Figure 20: The dataset **data2.mat** with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed with the Newton method results.

The results are

- $s = (\ 0.7402,\ 2.3577)$
- $r = 4.5554$
- Number of iterations = 9

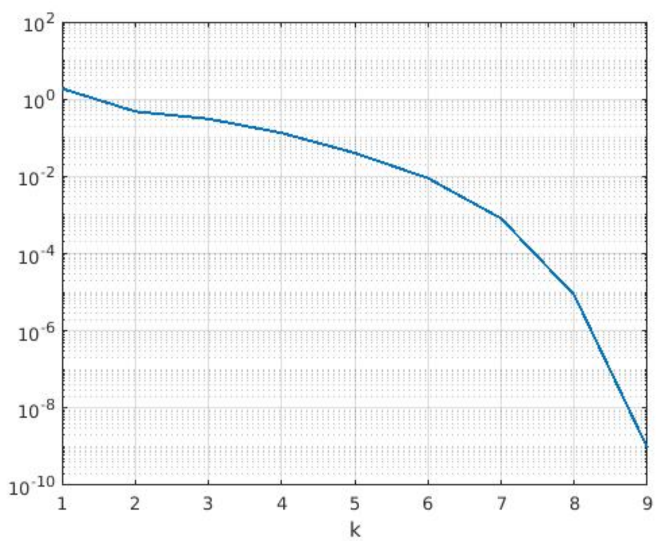


Figure 21: Norm of the gradient for **data2.mat**

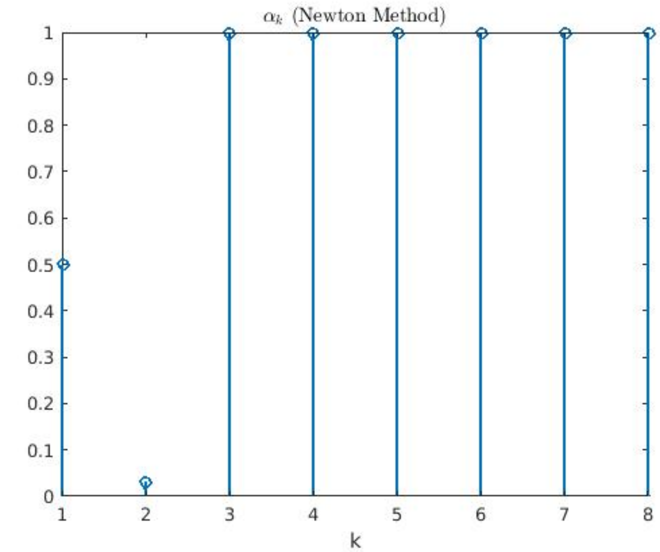


Figure 22: Value of α_k for **data2.mat**

c) Data3.m

The result of the Newton method for **data3.m** is present in figure

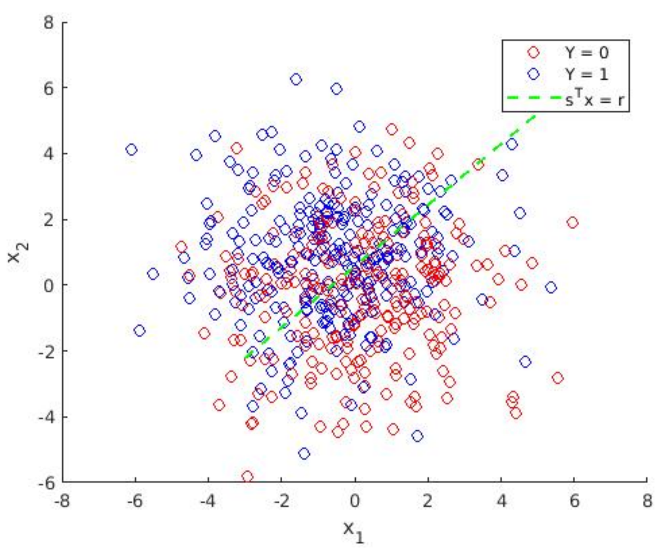


Figure 23: The dataset **data2.mat** with the green line $\{x \in \mathbb{R}^2 : s^T x = r\}$ superimposed with the Newton method results.

The results are

- s = (-1.3083 1.4079, 0.8049, -1.0025, 0.5548, -0.5489, -1.1998, 0.0792, -1.8280, -0.1484, 1.9242, -0.3586, -0.2900, 0.1925, 1.0615, 0.2107, -0.0929, 1.0477, -1.1249, -1.3311, 0.7662, -0.2729, -0.5349, 0.9996, -0.4192, -0.3133, 0.4075, -0.1965, -0.7380, -0.9815)
- $r = 4.7987$
- Number of iterations = 12

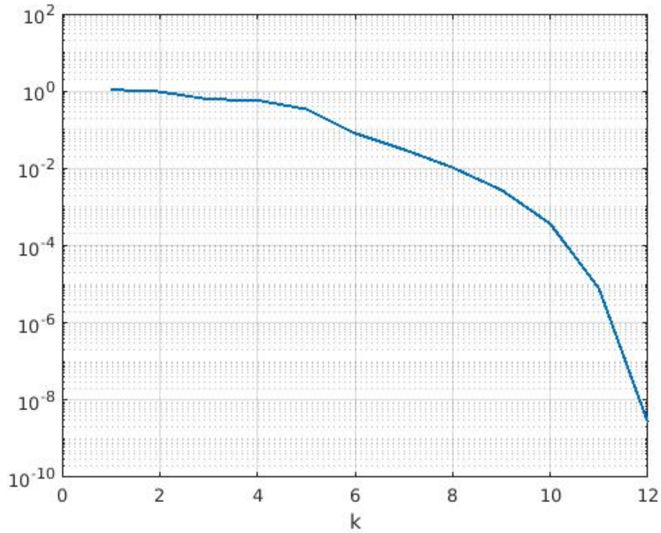


Figure 24: Norm of the gradient for **data3.mat**

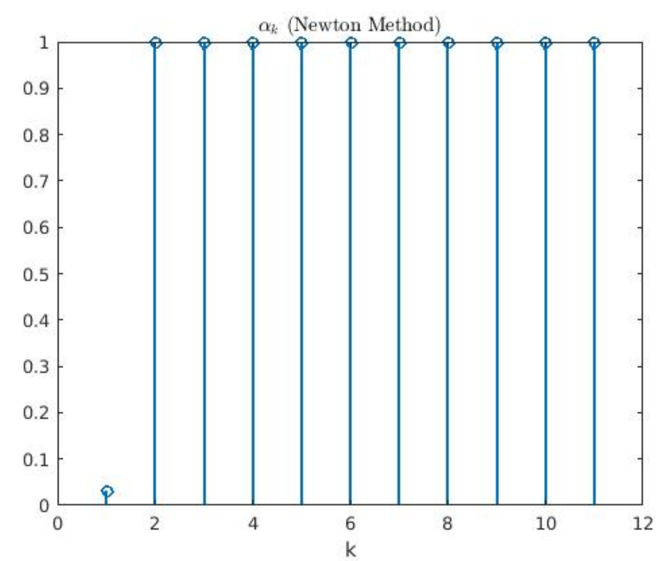


Figure 25: Value of α_k for **data3.mat**

d) Data4.m

The result of the Newton method for **data4.m**

- s = (0.1099, -0.6424, 0.1019, 1.2431, -1.6434, 1.0247, 0.0513, 0.8273, 0.3136, 0.7451, -0.5859, 0.6269, 1.3614, 0.1534, 2.3239, -0.0840, -0.9491, 2.4704, -0.8680, -1.6520, 0.6462, -0.4780, 1.6401, 0.9036, -1.2296, -0.7589, -0.4888, 1.0308, 0.0888, -1.0919, -1.2720, -2.0337, -0.2506, -0.3519, -0.3487, -2.5616, -0.3133, -0.4903, 0.7259, 0.5775, -1.0531, 0.6401, 0.3760, -0.1548, 0.0298, 0.9549, -0.2863, 0.6365, 0.7860, 0.7586, 0.2881, 0.1649, 0.6777, 2.0555, 1.0998, 0.5262, -0.5771, 1.1456, -0.5618, 0.0065, 0.4769, -2.3682, -1.1564, -2.6624, 0.0622, 0.1037, -0.6238, 0.1913, 0.6674, -1.0495, -0.3241, -0.3208, -1.0906, -0.8295, -0.3105, -0.4880, -0.1060, -0.1646, 2.2688, -1.2383, -0.8577, -2.4786, -0.4159, 0.1660, 0.7933, 0.3686, -0.0524, -0.9999, -0.5733, 0.3972, 1.1913, 1.8322, -1.7291, 0.2330, -1.1923, 1.6562, 0.4613, -0.6432, 0.8297, 0.2976)
- r = 7.6718
- Number of iterations = 12

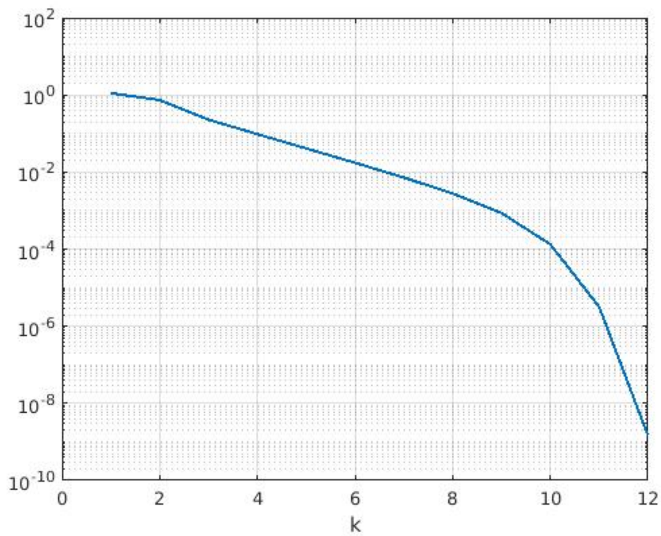


Figure 26: Norm of the gradient for **data4.mat**

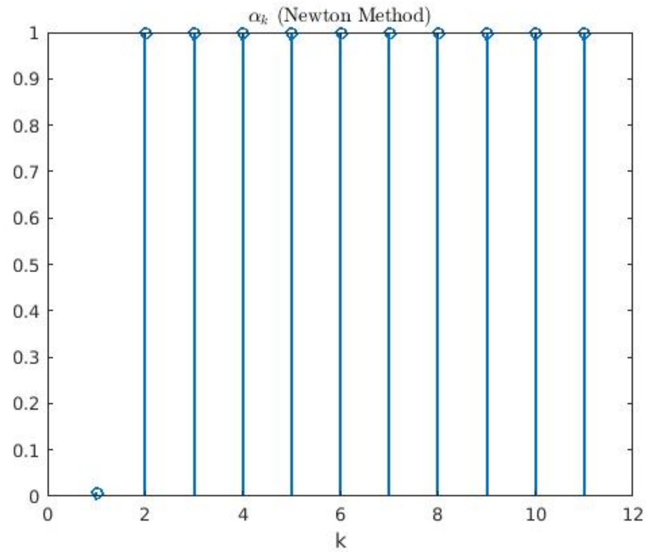


Figure 27: Value of α_k for **data4.mat**

For this task the following code was used:

```
% Uncomment the line for the dataset you want
%load('data1.mat');
%load('data2.mat');
%load('data3.mat');
%load('data4.mat');

%Amount_of input_features%
[n,K] = size(X);

%Stopping Criteria%
s0 = -ones(1,n);
r0 = 0;
epsilon = 10^(-6);
s_r0 = [s0 r0]';
A = [X; -ones(K,1)'];


%For the Backtracking Subroutine%
alpha0 = 1;
gama = 10^(-4);
beta = 0.5;
gradients=[];


s_r= s_r0;
alpha = alpha0;
alpha_k = [];
while (1)
    g_k = gradient_function(s_r, A ,Y,K);
    gradients = [gradients norm(g_k)];
    if norm(g_k) < epsilon
        break;
    end
```

```
    d = -hessian(s_r, A ,K)^(-1)*g_k;
    alpha = alpha0;
    while minimize_function(s_r+ alpha.*d, A, Y, K) ≥...
        minimize_function(s_r, A ,Y,K) + (gama.*g_k'*(alpha.*d))
        alpha = beta .* alpha;
    end
    alpha_k = [alpha_k alpha];
    s_r= s_r+ (alpha .* d);
end

s = s_r(1:length(s_r)-1)
r = s_r(length(s_r))
iterations = length(gradients)

% Ploting the gradient
figure();
semilogy(gradients);
grid on;

% Ploting alpha
figure('Name','Dataset 1(Newton Method)','NumberTitle','off');
stem(alpha_k);
title('$$\alpha_k$$ (Newton Method)','interpreter','latex')
xlabel('k');


% Ploting the dataset
figure('NumberTitle', 'off', 'Name', 'Dataset');
for i=1:K
   if Y(i) == 0
        a = scatter(X(1, i), X(2, i), [], 'red','LineWidth',1.25);
        hold on
    else
        b = scatter(X(1, i), X(2, i), [], 'blue','LineWidth',1.25);
        hold on
    end

end

x = linspace(-3,7);
y = s_r(3)/s_r(2) - x*(s_r(1)/s_r(2));
c = plot(x, y, '--y','Color','g', 'LineWidth', 1.5);
legend([a(1) b(1) c(1)], 'Y = 0', 'Y = 1', 's^Tx = r')
xlabel('x_1');
ylabel('x_2');
```

2.5 Task 7

Both gradient descent and Newton methods converge to similar results, as expected, since the cost function was proven to be convex, so there is a single global minimum.

The newton method requires a small amount of iterations to converge, always around the order of 10. This is impressive when compared to the gradient descent, which required thousand of iterations for smaller datasets and tens of thousands for medium to large datasets.

This can be explained by the Hessian matrix in computing the correction for the direction and norm of the gradient for each step. Taking a look at the value of the backtracking variable α along iterations - Figures 19, 22, 25 and 27 - after the second or third iteration the value does not change, there is no need to reduce the iteration step to get a better estimation of the minimum. Meanwhile, the gradient descent just takes steps with a fixed size in a direction, in contrast to the Newton method which takes longer or shorter steps when needed.

The Newton method does come with its caveats, the biggest one being the need to compute the hessian, which is computationally costly, increasingly slowly for problems with higher dimensions. Although the gradient descent requires more iterations, each iteration is cheap and fast, so this is an algorithm that scales much better. This is the reason why neural networks use gradient descent instead of the Newton method, if the hessian is not previously known and has to be computed, which will be very computationally expensive and slow, so it becomes better to use the gradient descent.

The choice between the two methods really comes down to the dimension of the problem and if the hessian is previously known or not. If the Hessian is known before-hand and it is easily computed, then the Newton method might be a good choice if the dimensions are not too big. For problems with low dimensions, the Newton method is preferred. As the dimensions increase, the computation of the Hessian just becomes too expensive to compute, which will give the gradient descent an advantage.

3 Part 3

3.1 Task 1

$$D_{mn} = ||x_m - x_n||_2 \tag{22}$$

$$D = \begin{bmatrix} D_{11} & \dots & D_{1n} \\ \vdots & \vdots & \vdots \\ D_{m1} & \dots & D_{mn} \end{bmatrix} \tag{23}$$

In order to reduce the dimensionality of our data, an Euclidean distance matrix, D (equation:23), was computed for the ten-dimensional dataset **data_opt.csv**. The largest distance in the given dataset was 83.0030 for the pair (134, 33).

```
X = csvread('../data/data_opt.csv');

Z = pdist(X);
D = squareform(Z);

D(2,3)
D(4,5)


max_D = max(Z)

[max_index_x, max_index_y] = find(D == max_D)
```

3.2 Task 2

The problem to be solved will be the one present in the equation 24

$$\min_{y} \quad f(y) = \min_{y} \quad \sum_{m=1}^{N} \sum_{n<m} (f_{nm}(y))^2 \tag{24}$$

with

$$f_{nm}(y) = ||y_m - y_n|| - D_{mn}. \tag{25}$$

In order to solve the problem, the $\nabla f(y)$ will be needed.

$$\nabla f(y) = \begin{bmatrix} \frac{\partial f(y)}{\partial y_{11}} & \dots & \frac{\partial f(y)}{\partial y_{nK}} \end{bmatrix} \tag{26}$$

Each element of the vector in equation 26 will be given by:

$$\frac{\partial f(y)}{\partial y_{nk}} = \frac{\partial}{\partial y_{nk}} (\sum_{m=1}^{N} \sum_{n<m} (f_{nm}(y))^2) = 2 \sum_{m=1}^{N} \sum_{n<m} \frac{\partial f_{nm}(y)}{\partial y_{nk}} f_{nm}(y) \tag{27}$$

So, the $\nabla f(y)$ can be represented as:

$$\nabla f(y) = 2 \sum_{m=1}^{N} \sum_{n<m} \nabla f_{nm}(y) f_{nm}(y) \tag{28}$$

To simplify the equation in 27, the $\frac{\partial f_{nm}(y)}{\partial y_{nk}}$ will be analytically computed, for that we have:

$$\begin{aligned} f_{12}(y) = ||y_2 - y_1|| - D_{21} &= \sqrt{(y_{11} - y_{21})^2 + \dots + (y_{1K} - y_{2K})^2} - D_{21} \\ \frac{\partial f_{12}(y)}{\partial y_{11}} &= \frac{1}{2}[(y_{11} - y_{21})^2 + \dots + (y_{1K} - y_{2K})^2]^{-\frac{1}{2}} 2(y_{11} - y_{21}) \\ \frac{\partial f_{12}(y)}{\partial y_{11}} &= [(y_{11} - y_{21})^2 + \dots + (y_{1K} - y_{2K})^2]^{-\frac{1}{2}} (y_{11} - y_{21}) \end{aligned} \tag{29}$$

With that, it is possible to represent $\nabla f_{nm}(y)$ as,

$$\nabla f_{nm}(y) = \begin{bmatrix} \frac{\partial f_{nm}(y)}{\partial y_{11}} & \dots & \frac{\partial f_{nm}(y)}{\partial y_{1k}} \\ & \vdots & \\ \frac{\partial f_{nm}(y)}{\partial y_{N1}} & \dots & \frac{\partial f_{nm}(y)}{\partial y_{Nk}} \end{bmatrix} \tag{30}$$

For the construction of the matrices A and b, to simplify the notation, p is the combination of the nm correspondent (example: for $p = 2$, $n = 1$ and $m = 2$). So, both matrices be of dimension $(NK+1)$x1.

Matrices A and b are defined as:

$$A = \begin{bmatrix} \nabla f_1(x_k)^T \\ \vdots \\ \nabla f_p(x_k)^T \\ \sqrt{\lambda_k} I \end{bmatrix} \qquad b = \begin{bmatrix} \nabla f_1(x_k)^T x_k - f_1(x_k) \\ \vdots \\ \nabla f_p(x_k)^T x_k - f_p(x_k) \\ \sqrt{\lambda_k} x_k \end{bmatrix} \tag{31}$$

3.3 Task 3

For this task, the goal was to solve

$$\min_{y} \quad \sum_{m}^{N} \sum_{n>m} (||y_m - y_n|| - D_{mn})^2 \tag{32}$$

.

It was used $f(y) = \sum_m^N \sum_{n>m}(||y_m - y_n|| - D_{mn})^2$. Equation 32 is used to perform a dimensionality reduction on our data, where $D_{mn} = ||x_m - x_n||_2$, which represents the distances between all points in the original dataset, with values x_k. So the goal was to solve for y in order to find the best y with a given dimension, either $\mathbb{R}^2$ or $\mathbb{R}^3$, which was dictated by the variable $k \in \{2, 3\}$, that would keep the distances between points in the original dataset but with a smaller dimension.

So taking $k = 2$ we managed to reduce the original dataset, of dimensions { 200, 10 }, dimension 10, to a dataset of dimensions { 200, 2 }, dimension 2. For this task the an initial value was supplied for y.

In order to do this the Levenerg-Marquardt method was used. This method required a rewrite in the problem formulation:

$$\min_{y} \quad \sum_{m=1}^{N} \sum_{n>m} (f_{nm}(y))^2$$

Where f_{nm} is the funtion defined in Equation 25.

a) 2 dimensions

The output of the dataset was:

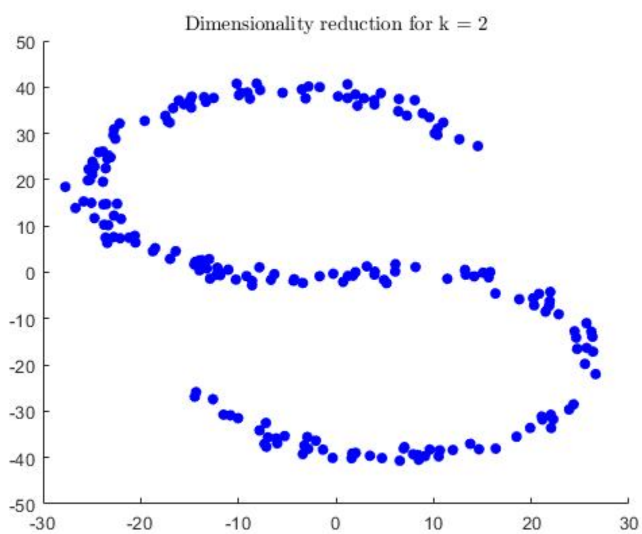


Figure 28: x_1, x_2 for $k = 2$

Where the y vector we get from the LM method is split in two, given that the y vector produced is of the form, taking N = 200:

$$y = \begin{bmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \\ \vdots \\ y_{N1} \\ y_{N2} \end{bmatrix} \tag{33}$$

It is transformed to:

$$x_1 = \begin{bmatrix} y_{11} \\ y_{21} \\ \vdots \\ y_{N1} \end{bmatrix} \qquad x_2 = \begin{bmatrix} y_{12} \\ y_{22} \\ \vdots \\ y_{N2} \end{bmatrix} \tag{34}$$

It is then plotted x_1 in the xx axis and x_2 in the yy axis.
Looking now at the changes of the cost function along iterations, in Figure 29.

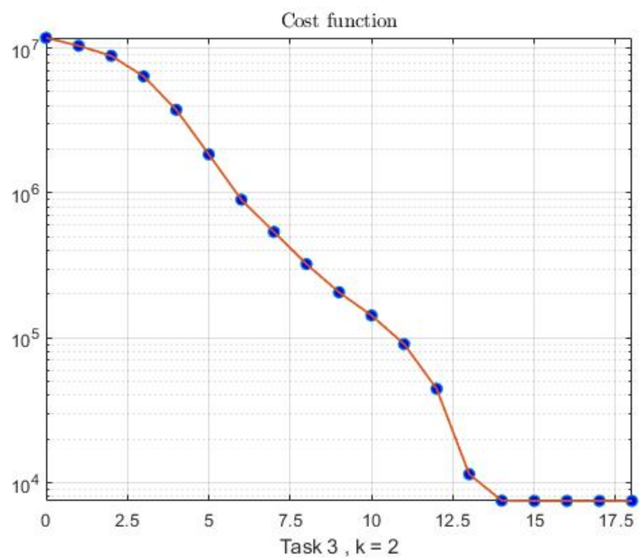


Figure 29: Cost function value along iterations of the LM method for $k = 2$

The cost function is decreasing along iterations and converging which means that the values obtained for y are the best one could get with the given initializations.

b) 3 dimensions

Lets now see how the dimensions are reduced from the 10 given dimensions to 3 dimensions, $k = 3$.

The y vector now comes in the following shape:

$$y = \begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{21} \\ y_{22} \\ y_{23} \\ \vdots \\ y_{N1} \\ y_{N2} \\ y_{N3} \end{bmatrix} \tag{35}$$

It is transformed to:

$$x_1 = \begin{bmatrix} y_{11} \\ y_{21} \\ \vdots \\ y_{N1} \end{bmatrix} \qquad x_2 = \begin{bmatrix} y_{12} \\ y_{22} \\ \vdots \\ y_{N2} \end{bmatrix} \qquad x_3 = \begin{bmatrix} y_{13} \\ y_{23} \\ \vdots \\ y_{N3} \end{bmatrix} \tag{36}$$

It is now obtained the following y:

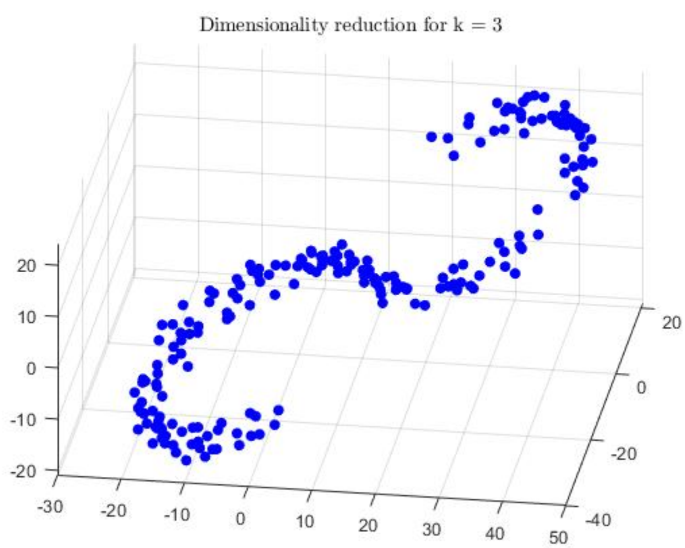


Figure 30: x_1, x_2, x_3 for $k = 3$

The cost function behaves as seen in Figure 31.

Where it can be observed that the method is able to converge very quickly to around its best possible value and then takes very small improvements.

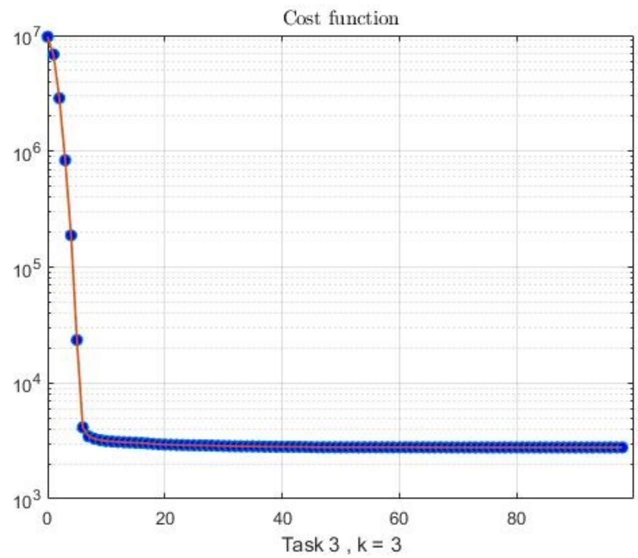


Figure 31: Cost function value along iterations of the LM method for $k = 3$

c) MATLAB Code

The code needed for both $k = 2$ and $k = 3$ is the following:

```
X = csvread('../data/data_opt.csv');

% Pick whether you want k=2 or k=3 and its corresponding dataset
% Y = csvread('../data/yinit2.csv');
Y = csvread('../data/yinit3.csv');
k = 3;


N = size(Y, 1)/k;

lambda_k = 1;
epsilon = k*10^(-2);

% Creating matrix D
Z = pdist(X);
D = squareform(Z);

% Staring values
y_k = Y;
i = 1;


cost_f = [];
while(1)
    % Getting the f_nm gradients stacked into grad_fnm
    % Getting the f_nm values stacked into f_nms
    % Getting the gradient of f into grad_f
    [grad_fnm,f_nms,grad_f] = f_gradient(y_k, D, N, k);
    if norm(grad_f) < epsilon
        break
    end
```

```

    I = eye(size(Y,1));
    A = [
        grad_fnm;
        sqrt(lambda_k).*I
    ];
    b = [
        grad_fnm*y_k - f_nms';
        sqrt(lambda_k).*y_k
    ];

    % Solving the least squares problem
    y_hat = A \ b;

    % Getting the values for the new y_hat
    [grad_fnm_hat,f_nms_hat,grad_f_hat] = f_gradient(y_hat, D, N, k);

    %LM comparison
    if f(f_nms_hat) < f(f_nms)
        y_k = y_hat;
        lambda_k = 0.7*lambda_k;
        f_nms = f_nms_hat;
        grad_fnm = grad_fnm_hat;
        grad_f = grad_f_hat;
    else
        lambda_k = 2*lambda_k;
    end
    i = i + 1;
    cost_f = [cost_f f(f_nms)];

end

% Plotting the cost function
figure('NumberTitle', 'off', 'Name', 'Cost function');
semilogy(0:length(cost_f)-1,cost_f,'o','MarkerFaceColor', 'b', 'LineWidth',1);
hold on;
semilogy(0:length(cost_f)-1,cost_f, 'LineWidth',1.25);
grid on;
title('Cost function','interpreter','latex')
xlabel('Task 3 , k = 2')
xticks(0:20:length(cost_f)-1)

% Splitting the y array into k arrays
y_hat =  reshape(y_hat, k, [] );


%Plot Scatter

if k == 2
    figure('NumberTitle', 'off', 'Name','Dimensionality reduction for k = 2');
    a = scatter(y_hat(1,:), y_hat(2,:), [], 'blue','filled','LineWidth',1);
    title('Dimensionality reduction for k = 2','interpreter','latex')
    hold on;
else
    figure('NumberTitle', 'off', 'Name','Dimensionality reduction for k = 3');
    a = scatter3(y_hat(1,:), y_hat(2,:),y_hat(3,:), [], 'blue',...
```

```
        'filled','LineWidth', 1);
    title('Dimensionality reduction for k = 3','interpreter','latex')
    hold on;
end
```

Where the following functions are featured:

```
function sumz = f(f_nm)
    sumz = sum(f_nm.^2);
end
```

```
function f = f_nm(ym, yn, D, n, m)
    f = norm(ym - yn) - D(m, n);
end
```

```
function gradient = f_nm_gradient(ym, yn, n, m, N, k)
    gradient = zeros(N*k, 1);
    for i=1:N
        for j = 1:k
            if i == m
                gradient((i*k) - k + j) = (ym(j) - yn(j)) / norm(ym-yn);
            elseif i == n
                gradient((i*k) - k + j) = (-1) * (ym(j) - yn(j)) / norm(ym-yn);
            end
        end
    end
end
```

```
function [gradientz,f_nms,grad_f] = f_gradient(Y, D, N, k)
    gradients = [];
    f_nms=[];
    for m =1:N
        for n = (m+1):N
            gradients = [
                gradients f_nm_gradient(Y( (k*m)-(k - 1):(k*m) ),...
                    Y( (k*n)-(k-1):(k*n) ), n, m, N, k)
            ];
            f_nms = [ f_nms f_nm(Y( (k*m)-(k - 1):(k*m) ),...
                Y( (k*n)-(k-1):(k*n) ), D, n, m)];
        end
    end
    gradientz = gradients';
    grad_f = 2*gradients*f_nms';
end
```

3.4 Task 4

In order to interpret the behaviour of the data present in the file *dataProj.m* there were made inicialization vectors with different random values. Images 32 and 33 represent the cost functions and the dataset achieved from those inicializations.

Table 8: Cost Values in the following intervals: $[-50, 50]$, $[-125, 125]$, $[-200, 200]$, $[-275, 275]$,$[-350, 350]$,$[-425, 425]$,$[-500, 500]$

$[-50, 50]$	$[-125, 125]$	$[-200, 200]$	$[-275, 275]$	$[-350, 350]$	$[-425, 425]$	$[-500, 500]$
1.90e+07	7.64-05	7.64e-05	7.64e-05	7.64e-05	7.64e-05	7.64e-05

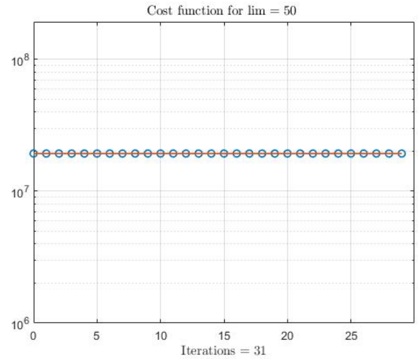
It's possible to take some conclusions from the images obtained from the previous step. One of them is that for initialization vectors with values that range -50 and 50 it's not possible to obtain a dimensionality reduction, the cost value is constant and very high. For values higher than the previous one, it's possible to make that reduction and the cost is always the same.

After figuring out the behaviour of the dataset with different kinds of initialization, the dataset was then split into subsets, taking pairs of combinations to see if using values from the dataset would give us a better initialization vector then plain random vectors. The total number of combinations from the given data is given by:

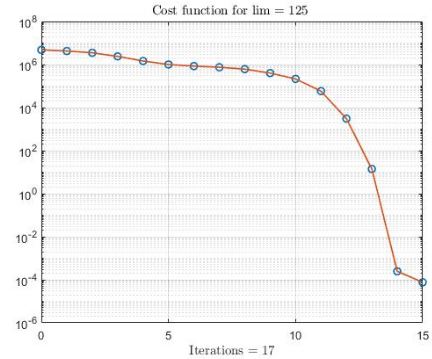
$$\binom{n}{k} = {}^{n}C_{k} = \frac{n!}{k!(n-k)!} = \binom{10}{2} = 45 \tag{37}$$

The values of the cost function for every combination were stored in a variable and was possible to conclude the unicity of the solution, all of the combinations could reduce the size of the dataset, there was only the need to choose the best one, the one that had the least amount of iterations.

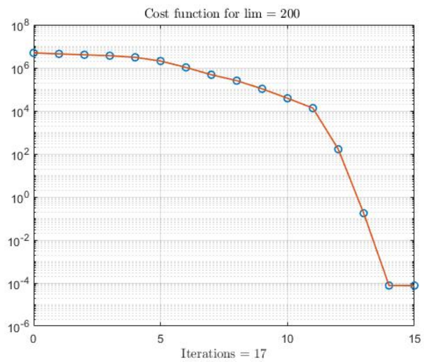
As it could be seen before, all of the combinations presented the same cost for the solution, the solution is unique. This is because the cost function can't be further minimized, this is the best it gets so no matter the initial y vector we use, the cost function will always get the same value.



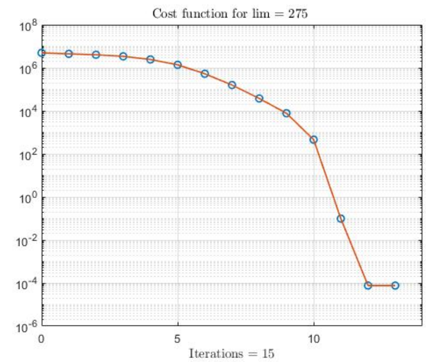
(a) $Values \in (-50, 50)$



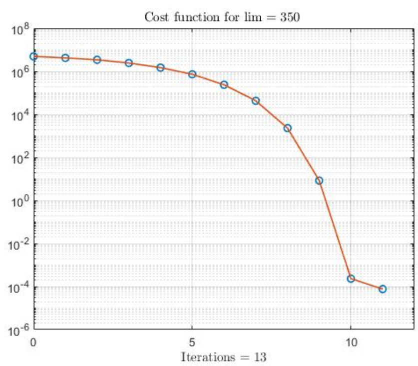
(b) $Values \in (-125, 125)$



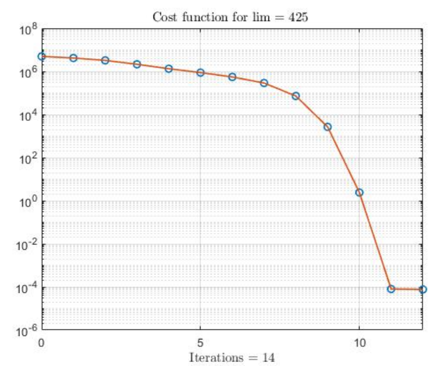
(c) $Values \in (-200, 200)$



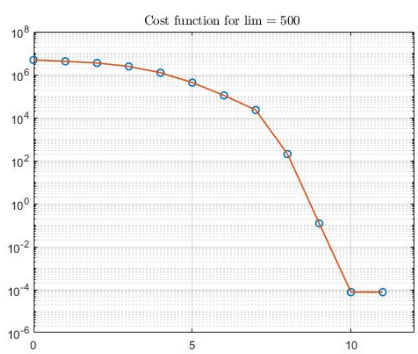
(d) $Values \in (-275, 275)$



(e) $Values \in (-350, 350)$

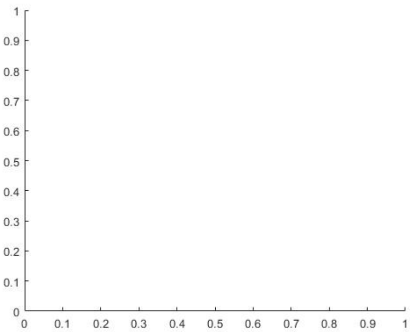


(f) $Values \in (-425, 425)$

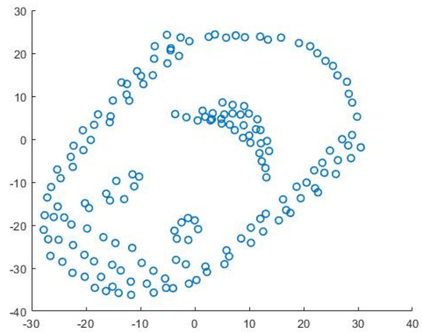


(g) $Values \in (-500, 500)$

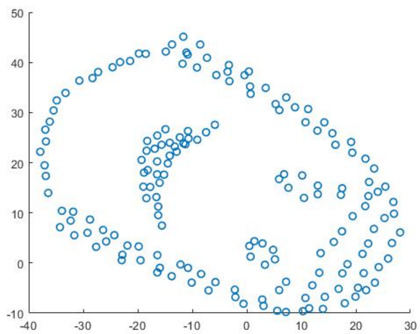
Figure 32: Cost functions for the different initializations.



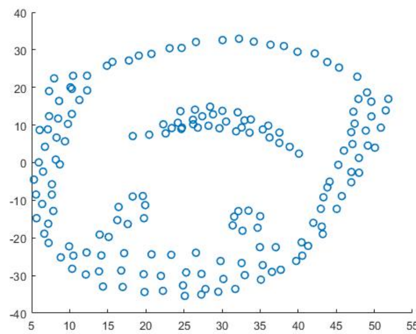
(a) $Values \in (-50, 50)$



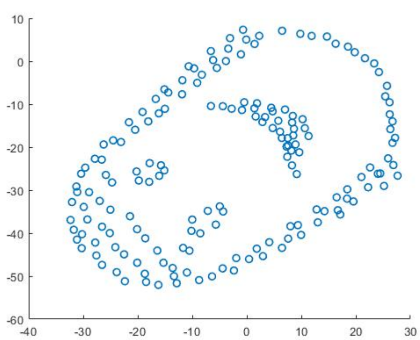
(b) $Values \in (-125, 125)$



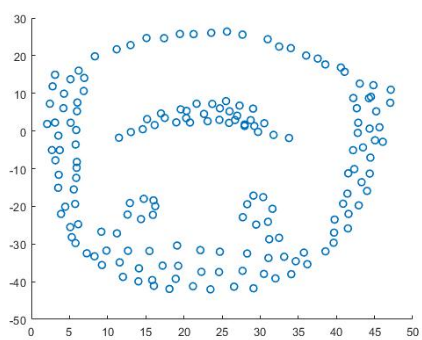
(c) $Values \in (-200, 200)$



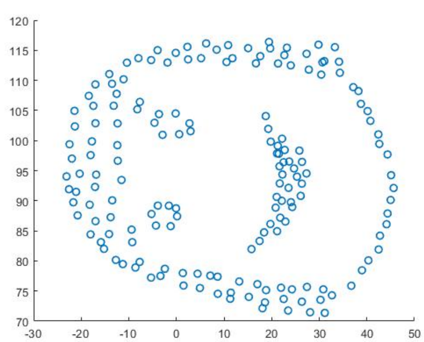
(d) $Values \in (-275, 275)$



(e) $Values \in (-350, 350)$

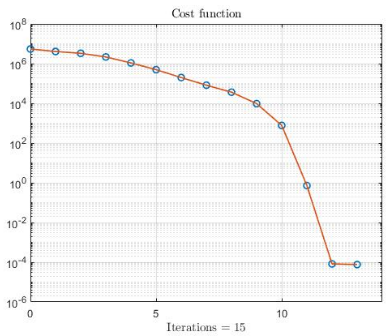


(f) $Values \in (-425, 425)$

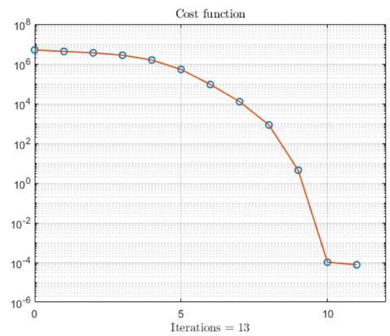


(g) $Values \in (-500, 500)$

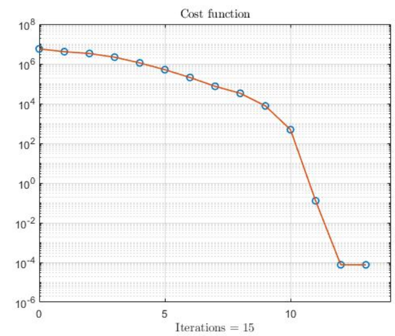
Figure 33: Dataset representation for the different initializations.



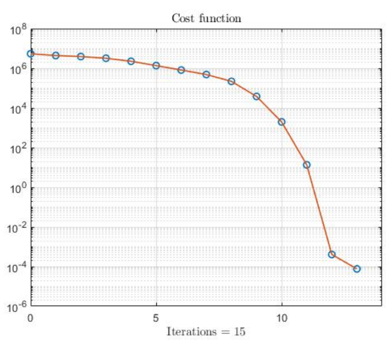
(a) $X(1,2)$



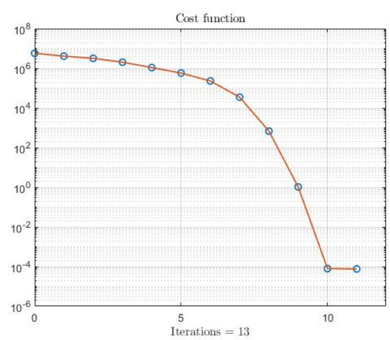
(b) $X(1,3)$



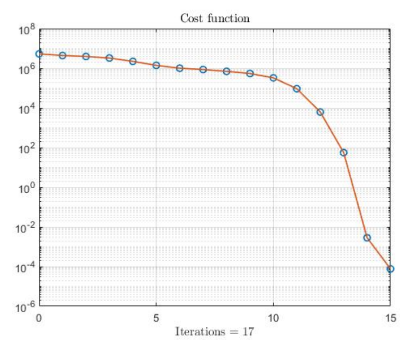
(c) $X(1,4)$



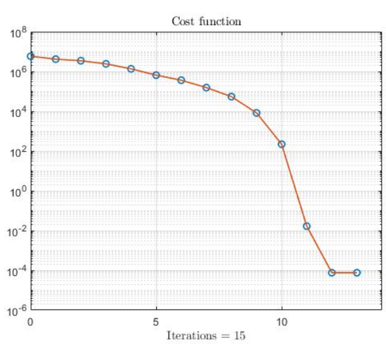
(d) $X(1,5)$



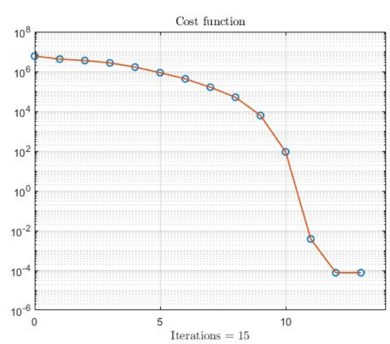
(e) $X(1,6)$



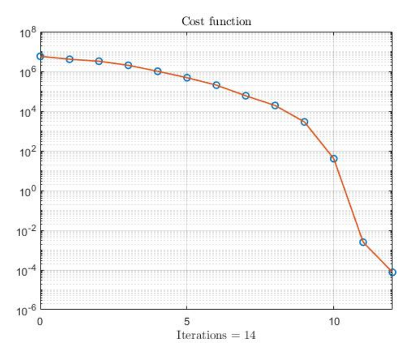
(f) $X(1,7)$



(g) $X(1,8)$

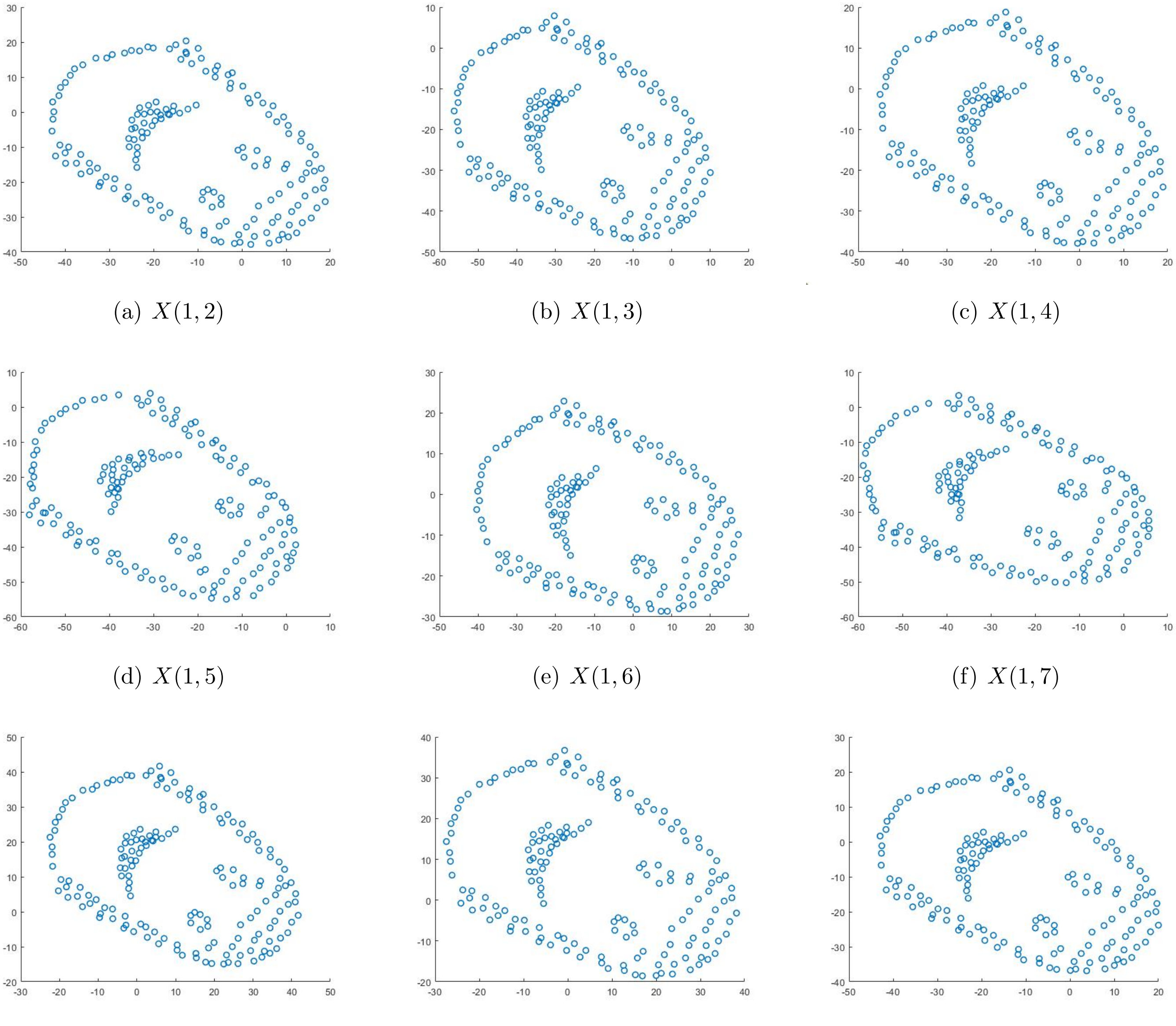


(h) $X(1,9)$



(i) $X(1,10)$

Figure 34: Cost function of the first 9 vector combinations from the dataset.



(a) $X(1,2)$ (b) $X(1,3)$ (c) $X(1,4)$

(d) $X(1,5)$ (e) $X(1,6)$ (f) $X(1,7)$

(g) $X(1,8)$ (h) $X(1,9)$ (i) $X(1,10)$

Figure 35: Dataset representation for the first 9 combinations.

a) MATLAB code

```
% Y = csvread('../data/yinit2.csv');
X = csvread('../data/dataProj.csv');

k = 2;

lambda_k = 1;
epsilon = k*10^(-4);




%%
store_costs_rand = [];
iterations = [];
y_rand = [];
lim = 50;

for t = 1:7
    y_rand = randi([-1*(lim), lim],[size(X,1)*2,1]);
    [store_costs_rand(:,t), iterations(:,t)] = Levenverg(y_rand,k,epsilon,X,lim);
    lim = lim +75 ;
end

%%
combos = nchoosek(1:size(X,2),2);
number_of_combos = size(combos,1);
store_cost = [1,number_of_combos];
number_of_iterations = [1,number_of_combos];
lim = 0;
for a =1:number_of_combos

    y_chosen = [X(:,combos(a,1));
                X(:,combos(a,2))];

    [store_cost(:,a) ,number_of_iterations(:,a)] = Levenverg(y_chosen,k,epsilon,X, lim);

end

%%
[C,D] = min(number_of_iterations);
%%
function [store_cost, iterations] = Levenverg(y_input,k,epsilon,X, lim)
    cost_f = [];
    y_hat = [];
    A = [];
    b = [];
    lambda_k = 1;
    i = 1;
    Y = y_input;
    y_k = Y;
    N = size(Y, 1)/k;
    Z = pdist(X);
    D = squareform(Z);
```

```

    while(1)
        [grad_fnm,f_nms,grad_f] = f_gradient(y_k, D, N, k);
        if norm(grad_f) < epsilon
            break
        end

        I = eye(size(Y,1));
        A = [
            grad_fnm;
            sqrt(lambda_k).*I
        ];
        b = [
            grad_fnm*y_k - f_nms';
            sqrt(lambda_k).*y_k
        ];

        y_hat = A \ b;

        [grad_fnm_hat,f_nms_hat,grad_f_hat] = f_gradient(y_hat, D, N, k);

        if f(f_nms_hat) < f(f_nms)
            y_k = y_hat;
            lambda_k = 0.7*lambda_k;
            f_nms = f_nms_hat;
            grad_fnm = grad_fnm_hat;
            grad_f = grad_f_hat;
        else
            lambda_k = 2*lambda_k;
        end
        i = i + 1;
        cost_f = [cost_f f(f_nms)];
%          aux = h
        if i > 30
            break
        end

    end
        iterations = i;
        store_cost = cost_f(1,length(cost_f));

    if lim > 0
        figure('NumberTitle', 'off', 'Name', 'Cost function');
        semilogy(0:length(cost_f)-1,cost_f,'o', 'LineWidth',1.25);
        hold on;
        semilogy(0:length(cost_f)-1,cost_f, 'LineWidth',1.25);
        grid on;
        str = sprintf('Cost function for lim = %d ', lim);
        title(str, 'interpreter','latex');
        str2 = sprintf('Iterations = %d', iterations);
        xlabel(str2,'interpreter','latex');
        xticks(0:5:length(cost_f)-1)


        %Plot Scatter
        figure('NumberTitle', 'off', 'Name', 'Task_4_Dataset');
```

```
        y_hat =  reshape(y_hat, 2, [] );
        plot( y_hat(1,:), y_hat(2,:))
        str = sprintf('Dataset for lim = %d ', lim);
        title(str,'interpreter','latex');

        a = scatter(y_hat(1,:), y_hat(2,:), [],'LineWidth',1.25);
        hold on

    end

end
```