Optimization and Algorithms
Project report

Group 7
Martim Reis 81427, Daniel Fortunato 81498 and Pedro Soares 81595
Instituto Superior Técnico

December 2018

Contents

1 Part 1

In this part of the project we were asked to solve a series of optimization problems looking at the same problem from different perspectives, dividing the problem in three sections. The problem was: How can we act on a robot, using simple controls, so it transfers from a given position to another desired position?

In the first section (task 1 to task 4) we had to solve the problem so the robot passes, as close as possible, to a series of intermediate points along its journey.

In the second section (task 5 and task 6) we had to do the same but now, instead of intermediate points, the robot has to pass, as close as possible, to a series of intermediate regions.

Lastly, in the third section (task 7 to task 12) the robot as to pass exactly on as many intermediate points as possible.

1.1 Task 1 - The ℓ_2^2 regularizer

In this sections we were asked to solve the problem 1 using CVX:

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

where $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$.

The code used for this task is listed bellow:

```

cvx_begin
    variable x(4,T);
    variable u(2,T);
    F1=0;
    F2=0;

    for k=1:6
        F1= F1 + square_pos(norm(x(1:2,tau(k))-w(:,k)));
    end

    for t=2:T-1
        F2 = F2 + square_pos(norm(u(:,t)-u(:,t-1)));
    end

    minimize(F1 + lambda(7)*F2)

    subject to
        x(:,1) == [pinitial; 0; 0];
        x(:,T) == [pfinal; 0; 0];

        for i=1:T-1
```

```
23                norm(u(:,i))≤Umax;
24            end
25
26            for j=1:T-1
27                x(:,j+1)==A*x(:,j) + B*u(:,j);
28            end
29
30 cvx_end
```

We solved the problem using CVX by splitting the function bellow into two functions, f_1 and f_2, that are shown in the equations 3 and 4.

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

$$\sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 \tag{3}$$

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 \tag{4}$$

To do the sum of both equations we did two cycles *for*, as it can be seen in the line 9 and 13 of the code. Finally we minimized $f_1 + \lambda f_2$ for every λ using CVX, with the constraints given. The solutions are show in the subsections bellow.

1.1.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red squares. We show that for all the λ in the figures 1, 2, 3 and 4.

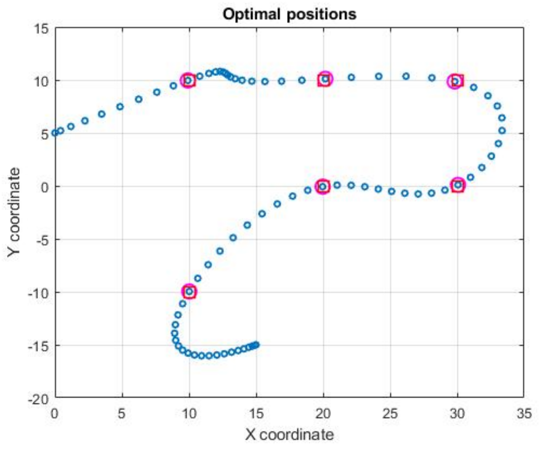
1.1.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$. We show that for all the λ in the figures 5, 6, 7 and 8.

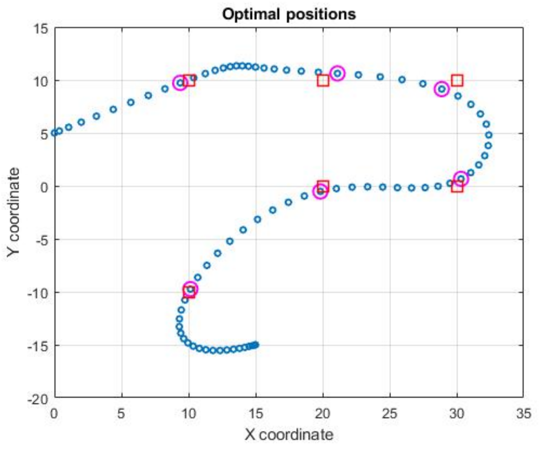
1.1.3 Changes of the optimal control signal

In this subsection we report how many times the optimal control signal changes from $t = 1$ to $t = T - 1$.

We say that the control signal changes at time t if $\|u(t) - u(t-1)\|_2 > 10^{-6}$. We show that for all λ in the table 1:

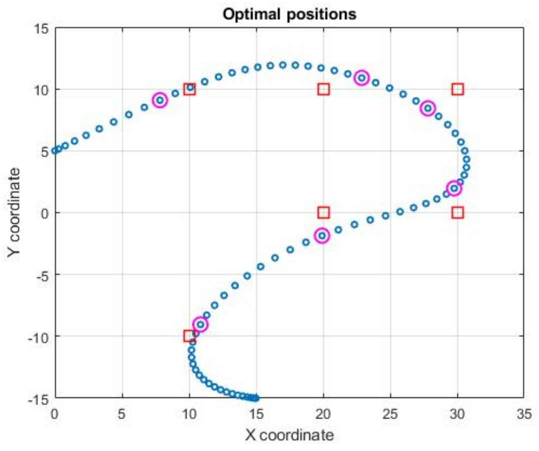


(a)

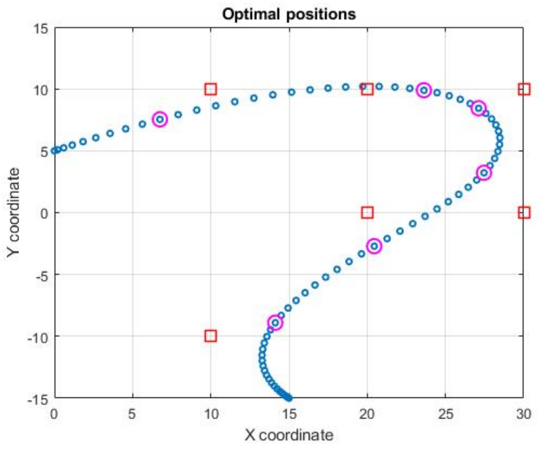


(b)

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

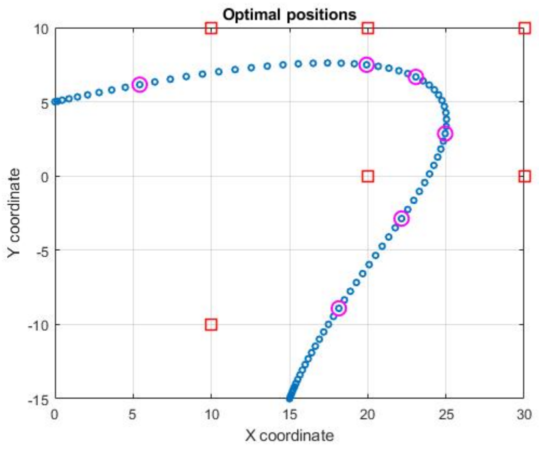


(a)

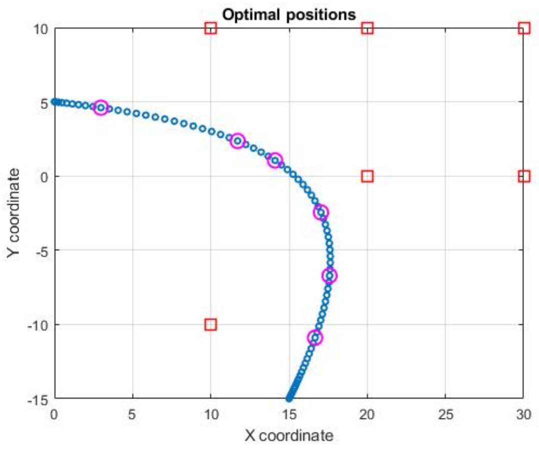


(b)

Figure 2: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$



(a)



(b)

Figure 3: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

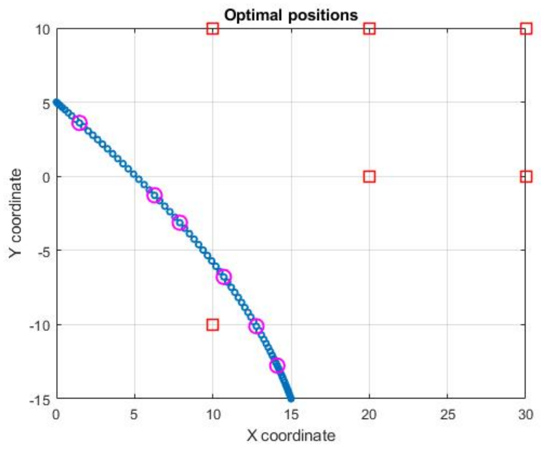
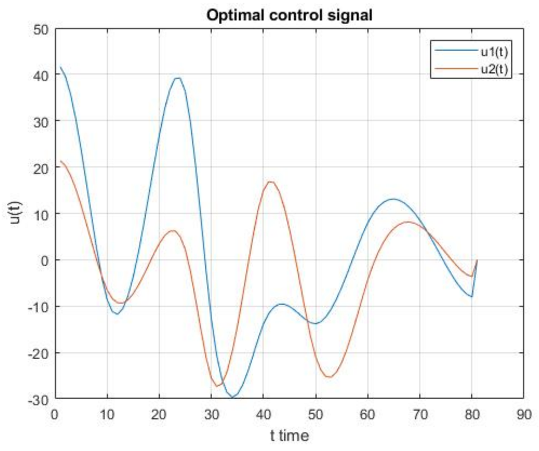
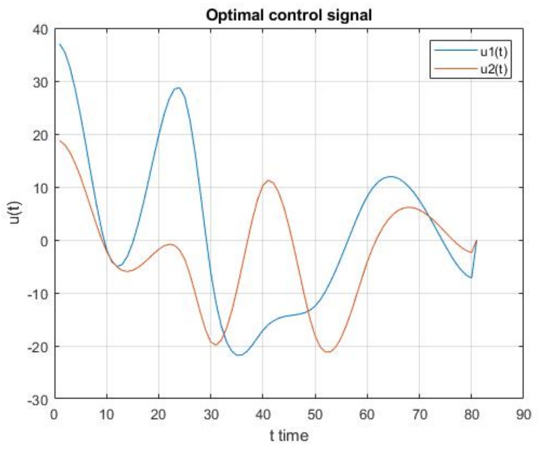


Figure 4: $\lambda = 10^3$

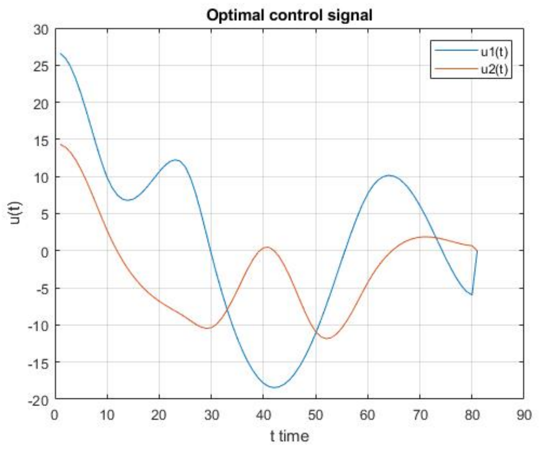


(a)

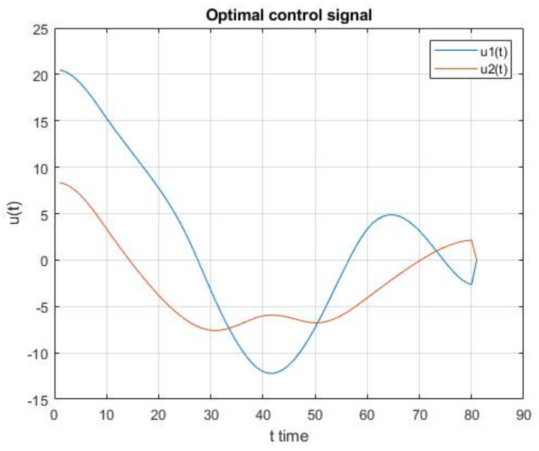


(b)

Figure 5: (a) $\lambda = 10^{-3}$; (b) $\lambda = 10^{-2}$

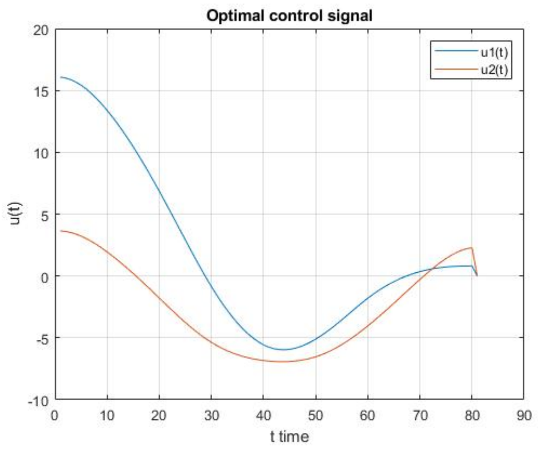


(a)

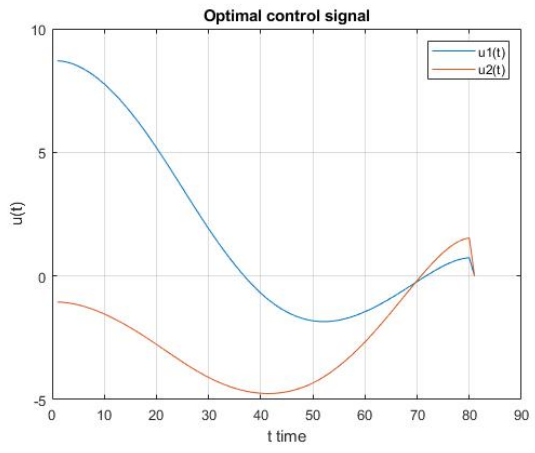


(b)

Figure 6: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$



(a)



(b)

Figure 7: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

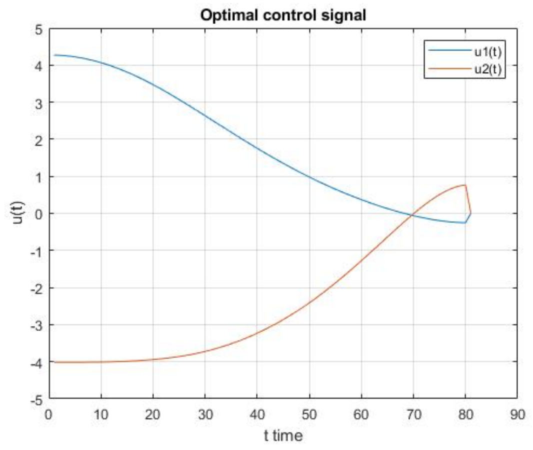


Figure 8: $\lambda = 10^3$

λ	Nº of times control signal changes
10^{-3}	79
10^{-2}	79
10^{-1}	79
10^{0}	79
10^{1}	79
10^{2}	79
10^{3}	79

Table 1: Number of times the control signal changes

1.1.4 Mean Deviation of the waypoints

In this subsection we report the mean deviation from the waypoints, defined as

$$\frac{1}{K}\sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2 \, , \tag{5}$$

which tells us how much the waypoints are ignored.

We show that for all the λ in the table 2:

λ	Mean deviation from the waypoints
10^{-3}	$0,1257$
10^{-2}	$0,8242$
10^{-1}	$2,1958$
10^{0}	$3,6826$
10^{1}	$5,6317$
10^{2}	$10,9042$
10^{3}	$15,3304$

Table 2: Mean deviation from the waypoints

1.2 Task 2 - The ℓ_2 regularizer

In this sections we were asked to solve the problem 6 using CVX. This problem is basically the same as the problem 1 but now we remove the square in the ℓ_2 regularizer.

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1. \end{array} \tag{6}$$

where $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$.

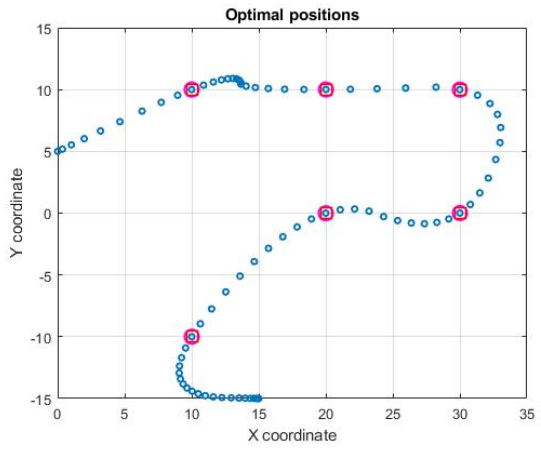
The code used for this task is listed bellow:

```
1  cvx_begin
2      variable x(4,T);
3      variable u(2,T);
4      F1=0;
5      F2=0;
6
7      for k=1:6
8          F1= F1 + square_pos(norm(x(1:2,tau(k))-w(:,k)));
9      end
10
11     for t=2:T-1
12         F2 = F2 + norm(u(:,t)-u(:,t-1));
13     end
14
15     minimize(F1 + lambda(1)*F2)
16
17     subject to
18         x(:,1) == [pinitial; 0; 0];
19         x(:,T) == [pfinal; 0; 0];
20
21         for i=1:T-1
22             norm(u(:,i))≤Umax;
23         end
24
25         for j=1:T-1
26             x(:,j+1)==A*x(:,j) + B*u(:,j);
27         end
28
29 cvx_end
```

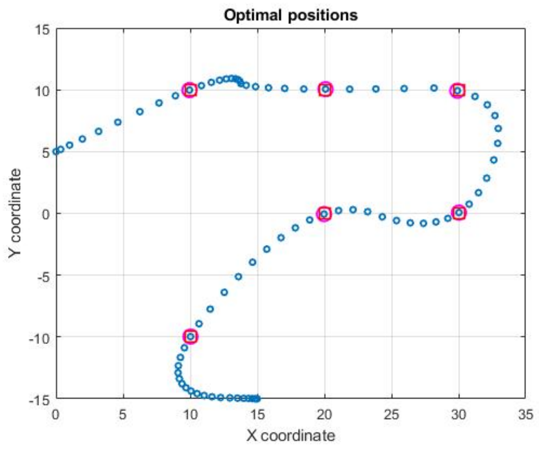
Computationally, to solve this problem we did exactly the same thing that we did in the problem 1 but, as we can see in the line 12 of the code, instead of the squared norm in the regularizer we just have the euclidean norm. The solutions of this problem 6 are shown in the subsections bellow.

1.2.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red squares. We show that for all the λ in the figures 9, 10, 11 and 12.

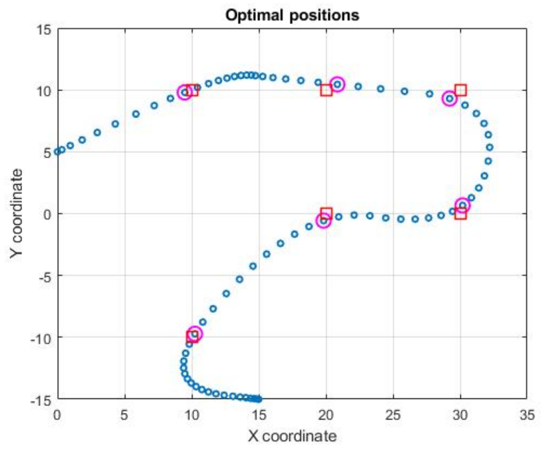


(a)

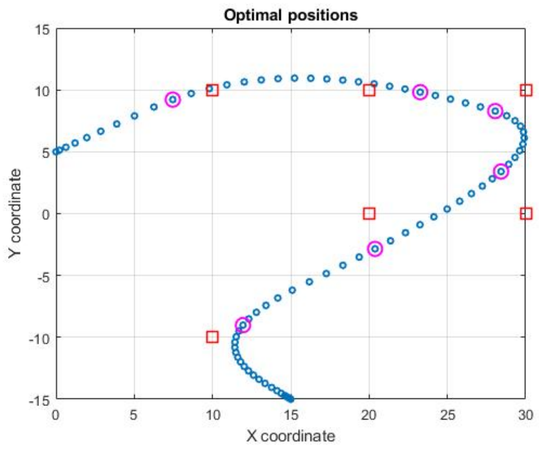


(b)

Figure 9: (a) $\lambda = 10^{-3}$; (b) $\lambda = 10^{-2}$

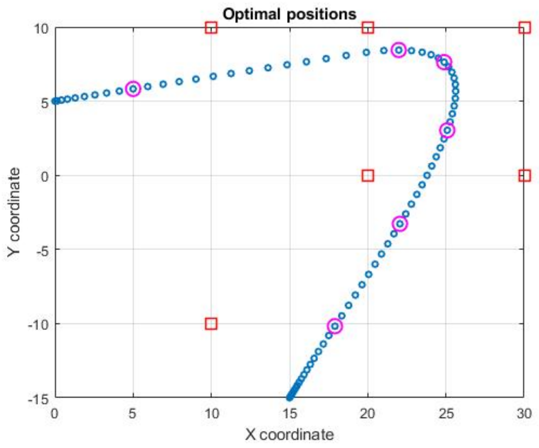


(a)

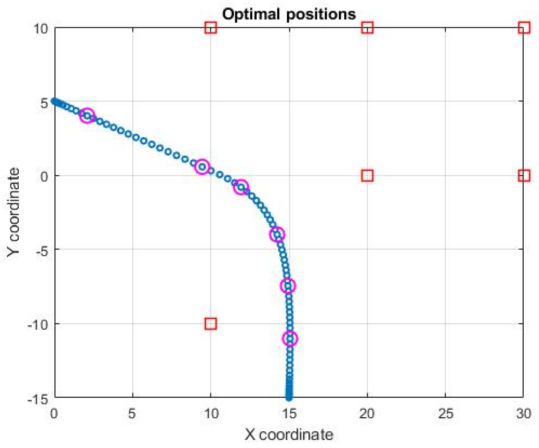


(b)

Figure 10: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$



(a)



(b)

Figure 11: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

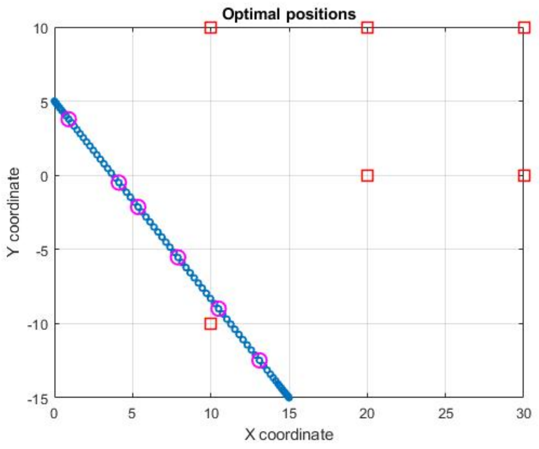
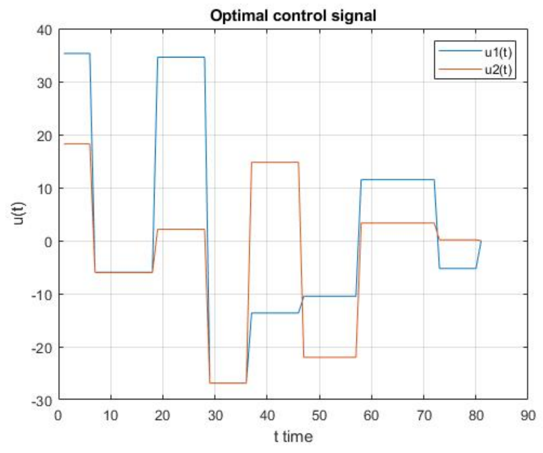


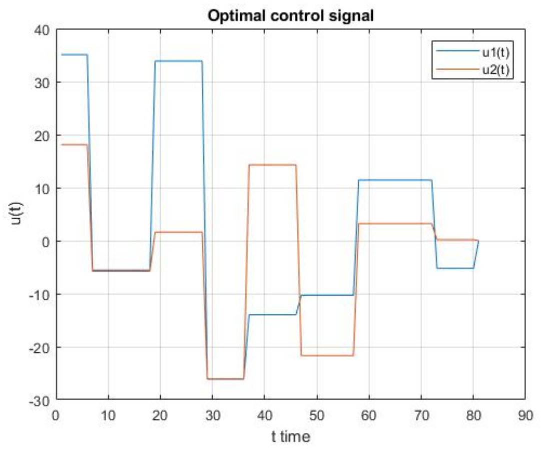
Figure 12: $\lambda = 10^3$

1.2.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$. We show that for all the λ in the figures 13, 14, 15 and 16.

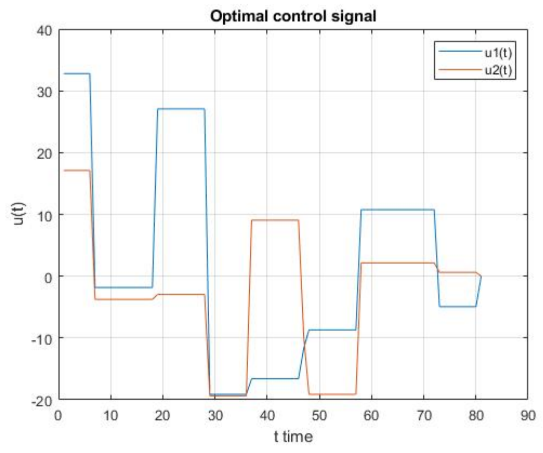


(a)

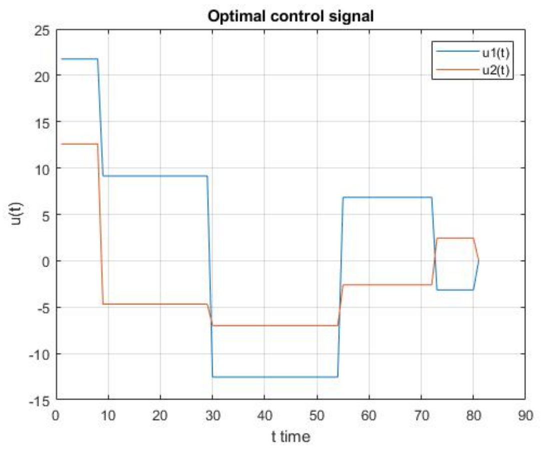


(b)

Figure 13: (a) $\lambda = 10^{-3}$; (b) $\lambda = 10^{-2}$



(a)

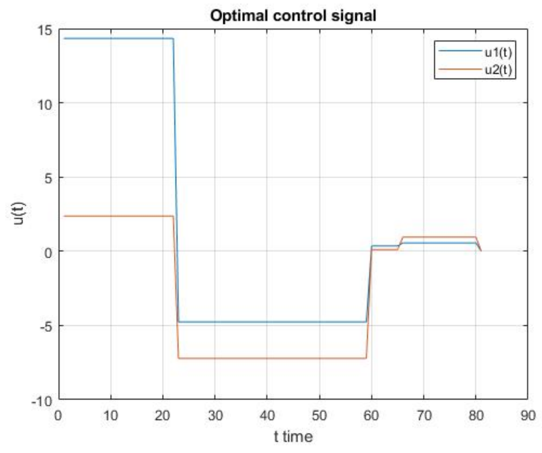


(b)

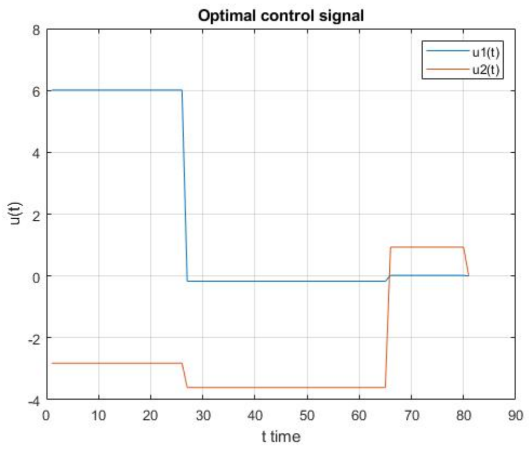
Figure 14: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$

1.2.3 Changes of the optimal control signal

In this subsection we report how many times the optimal control signal changes from $t = 1$ to $t = T - 1$.



(a)



(b)

Figure 15: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

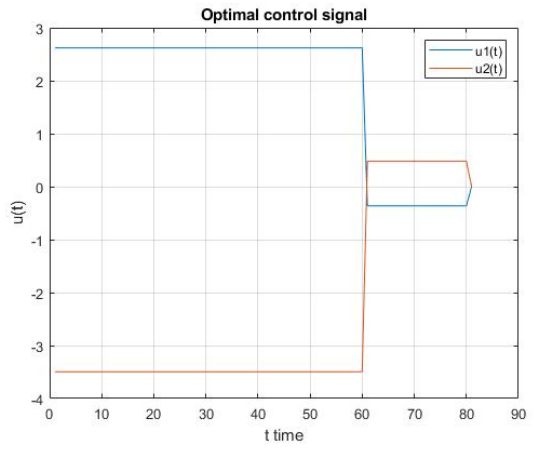


Figure 16: $\lambda = 10^3$

We say that the control signal changes at time t if $\|u(t) - u(t-1)\|_2 > 10^{-6}$. We show that for all λ in the table 3:

λ	Nº of times control signal changes
10^{-3}	10
10^{-2}	8
10^{-1}	11
10^{0}	4
10^{1}	4
10^{2}	4
10^{3}	1

Table 3: Number of times the control signal changes

1.2.4 Mean Deviation of the waypoints

In this subsection we report the mean deviation from the waypoints, defined as

$$\frac{1}{K}\sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2 \,, \tag{7}$$

which tells us how much the waypoints are ignored.

We show that for all the λ in the table 4:

λ	Mean deviation from the waypoints
10^{-3}	$0,0075$
10^{-2}	$0,0747$
10^{-1}	$0,7021$
10^{0}	$2,8876$
10^{1}	$5,3689$
10^{2}	$12,5914$
10^{3}	$16,2266$

Table 4: Mean deviation from the waypoints

1.3 Task 3 - The ℓ_1 regularizer

In this sections we were asked to solve the problem 8 using CVX. This problem is basically the same as the problem 6 but now we use a regularizer based on the ℓ_1 norm, that is shown in the equation bellow:

$$\|x\|_1 = |x_1| + \cdots + |x_n|.$$

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_1 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \le U_{\max}, \quad \text{for } 0 \le t \le T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \le t \le T-1. \end{array} \tag{8}$$

where $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}\}$.

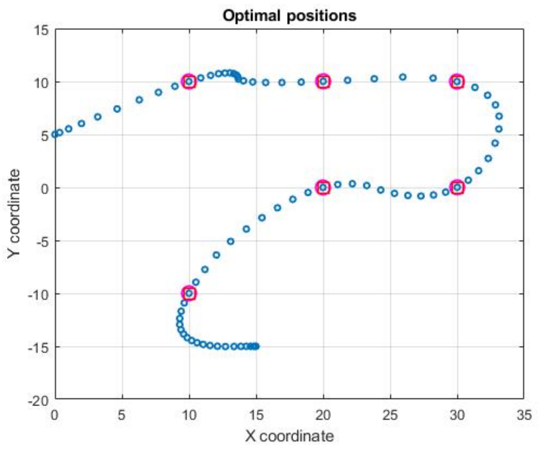
The code used for this task is listed bellow:

```
1  cvx_begin
2      variable x(4,T);
3      variable u(2,T);
4      F1=0;
5      F2=0;
6
7      for i=1:6
8          F1= F1 + square_pos(norm(x(1:2,tau(i))-w(:,i)));
9      end
10
11     for t=2:T-1
12         F2 = F2 + norm(u(:,t)-u(:,t-1),1);
13     end
14
15     minimize(F1 + lambda(7)*F2)
16
17     subject to
18         x(:,1) == [pinitial; 0; 0];
19         x(:,T) == [pfinal; 0; 0];
20
21         for i=1:T-1
22             norm(u(:,i))<=Umax;
23         end
24
25         for j=1:T-1
26             x(:,j+1)==A*x(:,j) + B*u(:,j);
27         end
28
29 cvx_end
```

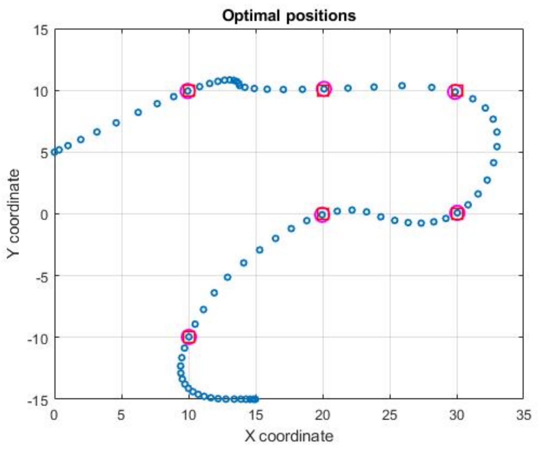
Computationally, to solve this problem we did exactly the same thing that we did in the problem 1 but, as we can see in the line 12 of the code, instead of the squared euclidean norm in the regularizer we have ℓ_1 norm. The solutions of this problem 8 are shown in the subsections bellow.

1.3.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red squares. We show that for all the λ in the figures 17, 18, 19 and 20.

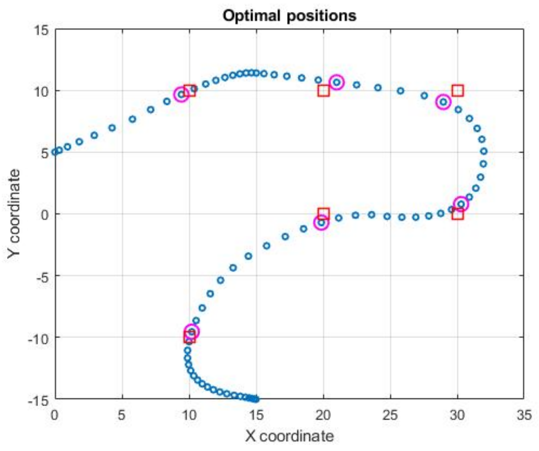


(a)

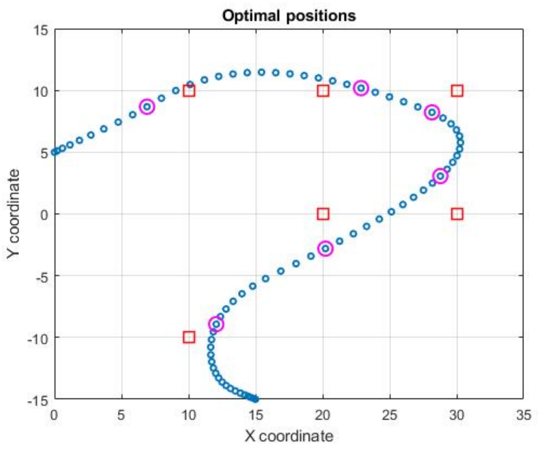


(b)

Figure 17: (a) $\lambda = 10^{-3}$; (b) $\lambda = 10^{-2}$

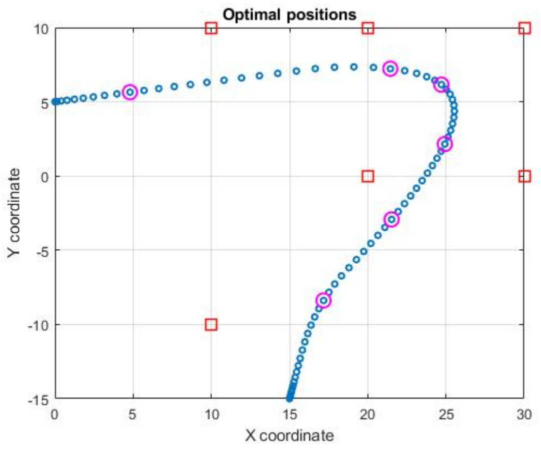


(a)

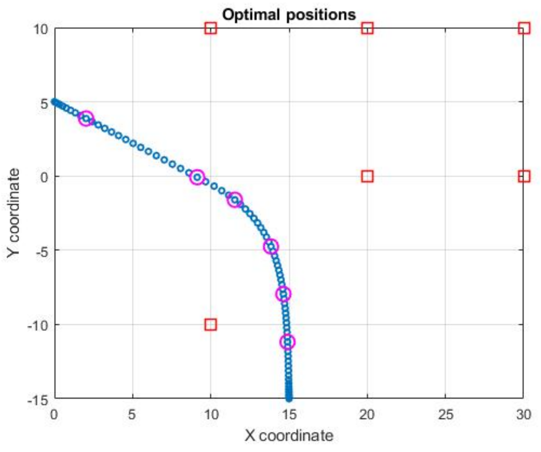


(b)

Figure 18: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$



(a)



(b)

Figure 19: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

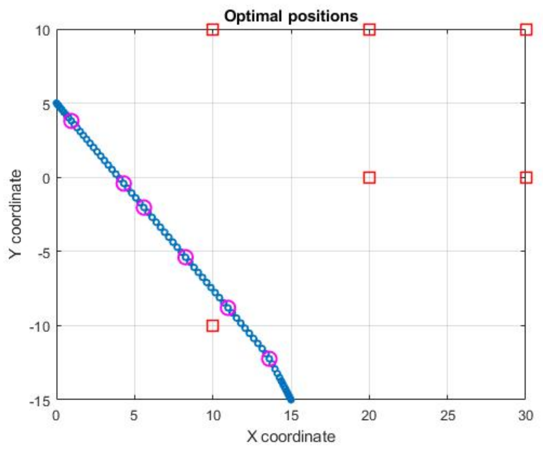
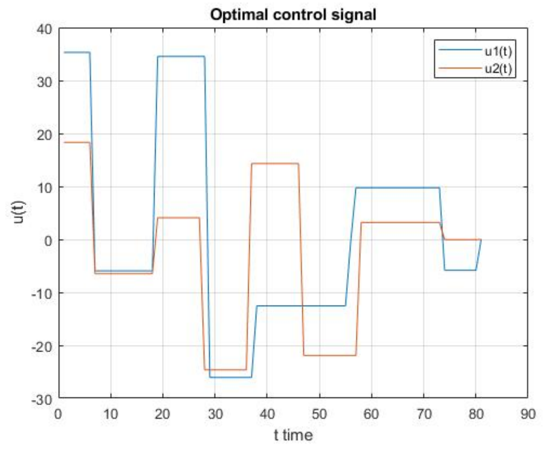


Figure 20: $\lambda = 10^3$

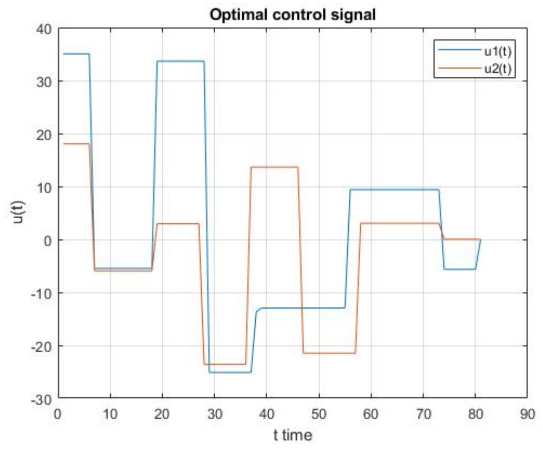
1.3.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$.

We show that for all the λ in the figures 21, 22, 23 and 24.

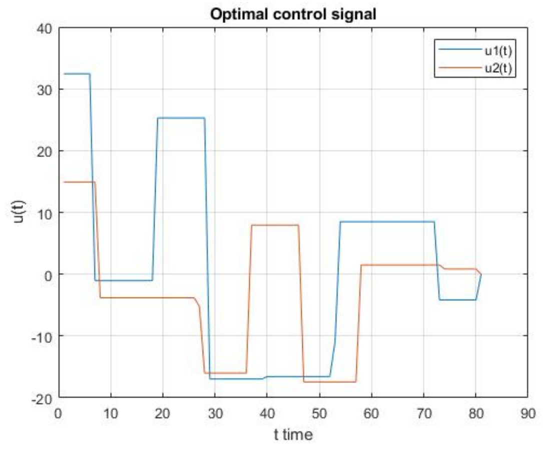


(a)

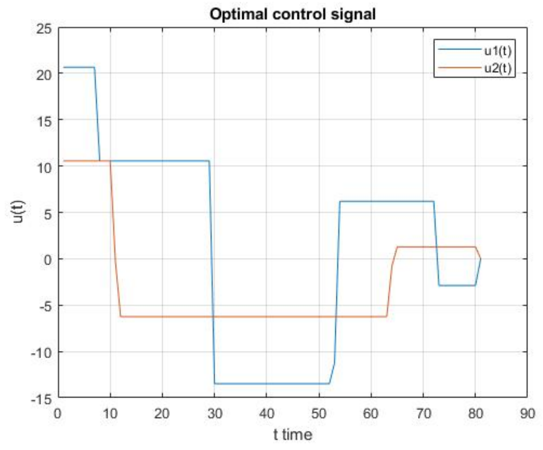


(b)

Figure 21: (a) $\lambda = 10^{-3}$; (b) $\lambda = 10^{-2}$



(a)

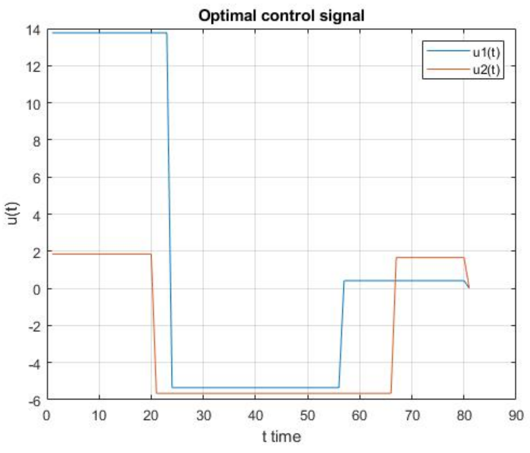


(b)

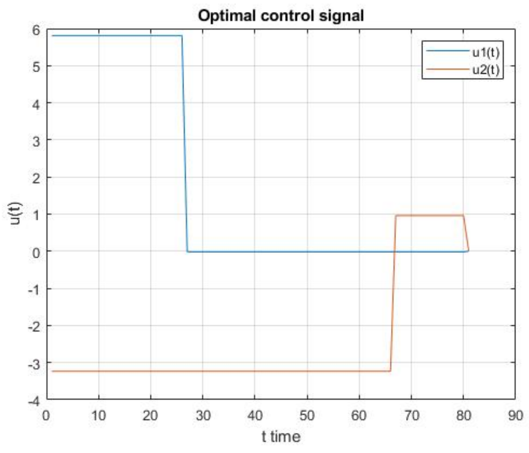
Figure 22: (a) $\lambda = 10^{-1}$; (b) $\lambda = 10^{0}$

1.3.3 Changes of the optimal control signal

In this subsection we report how many times the optimal control signal changes from $t = 1$ to $t = T - 1$.



(a)



(b)

Figure 23: (a) $\lambda = 10^1$; (b) $\lambda = 10^2$

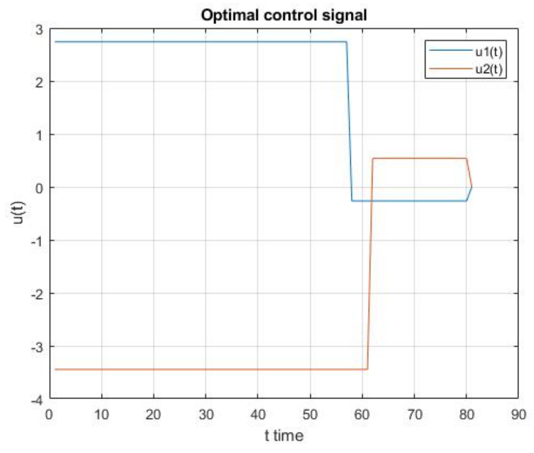


Figure 24: $\lambda = 10^3$

We say that the control signal changes at time t if $\|u(t) - u(t-1)\|_2 > 10^{-6}$. We show that for all λ in the table 5:

λ	Nº of times control signal changes
10^{-3}	13
10^{-2}	12
10^{-1}	14
10^{0}	11
10^{1}	5
10^{2}	3
10^{3}	2

Table 5: Number of times the control signal changes

1.3.4 Mean Deviation of the waypoints

In this subsection we report the mean deviation from the waypoints, defined as

$$\frac{1}{K}\sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2 \,, \tag{9}$$

which tells us how much the waypoints are ignored.

We show that for all the λ in the table 6:

λ	Mean deviation from the waypoints
10^{-3}	$0,0107$
10^{-2}	$0,1055$
10^{-1}	$0,8863$
10^{0}	$2,8732$
10^{1}	$5,4361$
10^{2}	$13,0273$
10^{3}	$16,0463$

Table 6: Mean deviation from the waypoints

1.4 Task 4 - Comments

From task 1 to task 3 we were asked to solve the same problem (the robot has to pass, as close as possible, to a series of intermediate points along its journey), but using different regularizers.

In task 1 we used a regularizer with the squared ℓ_2 norm in Ridge Regression (ℓ_2^2), as shown in equation 10:

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 \tag{10}$$

In task 2 we used a regularizer with the ℓ_2 norm in the Ridge Regression, as shown in equation 11:

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2 \tag{11}$$

In the task 3 we used a regularizer with the ℓ_1 norm in Lasso Regression, as shown in the equation 12:

$$\lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_1 \tag{12}$$

Looking at the results we can see that, as we increase the λ, the mean deviation from the waypoints also increases. We can see that in the table 2, 4 and 6. That is transversal to all the regularizers and happens because, if we increase the λ, the right side of the cost function also increases and the optimization algorithm gives more importance in keeping the control signal constant and forgets about the waypoints, making the path much more direct to the final position.

We can also see that, for the same values of λ, the problem 1 (ℓ_2^2 regularizer) has a much more frequent change of the optimal control signal. As we know, the ℓ_2^2 regularizer squares the difference between successive control inputs. That is, the algorithm penalizes more abrupt changes and, consequently, does not let the difference between successive control inputs be high. We can see that in the figures 5, 6, 7 and 8. The control input does not have abrupt changes because it is penalized and, consequently, the optimal control signal have more frequent changes, comparing to the other regularizers, as we can see in the table 1.

Contrary to the ℓ_2^2 regularizer, ℓ_2 and ℓ_1 regularizers do not penalize as much the abrupt changes between successive control inputs because there is no square in the difference. We can see in the figures 13, 14, 15, 16, 21, 22, 23 and 24 that there are much more abrupt changes in the optimal control signal and, consequently, less frequent changes as we can see in the tables 3 and 5.

1.5 Task 5

In this task and in the following one we were asked to solve the same problem but now, instead of passing, as close as possible, to certain intermediate points along its journey, the robot as to pass, as close as possible to certain intermediate regions. So the optimization problem is explained in the equation 13 bellow:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} d\left(Ex(\tau_k), D(c_k, r_k)\right)^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \le U_{\max}, \quad \text{for } 0 \le t \le T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \le t \le T-1. \end{array} \tag{13}$$

where $D(c_k, r_k)$ is the disk with center c and radius r.

Particularly, in this task we were asked to give a closed-form expression for $d(p, D(c, r))$, using simple geometric arguments.

If the circle is not centered at the origin but has a center (c_1, c_2) and a radius r , the shortest distance between the point (p_1, p_2) and the circle is given by the equation 14:

$$\sqrt{(p_1 - c_1)^2 + (p_2 - c_2)^2} - r \tag{14}$$

This is the function we want to minimize, because we want to find the points (p_1, p_2) that pass, as close as possible, to a certain region that, in this case, is given by a circle.

1.6 Task 6

Now that we have the closed-form of the distance from a point to a disk we can write the problem 13 in the form bellow:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \left(\|Ex(\tau_k) - c_k\|_2 - r_k\right)^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \le U_{\max}, \quad \text{for } 0 \le t \le T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \le t \le T-1. \end{array} \tag{15}$$

The code used for this task is listed bellow:

```
cvx_begin
    variable x(4,T);
    variable u(2,T);
    F1=0;
    F2=0;

    for i=1:K
        F1= F1 + square_pos(max(norm(x(1:2,tau(i))-c(:,i))- r, 0));
    end

    for t=2:T-1
        F2 = F2 + norm(u(:,t)-u(:,t-1));
    end
```

```
14
15        minimize(F1 + lambda(3)*F2)
16
17        subject to
18            x(:,1) == [pinitial; 0; 0];
19            x(:,T) == [pfinal; 0; 0];
20
21            for i=1:T-1
22                norm(u(:,i))<=Umax;
23            end
24
25            for j=1:T-1
26                x(:,j+1)==A*x(:,j) + B*u(:,j);
27            end
28
29    cvx_end
```

Computationally, to solve this problem we did exactly the same thing that we did in the problem 6 but, as we can see in the line 8 of the code, instead of optimizing the distance between the robot x_k and the waypoint w_k at time τ_k, the optimization is between the robot and a disk with radius of 2. The solutions of this problem 15 are shown in the subsections bellow.

1.6.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red circles.

We show that just for $\lambda = 10^{-1}$ in the figure 46.

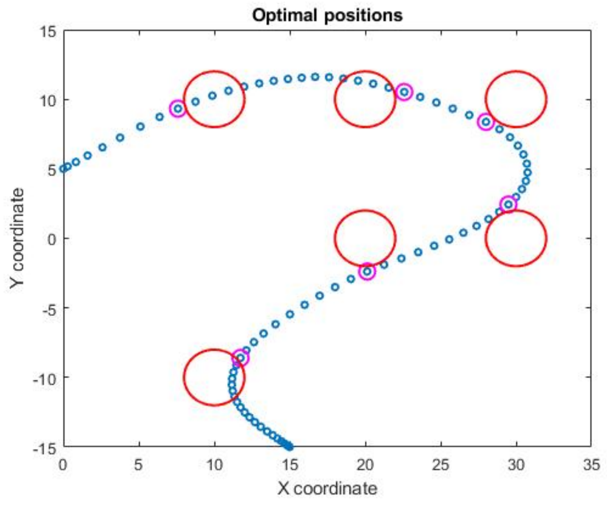


Figure 25: Optimal Positions of the robot for $\lambda = 10^{-1}$.

1.6.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$.

We show that just for $\lambda = 10^{-1}$ in the figure 47.

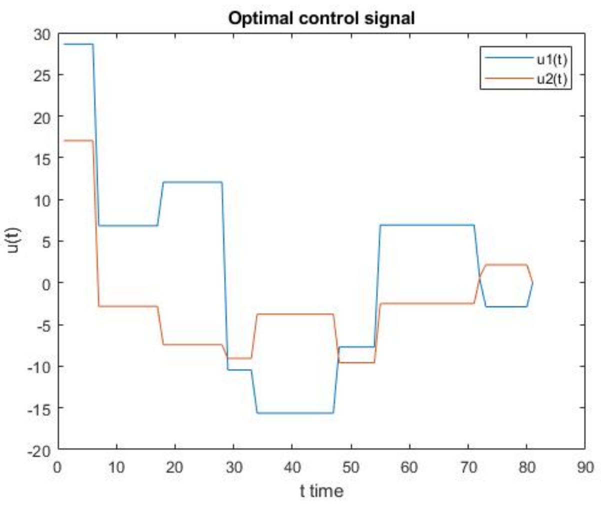


Figure 26: Optimal control signal for $\lambda = 10^{-1}$.

1.6.3 Changes of the optimal control signal

In this subsection we report how many times the optimal control signal changes from $t = 1$ to $t = T - 1$.

We say that the control signal changes at time t if $\|u(t) - u(t-1)\|_2 > 10^{-6}$. We show that just for $\lambda = 10^{-1}$ in the table 7:

λ	Nº of times control signal changes
10^{-1}	8

Table 7: Number of times the control signal changes for $\lambda = 10^{-1}$.

1.6.4 Mean Deviation of the waypoints

In this subsection we report the mean deviation from the waypoints, defined as

$$\frac{1}{K} \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2 , \tag{16}$$

which tells us how much the waypoints are ignored.

We show that for all the λ in the table 8:

λ	Mean deviation from the waypoints
10^{-1}	$2,4690$

Table 8: Mean deviation from the waypoints

1.6.5 Comments

Comparing this problem with the problem 6, we can see that the control signal in this problem has less abrupt changes than the problem of task 2. That is because in task 2 the robot had to pass, as close as possible to the waypoints, and now it has to pass, as close as possible, to a disk with a radius of 2. In this the second case the optimization is much more relaxed because the robot can pass anywhere near the region and not strictly near the point and, as consequence, it does not need to have such abrupt changes in the control signal to pass near the waypoints, like in problem 6.

1.7 Task 7

From this task until the end of part 1, we were asked to solve the problem similar to the problem of Section 2, when the robot had to pass approximately on certain waypoints, but now we want the robot to pass exactly on waypoints. That is, we want $p(\tau_k) = w_k$ for as many waypoints as possible.

In particular, for this task we were asked to solve the optimization problem 17 bellow:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & 0 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1, \\ & Ex(\tau_k) = w_k, \quad \text{for } 1 \leq k \leq K. \end{array} \tag{17}$$

The code used for this task is listed bellow:

```
cvx_begin
    variable x(4,T);
    variable u(2,T);

    minimize(0)

    subject to
        x(:,1) == [pinitial; 0; 0];
        x(:,T) == [pfinal; 0; 0];

        for i=1:T-1
```

```
12              norm(u(:,i))<=Umax;
13          end
14
15          for j=1:T-1
16              x(:,j+1)==A*x(:,j) + B*u(:,j);
17          end
18
19          for k=1:K
20              x(1:2,tau(k))==w(:,k);
21          end
22
23  cvx_end
```

In this task we don't have the regularizer anymore, that made sure the robot was making simple movements, and we don't want to minimize any function. Instead we transfer the function $Ex(\tau_k) = w_k$, that we wanted to optimize, to the constraints, as we can see in the line 20 in the code, and we put the function 0 as the function that we want to optimize, as we can see in the line 5 of the code.

Using the software CVX to solve this problem, MATLAB gives us an Infeasible Status. That means that the algorithm cannot make sure that the robot $x(\tau_k)$ passes in all the waypoints at all the times τ_k. That is because we are not minimizing any function and the software cannot find an optimal x and u that satisfies all the constraints.

1.8 Task 8

In this task we were asked to prove that the function ϕ is non-convex:

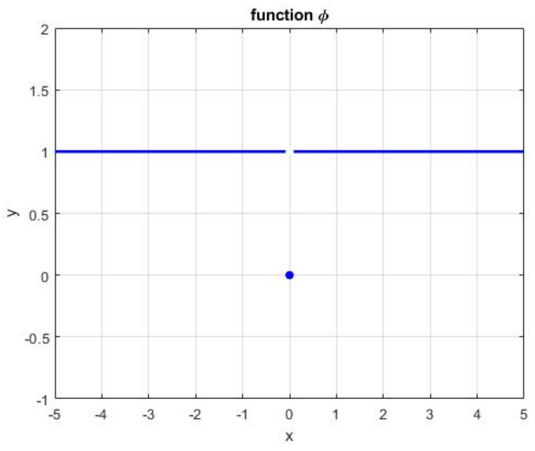
$$\phi(x) = \begin{cases} 0, & \text{if } x = 0 \\ 1, & \text{if } x \neq 0 \end{cases} \tag{18}$$

In mathematics, a real-valued function defined on an n-dimensional interval is called convex if the line segment between any two points on the graph of the function lies above or on the graph, in a Euclidean space (or more generally a vector space) of at least two dimensions.

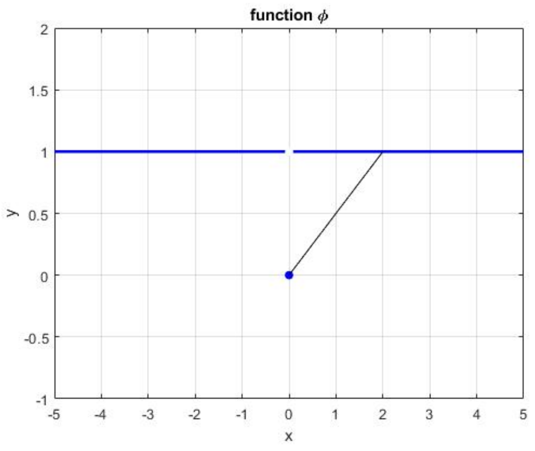
As we can see in the figure 27, if we connect any point of the graph to the point (0,0) we get a line segment that lies bellow the graph. So, from this, we can conclude that the function $\phi(x)$ is non-convex.

1.9 Task 9 - A ℓ_2^2 formulation

In this section we were asked to solve the following optimization problem:



(a)



(b)

Figure 27: (a) Function $\phi(x)$; (b) Function $\phi(x)$ with line segment

$$
\begin{array}{ll}
\underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 \\
\text{subject to} & x(0) = x_{\text{initial}} \\
& x(T) = x_{\text{final}} \\
& \|u(t)\|_2 \leq U_{\max}, \quad \text{for } 0 \leq t \leq T-1 \\
& x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1.
\end{array} \tag{19}
$$

The code used for this task is listed bellow:

```
cvx_begin
    variable x(4,T);
    variable u(2,T);
    F=0;

    for k=1:K
        F = F + square_pos(norm(x(1:2,tau(k))-w(:,k)));
    end

    minimize(F)

    subject to
        x(:,1) == [pinitial; 0; 0];
        x(:,T) == [pfinal; 0; 0];

        for i=1:T-1
            norm(u(:,i))<=Umax;
        end

        for j=1:T-1
```

```
21                  x(:,j+1)==A*x(:,j) + B*u(:,j);
22              end
23  cvx_end
```

In this task we solved the same problem as in task 1 but without the regularizer, that made sure the robot was making simple movements, as we can see in the line 7 of the code. The solutions for this task are presented in the subsections bellow.

1.9.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red squares. The plot that shows the positions of the robot is in figure 28.

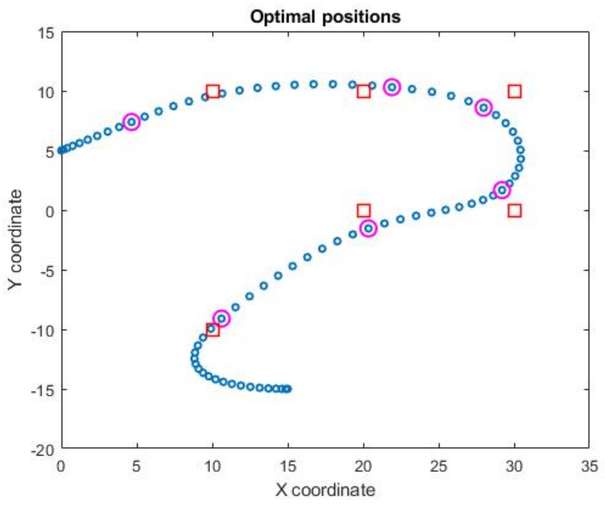


Figure 28: Optimal Positions of the robot.

1.9.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$. The plot that shows the optimal control signal is in figure 29.

1.9.3 How many waypoints are captured

In this section we were asked to report how many waypoints are captured by the robot considering that a waypoint w_k is captured if the robot is sufficiently close to it at the appointed time τ_k, namely, if $\|p(\tau_k) - w_k\|_2 \leq 10^{-6}$.

As we can see in the figure 28, and later concluded by the MATLAB code, the robot does not capture any of the waypoints.

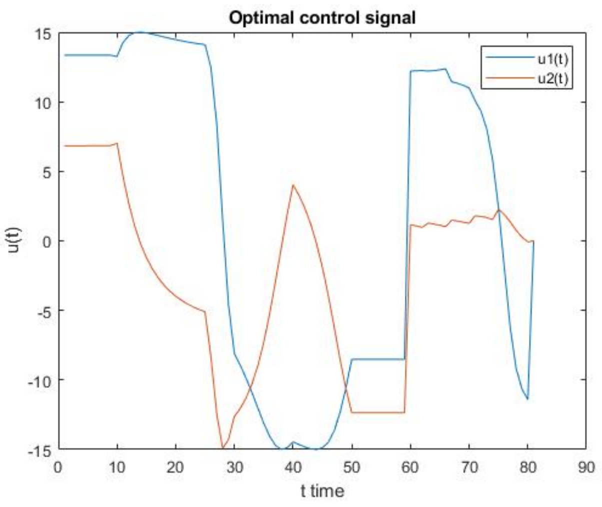


Figure 29: Optimal control signal

1.10 Task 10 - A ℓ_2 formulation

In this section we were asked to solve the following optimization problem:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & \|u(t)\|_2 \le U_{\max}, \quad \text{for } 0 \le t \le T-1 \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \le t \le T-1. \end{array} \tag{20}$$

The code used for this task is listed bellow:

```
cvx_begin
    variable x(4,T);
    variable u(2,T);
    F=0;

    for k=1:K
        F = F + norm(x(1:2,tau(k))-w(:,k));
    end

    minimize(F)

    subject to
        x(:,1) == [pinitial; 0; 0];
        x(:,T) == [pfinal; 0; 0];

        for i=1:T-1
            norm(u(:,i))<=Umax;
```

```
18              end
19
20              for j=1:T-1
21                  x(:,j+1)==A*x(:,j) + B*u(:,j);
22              end
23  cvx_end
```

In this task we solved the same problem as in task 9 but now the norm of the function that we want to optimize is the euclidean norm, instead of the squared euclidean norm, as we can see in the line 7 of the code. The solutions for this task are presented in the subsections bellow.

1.10.1 Optimal Positions of the Robot

In this subsection we show the positions of the robot from $t = 0$ to $t = T$ in the small blue circles; the positions at appointed times τ_k, for $1 \leq k \leq K$, are showed in the large magenta circles. The waypoints are showed in the red squares.

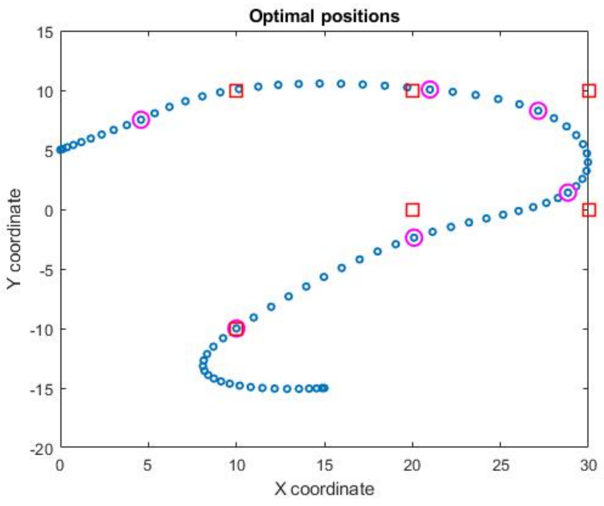


Figure 30: Optimal Positions of the robot.

1.10.2 Optimal Control Signal

In this section we show the two components, $u_1(t)$ and $u_2(t)$, of the control signal $u(t)$ from $t = 0$ to $t = T - 1$.

1.10.3 How many waypoints are captured

In this section we were asked to report how many waypoints are captured by the robot considering that a waypoint w_k is captured if the robot is sufficiently close to it at the appointed time τ_k, namely, if $\|p(\tau_k) - w_k\|_2 \leq 10^{-6}$.

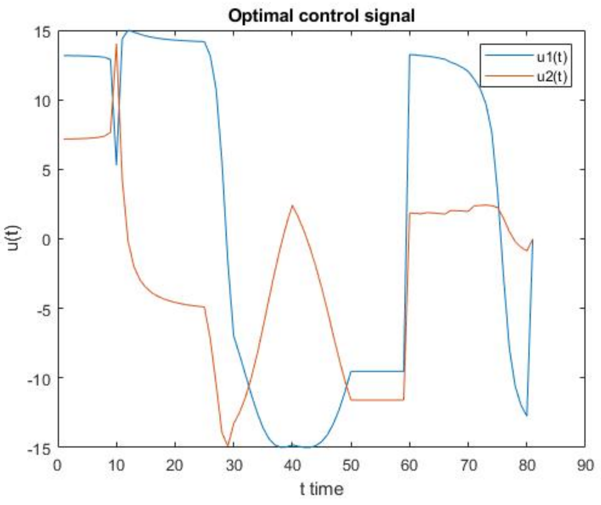


Figure 31: Optimal control signal

As we can see in the figure 30, and later concluded by the MATLAB code, the robot only capture one of the waypoints.

1.10.4 Comments and comparisons between task 9 and task 10

Looking at the plots of the positions of the robot and the control signal of the task 9 and task 10 we can see that, in comparison with the problems where we had the regularizer (problems 1, 6 and 8), the control signal appears to be much more unstable. That is due to the absence of the regularizer that was responsible for the simple control signal, that is, responsible for the control signal to be constant, or, at least, to have small differences.

Comparing also the problem 19 and 20 (task 9 and task 10) we can see in the equations that the only thing that differs is the norm of the function. In task 9 the function has the squared euclidean norm and in task 10 it has the normal euclidean norm. There is not much difference between the plots of these two tasks, but in problem 19, that has the squared norm, the algorithm does not capture any waypoint in the path of the robot and in problem 20, that has the normal euclidean norm, the algorithm capture one waypoint in the path of the robot. That is due to, when the algorithm is optimizing the problem, the differences $Ex(\tau_k) - w_k$ that are already small it become smaller with the squared norm and the algorithm in task 9 never reaches a point where the position of the robot x_k and the waypoint w_k at time τ_k are in the same spot and in task 10 it does.

1.11 Task 11

In this section we were asked to solve the optimization problem stated bellow:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \frac{1}{\left\|Ex^{(m)}(\tau_k)-w_k\right\|_2+\epsilon} \left\|Ex(\tau_k)-w_k\right\|_2 \\ \text{subject to} & x(0)=x_{\text{initial}} \\ & x(T)=x_{\text{final}} \\ & \left\|u(t)\right\|_2 \le U_{\max}, \quad \text{for } 0 \le t \le T-1 \\ & x(t+1)=Ax(t)+Bu(t), \quad \text{for } 0 \le t \le T-1. \end{array} \tag{21}$$

This problem is similar to the one in task 10 but with an iterative technique for improving on the results that consists in solving reweighted versions of the problem.

Specifically, each iteration of the technique takes a current guess $x^{(m)}$ and $u^{(m)}$ and produces a new one $x^{(m+1)}$ and $u^{(m+1)}$.

Solving (21) gives $x^{(m+1)}$ and $u^{(m+1)}$. The technique is initialized with $x^{(0)}$ and $u^{(0)}$, the solution of problem of the task 10.

The code used for this task is listed bellow:

```
for i=1:M
    ind(i).x=zeros(size(x_task10));
end

ind(1).x=x_task10;

for i=1:M
    ind(i).u=zeros(size(u_task10));
end

ind(1).u=u_task10;

for m=1:M

    cvx_begin
        variable x(4,T);
        variable u(2,T);
        F=0;

        for k=1:K
            F = F +  (1/(norm(ind(m).x(1:2,tau(k))-w(:,k))+epsilon))*
            *norm(x(1:2,tau(k))-w(:,k));
        end

        minimize(F)

        subject to
            x(:,1) == [pinitial; 0; 0];
            x(:,T) == [pfinal; 0; 0];

            for i=1:T-1
                norm(u(:,i))<=Umax;
```

```
33              end
34
35              for j=1:T-1
36                  x(:,j+1)==A*x(:,j) + B*u(:,j);
37              end
38      cvx_end
39
40      ind(m+1).x=x;
41      ind(m+1).u=u;
42  end
```

In this task we were asked to solve the same problem as in the task 10 but now we have a parcel in the equation that we want to optimize that is responsible for the weighting throughout the iterations.

In this case the number of iterations to do the reweighting is 10. And in every iteration the solution has to be saved to be entered in the weighted parcel of the function in the next iteration, so we saved every solution of the problem in two structures (ind.x for the positions of the robot and ind.u for the control signal), as we can see in the lines 2 and 8 of the code. The initialization for this structures is the solution of the problem 20, as we can see in the lines 5 and 11 of the code.

Next we ran the CVX for every iteration of the reweighting technique with a cycle *for*, as we can see in the line 13 of the code.

The solutions for this task are presented in the subsections bellow.

1.11.1 Positions of the robot

In the section we were asked to plot the positions of the robot in $x^{(m)}$ from $t = 0$ to $t = T$, marking its positions at the appointed times τ_k, for $1 \leq k \leq K$. We show the positions of the robot for every m in the figures 32, 33, 34, 35, 36.

1.11.2 Control signal

In this section we were asked to plot the control signal $u^{(m)}(t)$, where $u^{(m)}(t) = \left(u_1^{(m)}(t), u_2^{(m)}(t)\right)$, from $t = 0$ to $t = T - 1$.

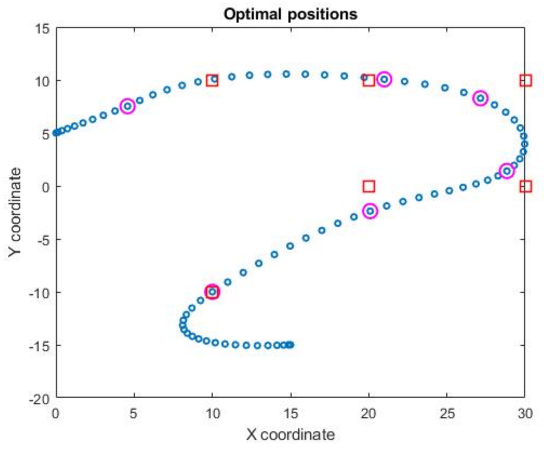
We show the positions of the robot for every m in the figures 37, 38, 39, 40, 41.

1.11.3 Waypoints that are captured

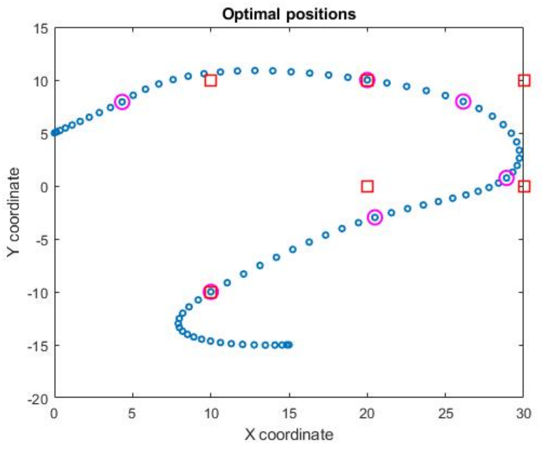
In this sections we were asked report how many waypoints were captured by the robot. We show the number of waypoints captured for every m in the table 9.

1.12 Task 12

The role of the weights in problem 21 is to approximate the position of the robot at times τ_k to the waypoints w_k. The fraction $\frac{1}{\left\|Ex^{(m)}(\tau_k)-w_k\right\|_2+\epsilon}$ uses x^m that is the solution of the

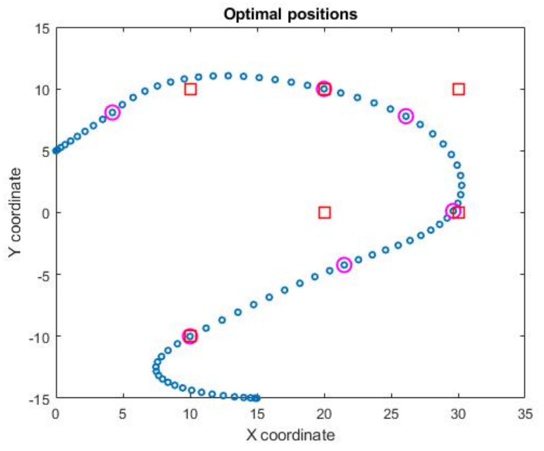


(a)

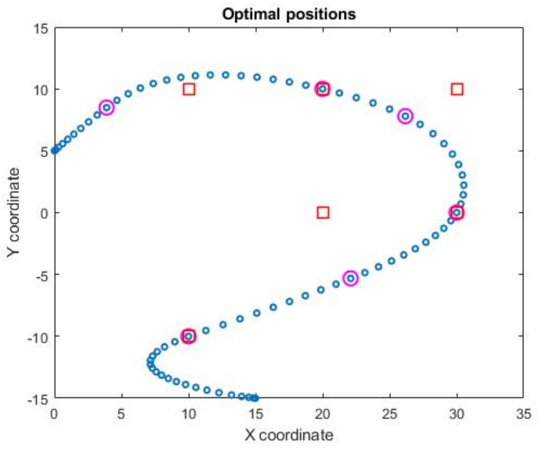


(b)

Figure 32: (a) $m = 0$; (b) $m = 1$

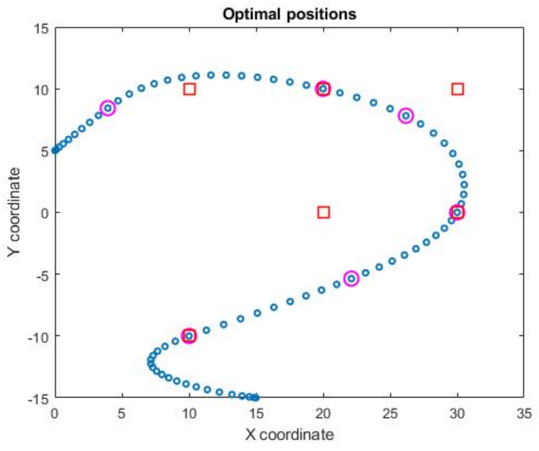


(a)

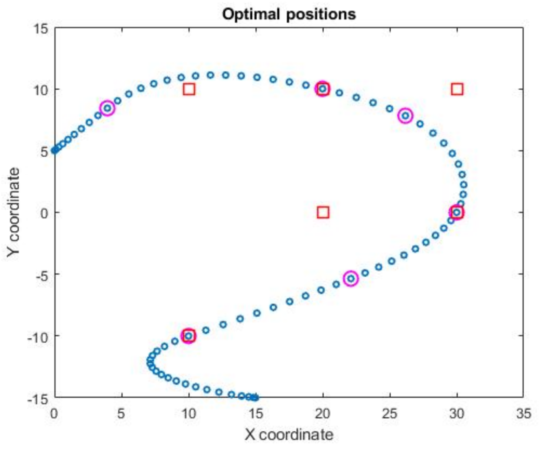


(b)

Figure 33: (a) $m = 2$; (b) $m = 3$

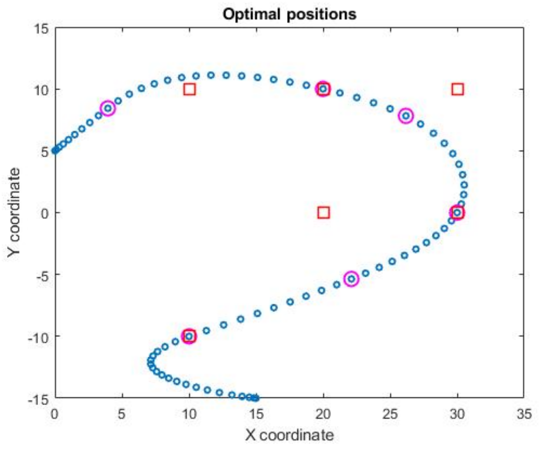


(a)

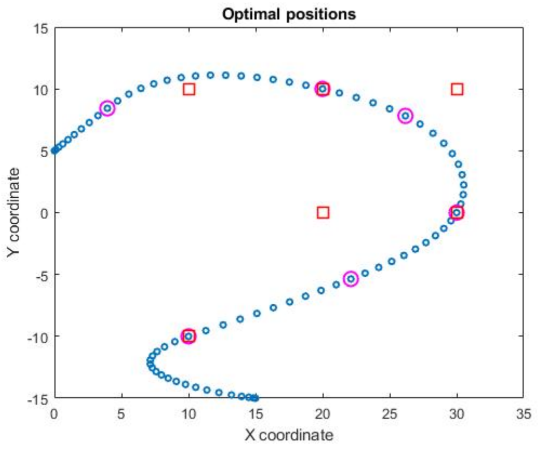


(b)

Figure 34: (a) $m = 4$; (b) $m = 5$

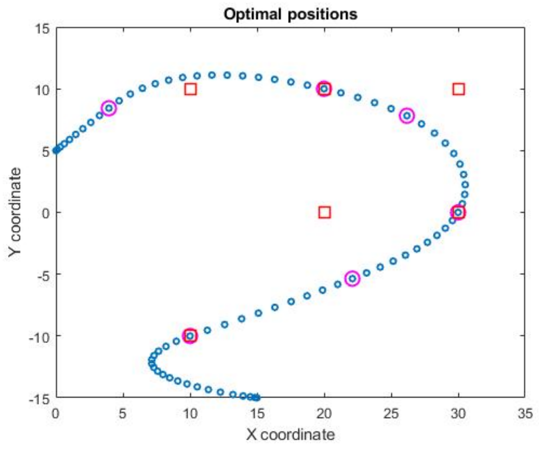


(a)

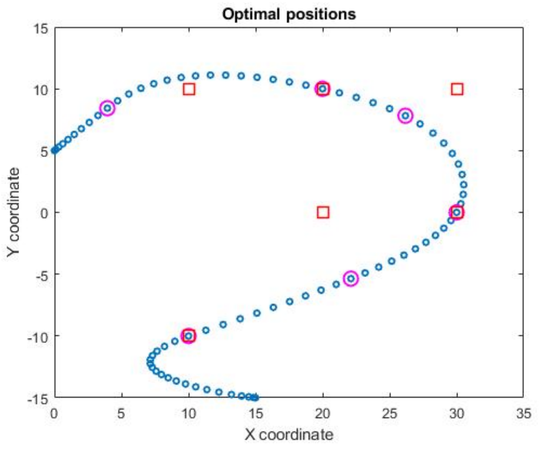


(b)

Figure 35: (a) $m = 6$; (b) $m = 7$

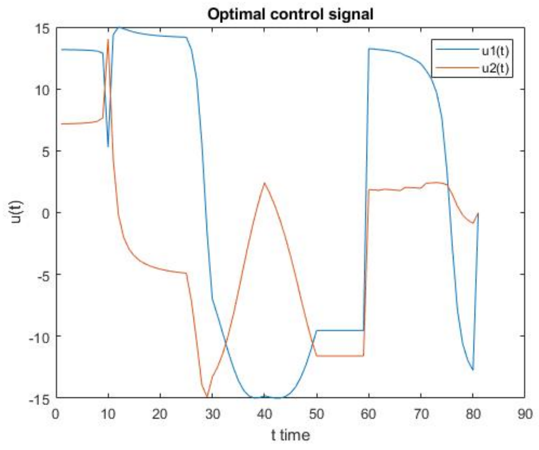


(a)

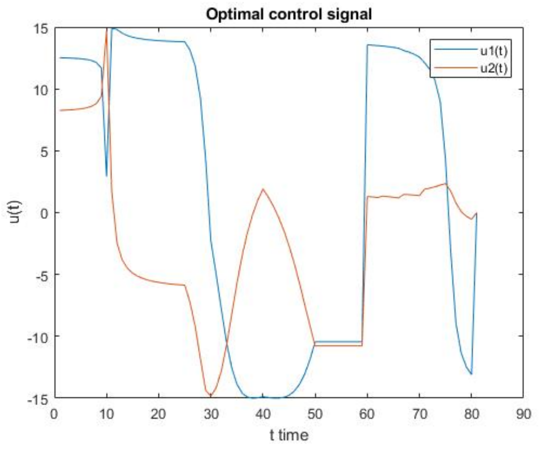


(b)

Figure 36: (a) $m = 8$; (b) $m = 9$

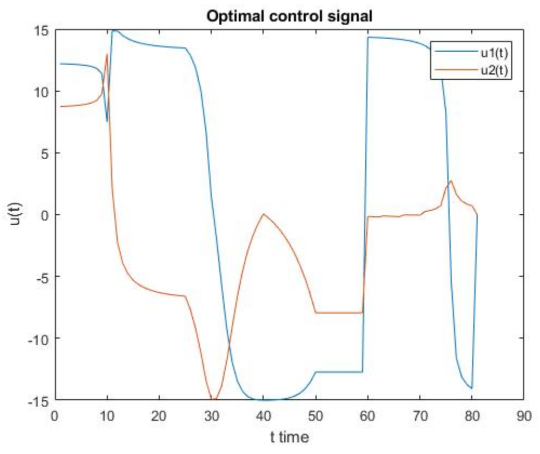


(a)

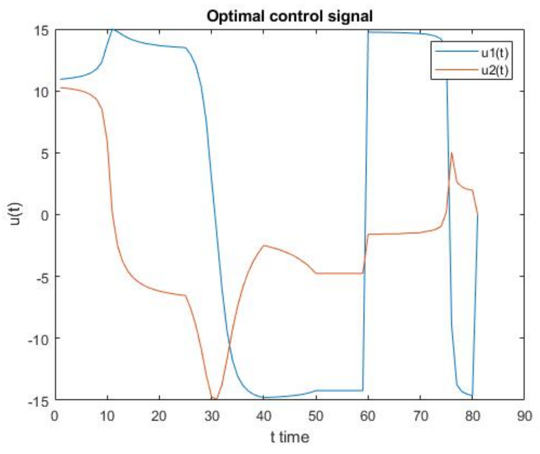


(b)

Figure 37: (a) $m = 0$; (b) $m = 1$

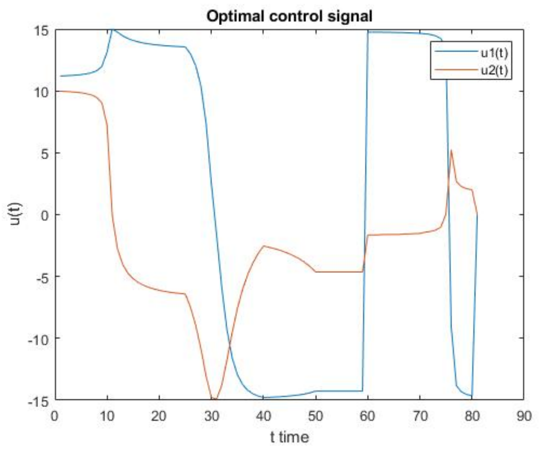


(a)

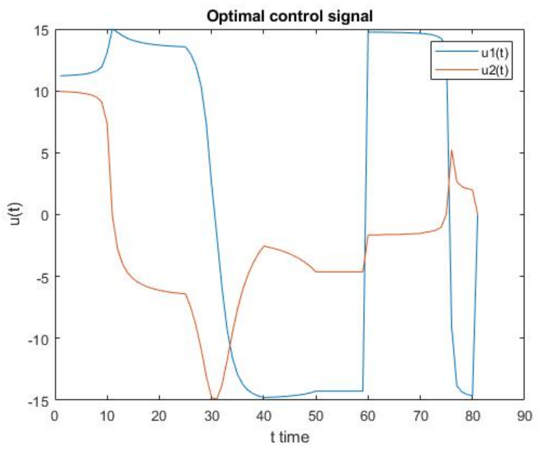


(b)

Figure 38: (a) $m = 2$; (b) $m = 3$

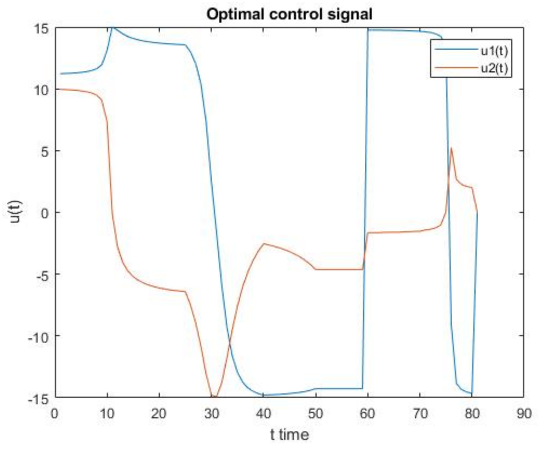


(a)

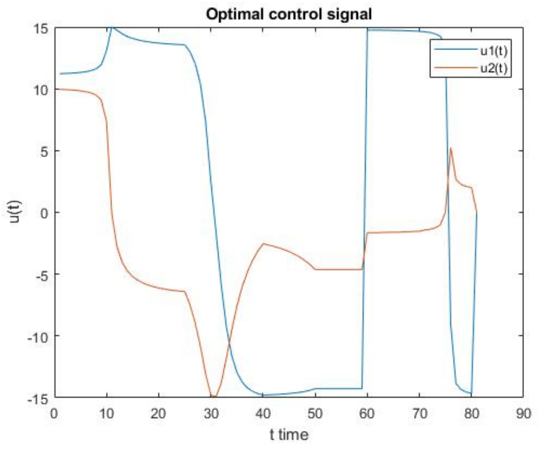


(b)

Figure 39: (a) $m = 4$; (b) $m = 5$

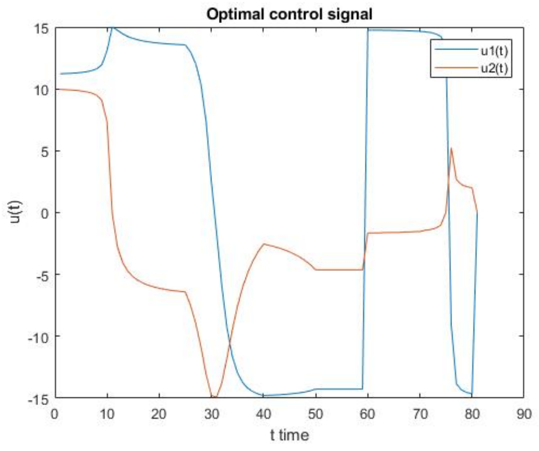


(a)

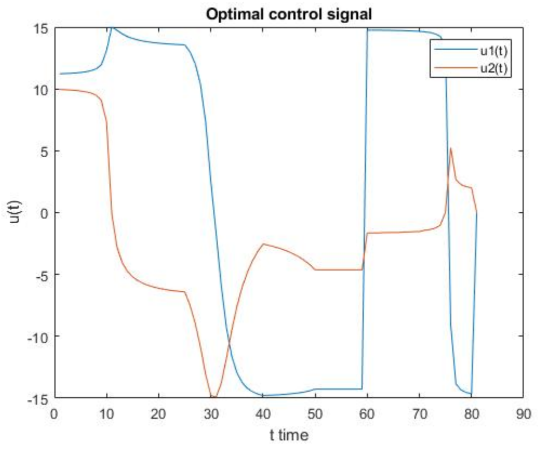


(b)

Figure 40: (a) $m = 6$; (b) $m = 7$



(a)



(b)

Figure 41: (a) $m = 8$; (b) $m = 9$

m	Waypoints Captured
0	1
1	2
2	2
3	3
4	3
5	3
6	3
7	3
8	3
9	3

Table 9: Mean deviation from the waypoints

previous iteration of the reweighting technique. This technique gives more importance to the waypoints w_k that are already close to the positions $x_k(\tau_k)$ of the robot. So, how does this work?

If the difference between the position of the robot and the waypoint in the previous iteration of the reweighting technique ($\left\|Ex^{(m)}(\tau_k) - w_k\right\|_2$) is big, the fraction $\frac{1}{\left\|Ex^{(m)}(\tau_k)-w_k\right\|_2+\epsilon}$ is small and the algorithm penalizes less the difference between the position of the robot x_k and the waypoint w_k at time τ_k.

Contrary to the previous case, if the difference between the position of the robot and the waypoint in the previous iteration of the reweighting technique ($\left\|Ex^{(m)}(\tau_k) - w_k\right\|_2$) is small, the fraction $\frac{1}{\left\|Ex^{(m)}(\tau_k)-w_k\right\|_2+\epsilon}$ is big and the algorithm penalizes more the difference between the position of the robot x_k and the waypoint w_k at time τ_k.

These two cases cause the algorithm to approximate even more the positions that are already close to the waypoints. That can be seen in figures 32, 33, 34, 35, 36 where the positions of the robot that are already close to the waypoints get even closer through the iterations until they are in the same point, and the ones that are too far away stay even further away from the waypoints. That's why the algorithm cannot capture more than three waypoints.

The way that we solved this was to put a structure that saves the solution for every iteration ($x^{(m)}$ and $u^{(m)}$) of the reweighting technique and we ran the program M times. In this problem we used $x^{(m)}$ as the solution of the of the previous iteration.

2 Part 2

In this part of the project we looked at two different problems. The first one it is called Logistic Regression. Logistic Regression is a statistical technique that aims to produce, from a set of observations, a model that allows the prediction of values taken by a categorical variable, often binary, from a series of continuous/binary explanatory variables. That is, a given x creates a probability distribution for the label y, thus making the label y either more likely to be 0 or 1. Specifically, for a given x, logistic regression assigns the probability of y being 1 to be equal to $\exp(s^T x - r)/(1 + \exp(s^T x - r))$ and the probability of y being 0 to be equal to $1/(1 + \exp(s^T x - r))$. In sum, when a vector of measurements x verifies $s^T x - r > 0$, its label y is more likely to be 1. Contrary to the previous case, if the when a vector of measurements x verifies $s^T x - r < 0$, its label y is more likely to be 0. In order to find s and r that match our dataset we have to solve the the following optimization problem:

$$\underset{s,r}{\text{minimize}} \quad \tfrac{1}{K} \textstyle\sum_{k=1}^{K} \log(1 + \exp(s^T x_k - r)) - y_k(\exp(s^T x_k - r)) \tag{22}$$

In this part we used two distinct approaches, explained in the following sections, to solve this problem: the Gradient Method and the Newton Method.

The second problem we solved it is called Network Localization. Particularly, in this part of the project we focused in finding the positions of several sensors, given noisy measurements of the distance between some of them and some anchors (sensors with known positions) and between some of the sensors.To find those positions we had so solve the following optimization problem:

$$\underset{x}{\text{minimize}} \quad \textstyle\sum_{(m,p)\in A} (\|a_m - s_p\| - y_{mp})^2 + \sum_{(p,q)\in S} (\|s_p - s_q\| - z_{pq})^2 \tag{23}$$

where A is the set of pairs (anchor, sensor) for which we have noisy measurements of their distance, and S is the set of pairs (sensor, sensor) for which we have noisy measurements of their distance.

The tasks related to the Gradient Method are the tasks 2 to 4. The ones related to the Newton Method are the tasks 5 to 7. Finally, the ones related to the Network Localization are the tasks 8 and 9.

2.1 Task 1 - Gradient Method

This task aims to prove that the function 24 is convex:

$$f(s,r) = \frac{1}{K} \sum_{k=1}^{K} \log(1 + \exp(s^T x_k - r)) - y_k(s^T x_k - r) \tag{24}$$

As we can see, the f function can be decomposed in a sum of two functions:

$$f = \frac{1}{K} \sum_{k=1}^{K} g_k(s,r) + h_k(s,r) \tag{25}$$

where,

$$g_k(s,r) = \log(1+\exp(s^T x_k - r)) \tag{26}$$
$$h_k(s,r) = y_k(r - s^T x_k) = \begin{cases} r - s^T x_k, & \text{if } y_k = 1 \\ 0, & \text{if } y_k = 0 \end{cases}$$

For $y_k = 1$, the function $h_k(s,r)$ can be decomposed as it is in the equation 27 bellow:

$$h_k(s,r) = r - s^T x_k = \begin{bmatrix} 1 & -x_k \end{bmatrix} \begin{bmatrix} r \\ s^T \end{bmatrix} \tag{27}$$

From the equation 27 we can conclude that $h_k(s,r)$ is affine and, consequently, it is convex.

For the $g_k(s,r)$ function, replacing "$r - s^T x_k$" by "Θ" we have:

$$g_k(\Theta) = \log(1+e^{-\Theta}) \tag{28}$$

Deriving this $g_k(\Theta)$ function we have:

$$g_k'(\Theta) = \frac{-e^{-\Theta}}{1+e^{-\Theta}} = (-1)\frac{1}{e^{\Theta}+1} \tag{29}$$

Deriving again we have:

$$g_k''(\Theta) = (-1)\frac{-e^{\Theta}}{(e^{\Theta}+1)^2} = \frac{e^{\Theta}}{(e^{\Theta}+1)^2} > 0, x \in \mathbb{R}^n \tag{30}$$

And for the limits of the function $g_k(s,r)$ we have,

$$\lim_{\Theta \to -\infty} g(\Theta) = \log(1+e^{-\Theta}) \simeq \log(e^{-\Theta}) = -\Theta \tag{31}$$

$$\lim_{\Theta \to +\infty} g(\Theta) = \log(1+e^{-\Theta}) \simeq \log(1) = 0 \tag{32}$$

From the previous equations we can conclude that the function $g_k(s,r)$ is convex.

The function $f(s,r)$ is a sum of two convex functions, as we can see in equation 25, so we can conclude that the function $f(s,r)$ is also convex, as we wanted to demonstrate.

2.2 Task 2

From this section we were asked to solve problem 22 for different data, using the Gradient Method.

For this method we need to compute the gradient for the function $f(s,r)$ that is show bellow in equation 33:

$$f(s,r) = \frac{1}{K}\sum_{k=1}^{K} \log(1+\exp(s^T x_k - r)) - y_k(s^T x_k - r) \tag{33}$$

We started by deriving partially the function $f(s,r)$ as we can see in the equations 34 and 35:

$$\frac{\partial}{ds}f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\frac{x_k \exp(s^T x_k - r)}{1+\exp(s^T x_k - r)} - y_k x_k \tag{34}$$

$$\frac{\partial}{dr}f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\frac{(-1)\exp(s^T x_k - r)}{1+\exp(s^T x_k - r)} + y_k \tag{35}$$

This partial derivatives can be vectorized, in order to compute the gradient in the vectorized form, as we can see in the equation 36 bellow:

$$g = \nabla f(s,r) = \frac{1}{K}\sum_{k=1}^{K}\frac{\begin{bmatrix} x_k^T & -1\end{bmatrix}\exp(\begin{bmatrix} x_k^T & -1\end{bmatrix}\begin{bmatrix} s \\ r\end{bmatrix})}{1+\exp(\begin{bmatrix} x_k^T & -1\end{bmatrix}\begin{bmatrix} s \\ r\end{bmatrix})} - y_k\begin{bmatrix} x_k^T & -1\end{bmatrix} \tag{36}$$

Now that we have the gradient of the function $f(s,r)$, we can solve the problem 22 using the Gradient Method. To find the minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point.

The code used for this task is listed bellow:

```
function f = logistic_regression(s_,X,y_)
K=size(X,2);

f = (1/K)*(sum(log(1+exp(s_*X)))-y_*(s_*X)');

end
```

```
s=-ones(1, size(X,1));
r=0;
P=[s r];
actual_x = [X;-ones(1,size(X,2))];
g=ones(size(actual_x,1));

while norm(g) > epsilon
    g=(1/K)*(sum(actual_x.*(exp(P*actual_x)./(1+exp(P*actual_x))),2)-
    - sum(actual_x.*Y,2));
    G=[G g];
    d=-g;
    alpha=alpha_hat;
    while logistic_regression(P+alpha*d',actual_x,Y)>
    >logistic_regression(P,actual_x,Y)+gama*g'*(alpha*d)
        alpha=beta*alpha;
```

```
16        end
17        P=P+alpha*d';
18        k=k+1;
19
20  end
```

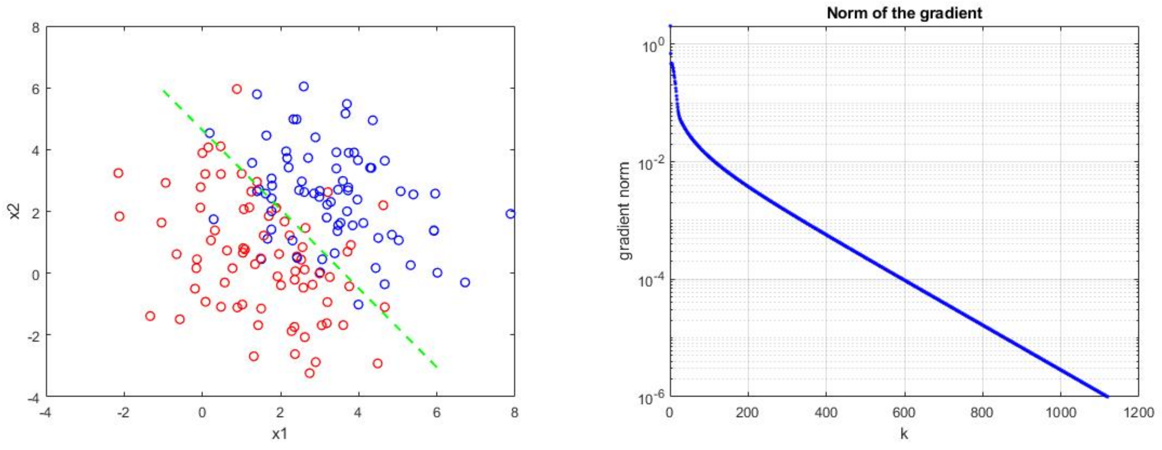
In this task, we wanted to save time in solving the problem using the Gradient Method. So instead of using the cycle *for* to do the sum of the function that we wanted to optimized, we used a vectorized form to compute the gradient of the function $f(s, r)$, like it is show in the equation 36. We computed the gradient this way because the cycle *for* takes a long time to solve the problem with big data and, using the function *sum* with vectors, it's faster. We can see the computation of the gradient in the line 8 of the code.

For every iteration we saved the gradient in a matrix, which can be seen in line 10 of the code, so we could plot the norm of the gradient along the iterations. We can also see that the variable *actual x* represents the concatenation $\begin{bmatrix} x_k^T & -1 \end{bmatrix}$ and the variable P represents the concatenation $\begin{bmatrix} s & r \end{bmatrix}^T$.

This algorithm starts with a loop where the stopping criteria is the $\|g_k\|$ being smaller than a certain threshold. In that loop we find the step size α_k with *backtracking routine* (line 13 to 16 of the code). Finally, we update the variables that we want to find (line 17 of the code) and check again the stopping criteria. And we do this until we find $\|g_k\| <$ threshold that means that the algorithm is close enough to the minimum, depending on the threshold that we choose.

Solution for data1.mat

For this specific task, we were asked to solve the problem 22 for the data1.mat. The plot that shows the data and the line that separates the data it is shown in the figure 42.



(a) (b)

Figure 42: (a) Data1.mat with the separation line; (b) Norm of the gradient

The solution that the Gradient Method gives us for this data1.mat is:

$$s^T = (1.3495, 1.0540); r = 4,8815. \tag{37}$$

2.3 Task 3

For this task, we were asked to solve the problem 22 for the data2.mat. The plot that shows the data and the line that separates the data it is shown in the figure 43.

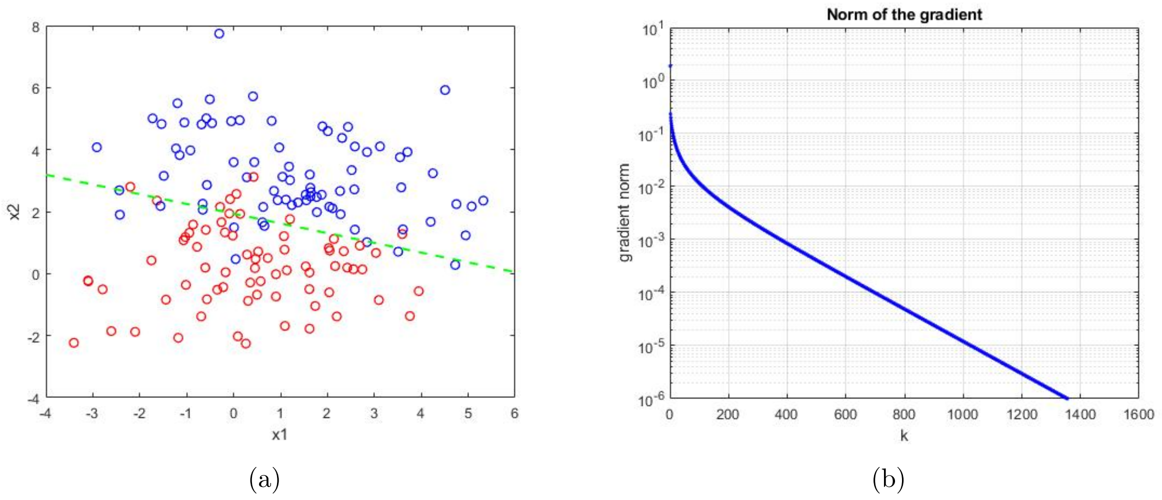


Figure 43: (a) Data2.mat with the separation line; (b) Norm of the gradient

The solution that the Gradient Method gives us for this data2.mat is:

$$s^T = (0.7402, 2.3577); r = 4,5553. \tag{38}$$

2.4 Task 4

2.4.1 Dataset 3

In this dataset4, were needed 3437 iterations for the Gradient Method. The norm of the gradient along the iterations can be seen in the figure 44.

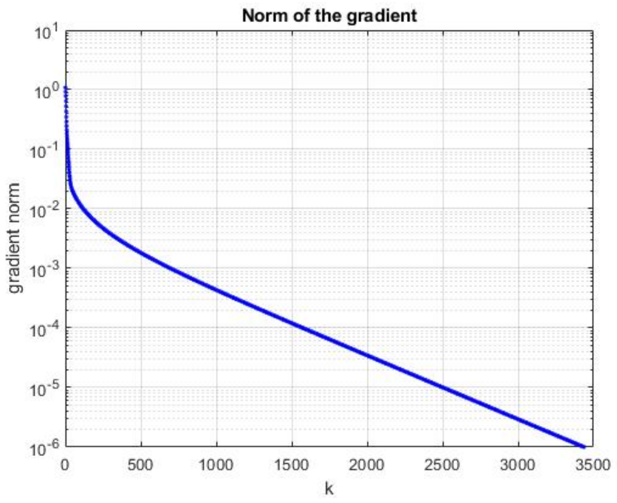


Figure 44: Norm of the gradient along the iterations for data3.mat

The solution that the Gradient Method gives us for this data3.mat is shown in the table 10.

$s(1)$ to $s(10)$	$s(11)$ to $s(20)$	$s(21)$ to $s(30)$	r
-1,308	1,924	0,766	4,798
1,408	-0,359	-0,273	
0,805	-0,290	-0,535	
-1,002	0,192	0,999	
0,555	1,061	-0,419	
-0,549	0,210	-0,313	
-1,199	-0,093	0,408	
0,079	1,047	-0,196	
-1,828	-1,125	-0,738	
-0,148	-1,331	-0,981	

Table 10: Solution for the data3.mat

2.4.2 Dataset 4

In this dataset4, were needed 19893 iterations for the Gradient Method to solve the problem, which is a lot of iterations that take a lot of time to run, but this was expected since the dataset4 has a very large dimension. The norm of the gradient along the iterations can be seen in the figure 45.

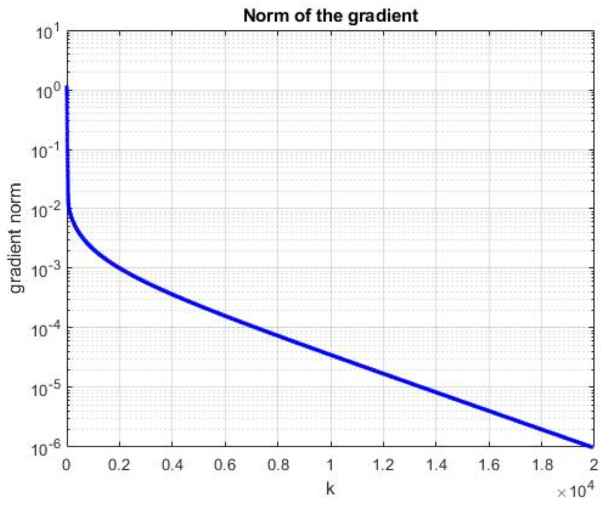


Figure 45: Norm of the gradient along the iterations.

The solution that the Gradient Method gives us for this data3.mat is shown in the table 11 and 12.

$s(1-10)$	$s(11-20)$	$s(21-30)$	$s(31-40)$	$s(41-50)$
0,110	-0,586	0,646	-1,271	-1,053
-0,642	0,627	-0,478	-2,033	0,640
0,102	1,361	1,640	-0,250	0,376
1,243	0,153	0,903	-0,352	-0,155
-1,643	2,323	-1,229	-0,349	0,030
1,024	-0,084	-0,759	-2,561	0,955
0,051	-0,949	-0,489	-0,313	-0,286
0,827	2,470	1,030	-0,490	0,636
0,313	-0,868	0,089	0,726	0,786
0,745	-1,652	-1,092	0,577	0,758

Table 11: Solution for the dataset4.mat

2.4.3 Comments and Comparisons

We can see in the figures 42 and 43 from task 1 and task 2 that the gradient did a very good job solving the problem, since the data seems to be very well separated.

As expected, for data with high dimensions, like dataset4.mat ,the Gradient Method needs a lot of iterations which takes a lot of time to solve the optimization problem, as we

$s(51-60)$	$s(61-70)$	$s(71-80)$	$s(81-90)$	$s(91-100)$	r
0,288	0,477	-0,324	-0,858	1,191	7,670
0,165	-2,368	-0,321	-2,478	1,832	
0,678	-1,156	-1,090	-0,416	-1,729	
2,055	-2,662	-0,829	0,166	0,233	
1,099	0,062	-0,310	0,793	-1,192	
0,526	0,104	-0,488	0,368	1,656	
-0,577	-0,624	-0,106	-0,052	0,461	
1,145	0,191	-0,164	-0,999	-0,643	
-0,561	0,667	2,268	-0,573	0,830	
0,006	-1,049	-1,238	0,397	0,297	

Table 12: Solution for the dataset4.mat

can see in the figure 45. We can also see that the norm of the gradient decreases very fast in the beginning and stays linear until a the minimum is found, like it was expected, because in the beginning the norm of the gradient is big due to being far from the minimum.

2.5 Task 5 - Newton Method

In this section of the project we aim to solve the Logistic Regression problem using only the Newton Method approach. The first task of this section has the objective of helping to find the compact matrix expression of both the gradient and the hessian , trough computing both the gradient and hessian of the function 39 , shown bellow .

$$p(x) = \sum_{k=1}^{K} \phi(a_K^T x) \tag{39}$$

where $a_k \in \mathbf{R}^3$ for $k = 1, \ldots, K$

(a) **Gradient of p demonstration**

We started by writing the total sum into partial sums, which can be seen in equation 41:

$$x = [x_1, x_2, x_3]' \quad a_1 = [a_{11}, a_{12}, a_{13}]' \quad a_2 = [a_{21}, a_{22}, a_{23}]' \quad a_3 = [a_{31}, a_{32}, a_{33}]' \tag{40}$$

So, $p(x)$ is

$$p(x) = \phi(a_1^T x) + \phi(a_2^T x) + \phi(a_3^T x) \tag{41}$$

The gradient of $p(x)$ is given by,

$$\nabla p(x) = \begin{bmatrix} \frac{\partial p(x)}{\partial x_1} \\ \frac{\partial p(x)}{\partial x_2} \\ \frac{\partial p(x)}{\partial x_3} \end{bmatrix} \tag{42}$$

The partial derivatives are:

$$\frac{\partial p(x)}{\partial x_1} = a_{11}\dot{\phi}(a_1^T x) + a_{21}\dot{\phi}(a_2^T) + a_{31}\dot{\phi}(a_3^T x)$$

$$\frac{\partial p(x)}{\partial x_a} = a_{12}\dot{\phi}(a_1^T x) + a_{22}\dot{\phi}(a_2^T) + a_{32}\dot{\phi}(a_3^T x)$$

$$\frac{\partial p(x)}{\partial x_3} = a_{13}\dot{\phi}(a_1^T x) + a_{23}\dot{\phi}(a_2^T) + a_{33}\dot{\phi}(a_3^T x)$$

$$\nabla p(x) = \begin{bmatrix} a_{11}\dot{\phi}(a_1^T x) + a_{21}\dot{\phi}(a_2^T) + a_{31}\dot{\phi}(a_3^T x) \\ a_{12}\dot{\phi}(a_1^T x) + a_{22}\dot{\phi}(a_2^T) + a_{32}\dot{\phi}(a_3^T x) \\ a_{13}\dot{\phi}(a_1^T x) + a_{23}\dot{\phi}(a_2^T) + a_{33}\dot{\phi}(a_3^T x) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \begin{bmatrix} \dot{\phi}(a_1^T x) \\ \dot{\phi}(a_2^T x) \\ \dot{\phi}(a_3^T x) \end{bmatrix} \tag{43}$$

The gradient of $p(x)$ is given by the following expression,

$$\nabla p(x) = [a_1, a_2, a_3] \begin{bmatrix} \dot{\phi}(a_1^T x) \\ \dot{\phi}(a_2^T x) \\ \dot{\phi}(a_3^T x) \end{bmatrix} \tag{44}$$

Then we compute the derivatives and vectorize them as shown in equation **??** bellow and obtain the matrices A and v which are denoted in equation 45:

$$\nabla p(x) = [a_1, a_2, ...a_k] \begin{bmatrix} \dot{\phi}(a_1^T x) \\ \dot{\phi}(a_2^T x) \\ ... \\ \dot{\phi}(a_k^T x) \end{bmatrix} = Av \tag{45}$$

As we wanted to conclude.

(b) **Hessian of p demonstration**

The Hessian matrix of any function $p(x)$ is given by the equation 46,

$$\nabla^2 p(x) = \begin{bmatrix} \frac{\partial p(x)}{\partial x_1 x_1} & \frac{\partial p(x)}{\partial x_1 x_2} & \frac{\partial p(x)}{\partial x_1 x_3} \\ \frac{\partial p(x)}{\partial x_2 x_1} & \frac{\partial p(x)}{\partial x_2 x_2} & \frac{\partial p(x)}{\partial x_2 x_3} \\ \frac{\partial p(x)}{\partial x_3 x_1} & \frac{\partial p(x)}{\partial x_3 x_2} & \frac{\partial p(x)}{\partial x_3 x_3} \end{bmatrix} \tag{46}$$

Only 6 partial derivatives need to be compute since,

$$\frac{\partial p(x)}{\partial x_1 x_2} = \frac{\partial p(x)}{\partial x_2 x_1} \quad \frac{\partial p(x)}{\partial x_1 x_3} = \frac{\partial p(x)}{\partial x_3 x_1} \quad \frac{\partial p(x)}{\partial x_2 x_3} = \frac{\partial p(x)}{\partial x_3 x_2}$$

$$\begin{aligned}
\frac{\partial p(x)}{\partial x_1 x_1} &= a_{11}a_{11}\ddot{\phi}(a_1^T x) + a_{21}a_{21}\ddot{\phi}(a_2^T x) + a_{31}a_{31}\ddot{\phi}(a_3^T x) \\
\frac{\partial p(x)}{\partial x_2 x_2} &= a_{12}a_{12}\ddot{\phi}(a_1^T x) + a_{22}a_{22}\ddot{\phi}(a_2^T x) + a_{32}a_{32}\ddot{\phi}(a_3^T x) \\
\frac{\partial p(x)}{\partial x_3 x_3} &= a_{13}a_{13}\ddot{\phi}(a_1^T x) + a_{23}a_{23}\ddot{\phi}(a_2^T x) + a_{33}a_{33}\ddot{\phi}(a_3^T x) \\
\frac{\partial p(x)}{\partial x_1 x_2} &= a_{11}a_{12}\ddot{\phi}(a_1^T x) + a_{21}a_{22}\ddot{\phi}(a_2^T x) + a_{31}a_{32}\ddot{\phi}(a_3^T x) \\
\frac{\partial p(x)}{\partial x_1 x_3} &= a_{11}a_{13}\ddot{\phi}(a_1^T x) + a_{21}a_{23}\ddot{\phi}(a_2^T x) + a_{31}a_{33}\ddot{\phi}(a_3^T x) \\
\frac{\partial p(x)}{\partial x_2 x_3} &= a_{12}a_{13}\ddot{\phi}(a_1^T x) + a_{22}a_{23}\ddot{\phi}(a_2^T x) + a_{32}a_{33}\ddot{\phi}(a_3^T x)
\end{aligned}$$

The Hessian matrix of $p(x)$ is then written has,

$$\nabla^2 p(x) = \ddot{\phi}(a_1^T) \begin{bmatrix} a_{11}a_{11} & a_{11}a_{12} & a_{11}a_{13} \\ a_{12}a_{11} & a_{12}a_{12} & a_{12}a_{13} \\ a_{13}a_{11} & a_{13}a_{12} & a_{13}a_{13} \end{bmatrix} + \ddot{\phi}(a_2^T) \begin{bmatrix} a_{21}a_{21} & a_{21}a_{22} & a_{21}a_{23} \\ a_{22}a_{21} & a_{22}a_{22} & a_{22}a_{23} \\ a_{23}a_{21} & a_{23}a_{22} & a_{23}a_{23} \end{bmatrix} + \tag{47}$$

$$\ddot{\phi}(a_3^T) \begin{bmatrix} a_{31}a_{31} & a_{31}a_{32} & a_{31}a_{33} \\ a_{32}a_{31} & a_{32}a_{32} & a_{32}a_{33} \\ a_{33}a_{31} & a_{33}a_{32} & a_{33}a_{33} \end{bmatrix}$$

That can be written as:

$$\nabla^2 p(x) = \ddot{\phi}(a_1^T x) \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix} [a_{11}, a_{12}, a_{13}] + \ddot{\phi}(a_2^T x) \begin{bmatrix} a_{21} \\ a_{22} \\ a_{23} \end{bmatrix} [a_{21}, a_{22}, a_{23}] + \tag{48}$$

$$\ddot{\phi}(a_3^T x) \begin{bmatrix} a_{31} \\ a_{32} \\ a_{33} \end{bmatrix} [a_{31}, a_{32}, a_{33}]$$

Finally, it is possible to conclude that the Hessian matrix for this example is given by,

$$\nabla^2 = [a_1, a_2, a_3] \begin{bmatrix} \ddot{\phi}(a_1^T x) & 0 & 0 \\ 0 & \ddot{\phi}(a_2^T x) & 0 \\ 0 & 0 & \ddot{\phi}(a_3^T x) \end{bmatrix} \begin{bmatrix} a_1^T \\ a_2^T \\ a_3^T \end{bmatrix} \tag{49}$$

Now to finish the computation of the hessian we just need to vectorize the partial sums into matrices , which yields the equality shown in equation b :

$$\nabla^2 = [a_1, ..., a_k] \begin{bmatrix} \ddot{\phi}(a_1^T x) & 0 & 0 \\ 0 & ... & 0 \\ 0 & 0 & \ddot{\phi}(a_k^T x) \end{bmatrix} \begin{bmatrix} a_1^T \\ ... \\ a_k^T \end{bmatrix} = ADA^T \tag{50}$$

As we wanted to conclude.

2.6 Task 6

In this section we are asked to solve problem 22 for the data sets data1.mat, data2.mat, data3.mat, and data4.mat using the Newton Method. Since we already computed the gradient of $f(s, r)$ which can be seen on equation 33 we only need to compute the hessian of this function to do that we apply the formulas demonstrated in task 5 , so first we find matrix A and then matrix v:

$$A = \begin{bmatrix} x_1 & x_2 & ... & x_K \end{bmatrix} \tag{51}$$

$$D = \begin{bmatrix} \frac{1}{K} exp(\begin{bmatrix} x_1^T & -1 \end{bmatrix} * \begin{bmatrix} s \\ r \end{bmatrix}) & \dots & 0 & \\ 0 & \frac{1}{K} exp(\begin{bmatrix} x_2^T & -1 \end{bmatrix} * \begin{bmatrix} s \\ r \end{bmatrix}) & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \dots & \frac{1}{K} exp(\begin{bmatrix} x_K^T & -1 \end{bmatrix} * \begin{bmatrix} s \\ r \end{bmatrix}) \end{bmatrix} \tag{52}$$

After computing these two matrices we just need to apply the formula stated in task 5 (b) and we get the hessian :

$$\nabla^2 f(s, r) = A * D * A \tag{53}$$

Now that we have the hessian of the function f(s,r) , we can solve problem 17 using the Newton Method. To find the minimum of a function using the Newton Algorithm one takes steps proportional to the negative of the hessian (or approximate hessian) of the function at the current point. This algorithm starts with a loop where the stopping criteria is the $||g_k||$ being smaller than a certain threshold. In that loop we nd the step size α_k with a backtracking routine. Finally, we update the variables that we want to nd and check again the stopping criteria.

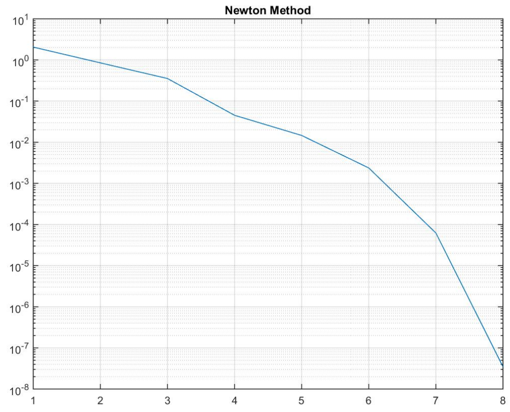
And we do this until we $||g_k|| <$ threshold that means that the algorithm is close enough to the minimum, depending on the threshold that we choose.

To solve this task we have to solve problem 22 for the data1.mat , data2.mat , data3.mat and data4.mat . Bellow are shown the code and results for each data set .

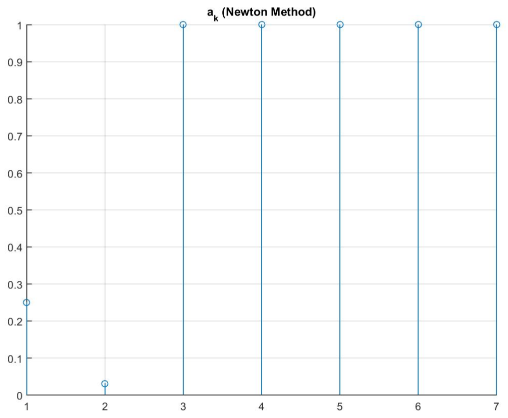
```matlab
function [ back ] = f1( x , X , Y )

K = length(Y);
A = [X; -ones(1, K)];

back = 1/K * sum(log(1+exp(A'*x)) - Y'.*(A'*x));

end
```

```matlab
clc
clear all

load('data1.mat'); %to test with different data change the filename here.
Miterations = 15;

s = -ones(1,size(X,1))';
r =0;

epsilon=10^-6;
a=1;
gamma=10^-4;
beta=0.5;

P=[s; r];

norm_vec = zeros(1,Miterations);
ak_vec = zeros(1,Miterations);

K = length(Y);
A = [X; -ones(1, K)];


for k = 1:Miterations

    v = exp(A'*P)./(1 + exp(A'*P)) - Y';
    g = 1/K * (A*v);

    norm_vec(k) = norm(g);
    if ( norm(g) ≤ epsilon )
        break;
    end

    D = 1/K * diag(exp(A'*P)./((1+exp(A'*P)).^2));
    hessian = A*D*A';
    d = -inv(hessian)*g;


    ak = a;
    while(1)
        if( f1(P + ak*d , X , Y) < f1(P, X, Y) + gamma * g * (ak*d)' )
            break;
        end
        ak = beta * ak;
    end


    ak_vec(k) = ak;
    P = P + ak*d;

end
```

2.6.1 Newton Method Dataset 1 results



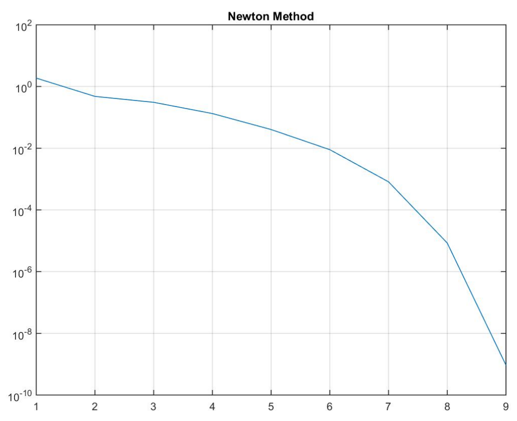
(a)



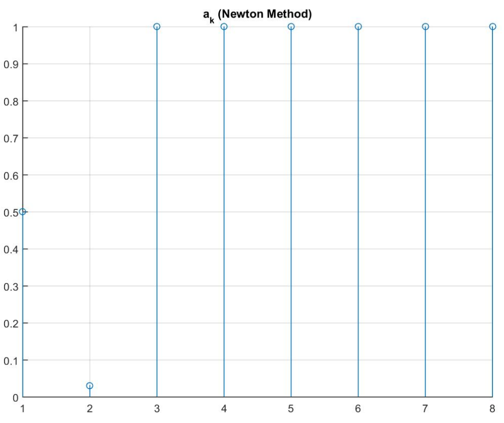
(b)

Figure 46: (a) Gradient norm along iterations for data1.mat; (b) Step sizes along iterations for data1.mat

2.6.2 Newton Method Dataset 2 results



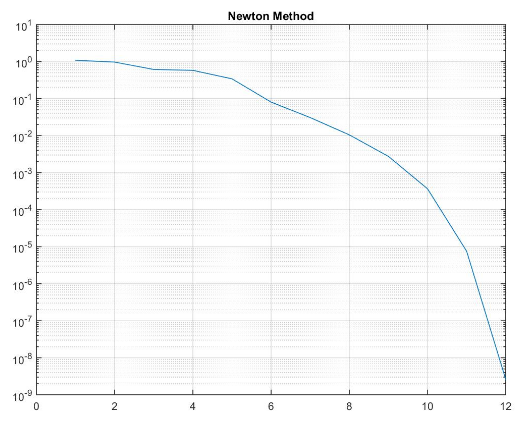
(a)



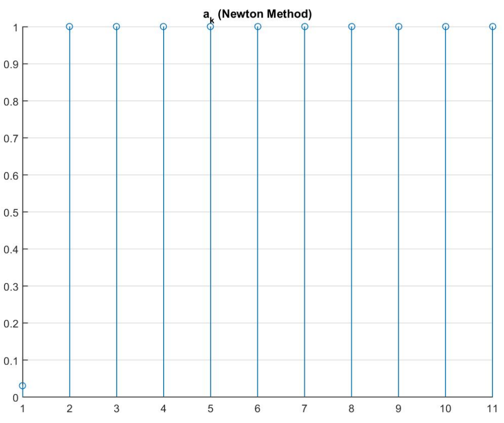
(b)

Figure 47: (a) Gradient norm along iterations for data2.mat; (b) Step sizes along iterations for data2.mat

2.6.3 Newton Method Dataset 3 results

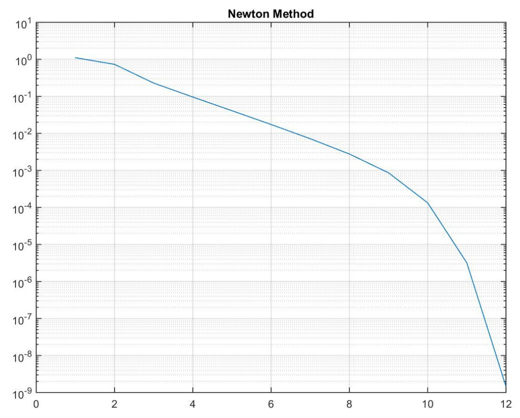


(a)

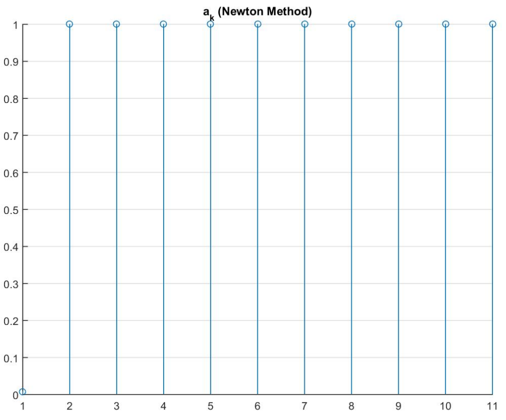


(b)

Figure 48: (a) Gradient norm along iterations for data3.mat; (b) Step sizes along iterations for data3.mat



(a)



(b)

Figure 49: (a) Gradient norm along iterations for dataset4.mat; (b) Step sizes along iterations for dataset4.mat

2.6.4 Newton Method Dataset 4 results

2.7 Task 7

Newton's Method minimizes a function by using it's knowledge of its first and second derivatives . When the second derivative is known and easy to compute as it is in the case of problem 22 this method tends to be very fast and in comparison with the gradient descent method its always much faster and requires far less iterations. For data1.mat and data2.mat the Newton Method converges in less than 10 iterations and takes less than 0.008 seconds , while the gradient descent method takes more than 1000 iterations and takes in average 0.01 seconds. For the case of this first two datasets the amount of data in question is not significantly high so both the methods take an insignificant amount of time , but the Newton Method takes far less iterations . For the last two data sets (data3.mat and data4.mat) the amount of data is bigger , so its easy to tell the difference between both methods , as for the Newton Method for data3.mat the Method converges in 11 iterations and only takes 0.03 seconds , while the gradient descent method takes 3437 iterations and 1,20 seconds which is two degree orders above of the Newton Method results. For data4.mat the difference is remarkable , the Newton Method converges in 11 iterations as in the previous data and takes approximately 5 seconds and the gradient descent takes a whopping 19893 iterations and 587 seconds . By analyzing the results gathered from this experience alone , one would say that the Newton Method is better than the Gradient descent Method since it's always faster and takes far less iterations , but that is not always the case , in this situation the second derivative is easy to compute and the function is twice differentiable so its expected that the Newton Method shows a far better performance because by taking in regard the information of the second derivative the method converges faster because it not only knows what is the steepest descent from the current point but also what is the right direction to move towards in order to minimize the function . However the Gradient Descent Method as its pros in comparison to the Newton's Method (although they are not quite shown here) , because it only depends of the first derivative the method is much more robust and it is applicable to a much broader class of functions. Also when the hessian is hard to compute , the Newton Method iterations may take a long time to execute in comparison with the gradient method iteration because if we need $O(n)$ operations to compute the first derivative , to compute second derivative we need O(n^2) operations. In sum both Methods have their good and bad points , their use is dependent upon the conditions of the problem in question , but for the problem 22 it is safe to say that Newton's Method is a better approach especially if the amount of data is large.

2.8 Task 8 - Network localization

In this section of the project, the objective is to find the positions of several sensors, given noisy measurements of the distance between some of them and some anchors (sensors with known positions) and between some of the sensors, using the **Levenberg-Marquardt method**. Representing the unknown positions $s_1, \dots, s_P$ of the sensors in the variable $x = \begin{bmatrix} s_1 & s_2 & \dots & s_P \end{bmatrix}^T$. The sensors' positions can be found using the following optimization problem:

$$\underset{x}{\text{minimize}} \quad \underbrace{\sum_{(m,p)\in\mathcal{A}} (\|a_m - s_p\| - y_{mp})^2 + \sum_{(p,q)\in\mathcal{S}} (\|s_p - s_q\| - z_{pq})^2}_{f(x)}, \tag{54}$$

where $\mathcal{A}$ is the set of pairs (anchor, sensor) for which we have noisy measurements of their distance, and $\mathcal{S}$ is the set of pairs (sensor, sensor) for which we have noisy measurements of their distance.

Calculation of f(x)

To compute the function that minimizes the optimization problem, $f(x)$ was divided in two functions $f_1(x)$ and $f_2(x)$, which were broke down in parallel. In order to calculate each in a vector form, two vectors h_1 and h_2 were created. As a way of making the calculations easier, $\mathcal{A}$ and $\mathcal{S}$ are considered all the combinations of m and p, and of p and q, respectively. All the intermediary variables can be observed in the equations form 55 to 61.

$$f_1(x) = \sum_{m=1}^{M}\sum_{p=1}^{P}(||a_m - s_p|| - y_{mp})^2 \qquad f_2(x) = \sum_{p=1}^{P}\sum_{q=1}^{P}(||s_p - s_q|| - z_{p_q})^2 \tag{55}$$

$$n_{mp} = a_m - s_p \qquad n_{pq} = s_p - s_q \tag{56}$$

$$h_{mp} = ||n_{mp}|| - y_{mp} \qquad h_{pq} = ||n_{pq}|| - z_{p_q} \tag{57}$$

$$f_1(x) = \sum_{m=1}^{M}\sum_{p=1}^{P} h_{mp}^2 \qquad f_2(x) = \sum_{p=1}^{P}\sum_{q=1}^{P} h_{pq}^2 \tag{58}$$

$$h_1 = [h_{11} \dots h_{mp} \dots h_{MP}] \qquad h_2 = [h_{11} \dots h_{pq} \dots h_{PP}] \tag{59}$$

$$f_1(x) = ||h_1||^2 \qquad f_2(x) = ||h_2||^2 \tag{60}$$

$$f(x) = f_1(x) + f_2(x) = ||h_1||^2 + ||h_2||^2 \tag{61}$$

The code used to implement $f(x)$, located in the function `getFunctions`, is presented bellow:

```matlab
1  %get norm vectors
2  [norm1, n1]=normOfDiff(A,x);
3  [norm2, n2]=normOfDiff(x,x);
4
5  %select values correspondent to the set (prune values)
6  n1=n1(:,(iA(:,1)-1)*length(x)+iA(:,2));
7  n2=n2(:,(iS(:,1)-1)*length(x)+iS(:,2));
8  norm1=norm1((iA(:,1)-1)*length(x)+iA(:,2));
9  norm2=norm2((iS(:,1)-1)*length(x)+iS(:,2));
10
11 h1=norm1-y;
12 h2=norm2-z;
13
14 f1=norm(h1)^2;
15 f2=norm(h2)^2;
16
17 f=f1+f2;
```

In line 2, the variables *n1* and *norm1*, represent, respectively, $[n_{11}...n_{mp}...n_{MP}]$ and $\Big[||n_{11}||...||n_{mp}||...||n_{MP}||\Big]$. The same applies to line 3, but with respect to $f_2(x)$. As can be seen, from line 6 to 9, only the vectors with a y_{mp} or z_{pq} correspondent (observations) are considered, in order to prune the values for which there aren't observations. To get the *norm* and *n* vectors, an auxiliary function `normOfDiff` (presented bellow) was created.

```matlab
function [ v_norm, diff ] = normOfDiff( X , Y )

%separate the coordinates
[A1,S1]=meshgrid(X(1,:),Y(1,:));
[A2,S2]=meshgrid(X(2,:),Y(2,:));

%get the difference
diff1=A1-S1;
diff2=A2-S2;

%convert to array (column vectors)
diff1=diff1(:);
diff2=diff2(:);

diff=[diff1'; diff2'];

%vertical norm of the vector
v_norm=vecnorm(diff);

% convert to collumn vector
v_norm=v_norm';

end
```

Calculation of the gradient of f(x)

The gradient of $f(x)$ is the same as the sum of the gradients of $f_1(x)$ and $f_2(x)$ (equation 62), so it is possible to calculate each one individually.

$$\nabla f(x) = \nabla f_1(x) + \nabla f_2(x) \tag{62}$$

Gradient of f1(x): The gradient of $f_1(x)$ is equal to the derivative with respect to every s_i, for $i = 1, ...p$. Knowing that each point as two coordinates, the derivative has to be calculated with respect the each one, as showed in 63.

$$\nabla f_1(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial s_1} \\ \vdots \\ \frac{\partial f_1(x)}{\partial s_P} \end{bmatrix} \qquad \frac{\partial f_1(x)}{\partial s_i} = \begin{bmatrix} \frac{\partial f_1(x)}{\partial s_{i_1}} \\ \frac{\partial f_1(x)}{\partial s_{i_2}} \end{bmatrix} \tag{63}$$

Using the change of variables used before (equations 56 to 60), knowing that the sum of the derivatives is the derivative of the sum, we get 64.

$$\frac{\partial f_1(x)}{\partial s_{i_1}} = \frac{\partial}{\partial s_{i_1}} \left(\sum_{m=1}^{M} \sum_{p=1}^{P} h_{mp}^2 \right) = \sum_{m=1}^{M} \sum_{p=1}^{P} \left(\frac{\partial}{\partial s_{i_1}} h_{mp}(x)^2 \right) = 2 \sum_{m=1}^{M} \sum_{p=1}^{P} \left(\frac{\partial h_{mp}}{\partial s_{i_1}} h_{mp} \right) \tag{64}$$

Breaking down the derivative of h_{mp} with respect to the first coordinate of any point s_i, we easily conclude the derivative only exist when $p = i$ (equation 66). Then, proceeding analogy with respect to the second coordinate, we infer 67.

$$\frac{\partial h_{mp}}{\partial s_{i_1}} = \frac{\partial}{\partial s_{i_1}} \Big(||n_{mp}|| - y_{mp} \Big) = \frac{\partial}{\partial s_{i_1}} \left(\Big((a_{m_1} - s_{p_1})^2 + (a_{m_2} - s_{p_2})^2 \Big)^{\frac{1}{2}} \right) \tag{65}$$

$$= \frac{\partial h_{mp}}{\partial s_{p_1}} = \begin{cases} -\frac{a_{m_1} - s_{p_1}}{||n_{mp}||}, & \text{if } p = i \\ 0, & \text{if } p \neq i \end{cases} \tag{66}$$

$$\frac{\partial h_{mp}}{\partial s_i} = \begin{bmatrix} \frac{\partial h_{mp}}{\partial s_{i_1}} \\ \frac{\partial h_{mp}}{\partial s_{i_2}} \end{bmatrix} = \frac{\partial h_{mp}(x)}{\partial s_p} = \begin{cases} -\frac{1}{||n_{mp}||} \begin{bmatrix} a_{m_1} - s_{p_1} \\ a_{m_2} - s_{p_2} \end{bmatrix} = -\frac{n_{mp}}{||n_{mp}||}, & \text{if } p = i \\ 0, & \text{if } p \neq i \end{cases} \tag{67}$$

Subsequently, summing up everything, in (68), it possible to find the equation representing the derivative of $f_1(x)$ with respect the sensor s_i.

$$\frac{\partial f_1(x)}{\partial s_i} = 2\sum_{m=1}^{M}\sum_{p=1}^{P}\left(\frac{\partial h_{mp}}{\partial s_i}h_{mp}\right) = 2\sum_{m=1}^{M}\left(\frac{\partial h_{mi}}{\partial s_i}h_{mi}\right) = -2\sum_{m=1}^{M}\left(\frac{n_{mi}}{||n_{mi}||}h_{mi}\right) \tag{68}$$

Finally, the gradient of $f_1(x)$ is given by:

$$\nabla f_1(x) = \begin{bmatrix} -2\sum_{m=1}^{M}\left(\frac{n_{m1}}{||n_{m1}||}h_{m1}\right) \\ \vdots \\ -2\sum_{m=1}^{M}\left(\frac{n_{mP}}{||n_{mP}||}h_{mP}\right) \end{bmatrix} = -2\sum_{m=1}^{M}\begin{bmatrix} \frac{n_{m1}}{||n_{m1}||}h_{m1} \\ \vdots \\ \frac{n_{mP}}{||n_{mP}||}h_{mP} \end{bmatrix} \tag{69}$$

Gradient of f2(x): Similarly to $\nabla f_1(x)$, the derivative of h_pq is computed in the equations 70 to 72. However, this time, as the there are 2 terms with respect to the sensors coordinates, three types of derivatives are possible:

- 0 - if the any of the sensor coordinates is equal to s_i or if $s_p = s_q$;
- positive value - if $s_p = s_i$;
- negative value - if $s_q = s_i$.

$$\frac{\partial h_{pq}}{\partial s_{i_1}} = \frac{\partial}{\partial s_{i_1}}\left(||n_{pq}|| - z_{pq}\right) = \frac{\partial}{\partial s_{i_1}}\left(\left((s_{p_1} - s_{q_1})^2 + (s_{p_2} - s_{q_2})^2\right)^{\frac{1}{2}}\right) \tag{70}$$

$$= \frac{\partial h_{pq}}{\partial s_{p_1}} = \begin{cases} \frac{s_{p_1} - s_{q_1}}{||n_{pq}||}, & \text{if } p = i \land q \neq i \\ -\frac{s_{p_1} - s_{q_1}}{||n_{pq}||}, & \text{if } p \neq i \land q = i \\ 0, & \text{otherwise} \end{cases} \tag{71}$$

$$\frac{\partial h_{pq}}{\partial s_i} = \begin{bmatrix} \frac{\partial h_{pq}}{\partial s_{i_1}} \\ \frac{\partial h_{pq}}{\partial s_{i_2}} \end{bmatrix} = \begin{cases} \frac{n_{pq}}{||n_{pq}||}, & \text{if } p = i \land q \neq i \\ -\frac{n_{pq}}{||n_{pq}||}, & p \neq i \land q = i \\ 0, & \text{otherwise} \end{cases} \tag{72}$$

Looking to the previous equation, we observe that the derivatives are equal with just a difference in sign. So, to compute the derivative of $f_2(x)$ with respect to s_i, it is only necessary to sum the terms were $p = i$ and subtract the ones were $q = i$, as can be seen in 73.

$$\frac{\partial f_2(x)}{\partial s_i} = 2\sum_{p=1}^{P}\sum_{q=1}^{P}\left(\frac{\partial h_{pq}}{\partial s_i}h_{pq}\right) = 2\sum_{q=1}^{P}\left(\frac{n_{iq}}{||n_{iq}||}h_{iq}\right) - 2\sum_{p=1}^{P}\left(\frac{n_{pi}}{||n_{pi}||}h_{pi}\right) \tag{73}$$

Then, the gradient of $f_2(x)$ is given by 74.

$$\nabla f_2(x) = 2\sum_{q=1}^{P}\begin{bmatrix}\frac{n_{1q}}{||n_{1q}||}h_{1q}\\ \vdots \\ \frac{n_{Pq}}{||n_{Pq}||}h_{Pq}\end{bmatrix} - 2\sum_{p=1}^{P}\begin{bmatrix}\frac{n_{p1}}{||n_{p1}||}h_{p1}\\ \vdots \\ \frac{n_{pP}}{||n_{pP}||}h_{pP}\end{bmatrix} \tag{74}$$

At last, the gradient of $f(x)$ can be obtain in 76.

$$\nabla f(x) = \nabla f_1(x) + \nabla f_2(x) \tag{75}$$

$$= -2\sum_{m=1}^{M}\begin{bmatrix}\frac{n_{m1}}{||n_{m1}||}h_{m1}\\ \vdots \\ \frac{n_{mP}}{||n_{mP}||}h_{mP}\end{bmatrix} + 2\sum_{q=1}^{P}\begin{bmatrix}\frac{n_{1q}}{||n_{1q}||}h_{1q}\\ \vdots \\ \frac{n_{Pq}}{||n_{Pq}||}h_{Pq}\end{bmatrix} - 2\sum_{p=1}^{P}\begin{bmatrix}\frac{n_{p1}}{||n_{p1}||}h_{p1}\\ \vdots \\ \frac{n_{pP}}{||n_{pP}||}h_{pP}\end{bmatrix} \tag{76}$$

In order to make it clearer, we can make $dh_{1_{mp}} = \frac{n_{mp}}{||n_{mp}||}$ and $dh_{2_{pq}} = \frac{n_{pq}}{||n_{pq}||}$. Then, $df_{1_{mp}} = dh_{1_{mp}}h_{mp}$ and $df_{2_{pq}} = dh_{2_{pq}}h_{pq}$, obtain the final equation 78.

$$\nabla f(x) = -2\sum_{m=1}^{M}\begin{bmatrix}dh_{1_{m1}}h_{m1}\\ \vdots \\ dh_{1_{mP}}h_{mP}\end{bmatrix} + 2\sum_{q=1}^{P}\begin{bmatrix}dh_{2_{1q}}h_{1q}\\ \vdots \\ dh_{2_{Pq}}h_{Pq}\end{bmatrix} - 2\sum_{p=1}^{P}\begin{bmatrix}dh_{2_{p1}}h_{p1}\\ \vdots \\ dh_{2_{pP}}h_{pP}\end{bmatrix} \tag{77}$$

$$= -2\sum_{m=1}^{M}\begin{bmatrix}df_{1_{m1}}\\ \vdots \\ df_{1_{mP}}\end{bmatrix} + 2\sum_{q=1}^{P}\begin{bmatrix}df_{2_{1q}}\\ \vdots \\ df_{2_{Pq}}\end{bmatrix} - 2\sum_{p=1}^{P}\begin{bmatrix}df_{2_{p1}}\\ \vdots \\ df_{2_{pP}}\end{bmatrix} \tag{78}$$

Important observation: Once there is only a limited number of observations (it is not given all the combinations of mp and of pq), we compute $f(x)$ and $\nabla f(x)$, and all the intermediary calculations for just those observations.

The code that computes the gradient of $f(x)$ is showed bellow:

```
%find gradient of f1
dh1=n1'./[norm1,norm1];
df1=-2*dh1.*[h1,h1];
grad_f1=pA*df1;

```

```
%find the gradient of f2
dh2=n2'./[norm2,norm2];
df2=2*dh2.*[h2,h2];
grad_f2=pS*df2-qS*df2;

grad_f=grad_f1+grad_f2;
```

The matrices pA, pS and qS are selecting matrix, specially designed to generate the gradients. To understand how this matrices work, let's look at the example bellow, where there are 2 anchors and 4 sensors. The goal is to extract the values from df_1 in order to get the gradient with respect to s_p.

$$df_1 = \begin{bmatrix} df_{1_{12}} \\ df_{1_{14}} \\ df_{1_{23}} \\ df_{1_{24}} \\ df_{1_{21}} \\ df_{1_{22}} \end{bmatrix} \rightarrow \nabla f_1 = \begin{bmatrix} df_{1_{21}} \\ df_{1_{12}} + df_{1_{22}} \\ df_{1_{23}} \\ df_{1_{14}} + df_{1_{24}} \end{bmatrix}$$

A selecting matrix (pA) is necessary to make this transformation. Then, if

$$pA = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \implies pA(df_1) = \nabla f_1$$

The procedure for matrix pS (with respect to s_p) and qS (with respect to s_q) is similar, but with df_2 instead of df_1.

The code used to generate these matrices is shown in the function `getSelectionMatrix` bellow. This function is only run one time, because the matrices are static (don't change with x). The data produced is saved as `processed_data.m`, along with the data from the `lmdata.m` file.

```
function [] = getSelectionMatrices(filename)
load(filename);

N1=length(iA); % number of observations of y
N2=length(iS); % number of observations of y
M=length(A); %number os anchors
P=8; %number of sensors

pA=zeros(P,N1);
pS=zeros(P,N2);
qS=zeros(P,N2);

```

```
for i=1:P
    pA(i,find(iA(:,2)'==i))=1;
    pS(i,find(iS(:,1)'==i))=1;
    qS(i,find(iS(:,2)'==i))=1;
end

save 'processed_data.mat'

end
```

Levenberg-Marquardt (LM) algorithm

To compute the LM algorithm, $f(x)$ has to be seen in the following form:

$$f(x) = \sum_{n=1}^{N} f_n(x)^2 \tag{79}$$

In order to generate each iteration of x, using this method, it is necessary to solve the problem presented in 80, which is a standard least-squares.

$$\underset{x}{\text{minimize}} \quad \left(\sum_{n=1}^{N} (f_n(x_k) + \nabla f_n(x_k)^T (x - x_k))^2 + \lambda_k ||x - x_k||^2 \quad = ||Ax - b||^2 \right) \tag{80}$$

If we look back at the equation 58, we can observe that $f_1(x)$ and $f_2(x)$ are already in the wanted form in 79. So, in our current problem, $f(x)$ can be seen as:

$$f(x) = \sum_{(m,p)\in\mathcal{A}} h_{mp}^2 + \sum_{(p,q)\in\mathcal{S}} h_{pq}^2, \tag{81}$$

where each f_n is equal to h_{ij}.

Remembering that $h_{mp} \neq h_{pq}$, to avoid confusion, we will refer as $h1_{mp}$ and $h2_{mp}$, respectively.

To solve the least-squares problem, the matrices obtained were:

$$A = \begin{bmatrix} \nabla h1_{11}(x_k)^T \\ \vdots \\ \nabla h1_{mp}(x_k)^T \\ \vdots \\ \nabla h1_{MP}(x_k)^T \\ \nabla h2_{11}(x_k)^T \\ \vdots \\ \nabla h2_{pq}(x_k)^T \\ \vdots \\ \nabla h2_{PP}(x_k)^T \\ \sqrt{\lambda_k} I \end{bmatrix} \qquad b = \begin{bmatrix} \nabla h1_{11}(x_k)^T x_k - h1_{11}(x_k) \\ \vdots \\ \nabla h1_{mp}(x_k)^T x_k - h1_{mp}(x_k) \\ \vdots \\ \nabla h1_{MP}(x_k)^T x_k - h1_{MP}(x_k) \\ \nabla h2_{11}(x_k)^T x_k - h2_{11}(x_k) \\ \vdots \\ \nabla h2_{pq}(x_k)^T x_k - h2_{pq}(x_k) \\ \vdots \\ \nabla h2_{PP}(x_k)^T x_k - h2_{PP}(x_k) \\ \lambda_k x_k \end{bmatrix} \tag{82}$$

$$\tag{83}$$

From previous calculations, we know that the gradient of $h1_{mp}$ is always 0, except when $s_p = s_i$; and the gradient of $h2_{pq}$, the non-zero values occur when $s_p = s_i \vee s_q = s_i$.

$$\nabla h1_{ij}(x) = \begin{bmatrix} 0 & \dots & -dh^T_{1_{mp}}|_{p=j} & \dots & 0 \end{bmatrix} \tag{84}$$

$$\nabla h2_{ij}(x) = \begin{bmatrix} 0 & \dots & dh^T_{2_{pq}}|_{p=i} & \dots & 0 & \dots & -dh^T_{2_{pq}}|_{q=i} & \dots \end{bmatrix} \tag{85}$$

It is important to refer again, that only the $h1_{mp}$ $h2_{pq}$, whose observations are given, are used in the algorithm. The function that solves the least squares problem and generates x_{k+1} is implemented by the following code:

```
function [ x_next ] = getNextX( x, h1, h2, dh1, dh2, lambda)
load 'processed_data.mat'

A1=zeros(length(dh1),length(x)*2);
for i=1:length(dh1)
    for j=1:length(x)
        if j==iA(i,2)
            A1(i,j*2-1:j*2)=-dh1(i,:);
        end
    end
end

for i=1:length(dh2)
    for j=1:length(x)
        if j==iS(i,1);
            A2(i,j*2-1:j*2)=dh2(i,:);
        end
```

```
            if  j==iS(i,2);
                A2(i,j*2-1:j*2)=-dh2(i,:);
            end
        end
end

sqr_lambda=(lambda)^0.5*eye(length(x)*2);
b1=A1*x(:)-h1;
b2=A2*x(:)-h2;
A=[A1;A2;sqr_lambda];
b=[b1;b2;(lambda)^0.5*x(:)];

x_next=A\b;
end
```

Finally, using the dataset `lmdata1.mat`, where it was provided an initial x (xinit) and using $\lambda_0 = 1$ and $\epsilon = 10^{-6}$, the LM predicted positions can be observed in figure 50. In figure 51, it can be seen the evolution of $||\nabla f(x)||$. By the observation of the graph, it is concluded that this algorithm can converge to a solution in a small number of iterations, in this case 17 were enough.

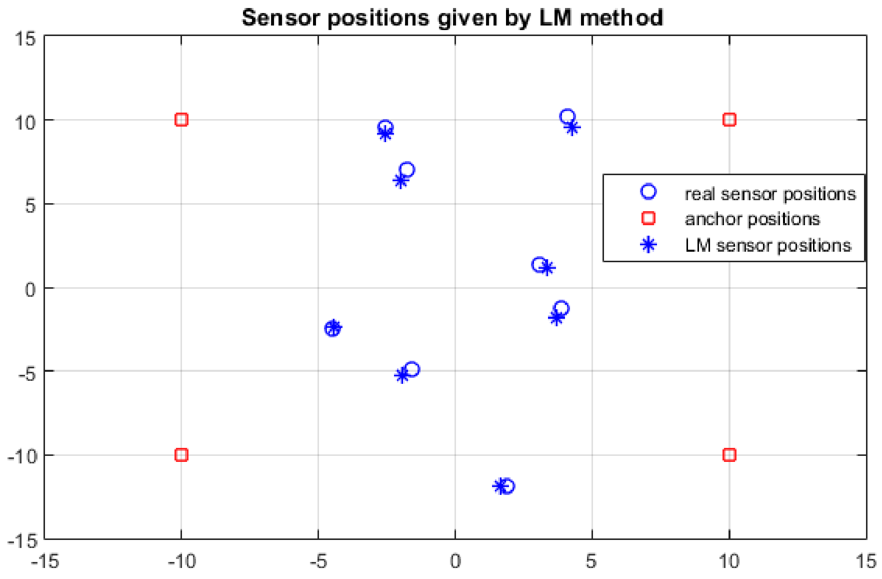


Figure 50: Network localization positions obtained, when the LM method is applied to problem 80 with the data from the file lmdata1.mat

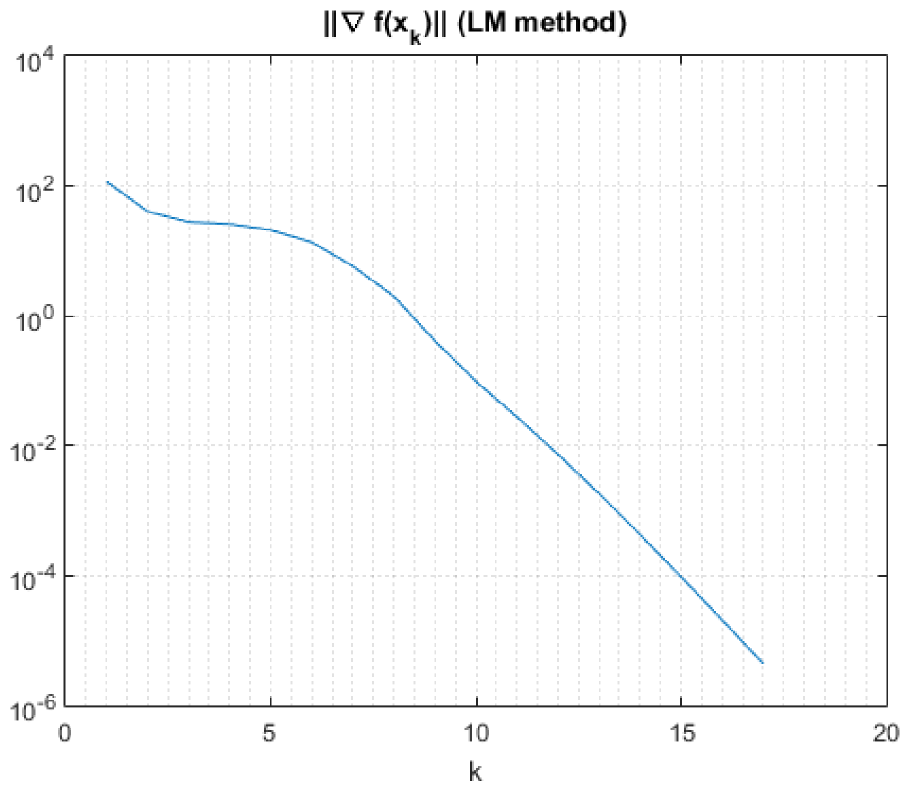


Figure 51: Norm of the gradient along the iterations of the LM method, when the LM method is applied to problem 80 with the data from the file lmdata1.mat

The code used to compute to algorithm is showed bellow:

```
load 'processed_data.mat'

%constants for LM Algorithm
lambda=1;
e=10^-6;
max_iterations=100;

%get initial values
xk=[xinit(1:2:end),xinit(2:2:end)]';
[f_k,g_k,h1_k,h2_k,dh1_k,dh2_k,n1,n2,norm1,norm2]=getFunctions(xk);

%LM algorithm
for i=1:max_iterations

     if norm(g_k(:))< e
         break;
     end


```

```
    %solve the LS to get the next x
    xk_1=getNextX(xk,h1_k,h2_k,dh1_k,dh2_k,lambda);
    xk_1=[xk_1(1:2:end),xk_1(2:2:end)]';

    %get f, the gradient of f and auxiliary functions of the new x
    [f_k1,g_k1,h1_k1,h2_k1,dh1_k1,dh2_k1]=getFunctions(xk_1);

    if f_k1 <= f_k
        %update values
        xk=xk_1;
        f_k=f_k1;
        g_k=g_k1;
        h1_k=h1_k1;
        h2_k=h2_k1;
        dh1_k=dh1_k1;
        dh2_k=dh2_k1;
        lambda=0.7*lambda;
    else
        lambda=2*lambda;
    end
end
```

2.9 Task 9

In this task, it was asked to proceed as before, but this time with no given initial value and to a different set of observations (`lmdata2.mat`). The LM was computed from 10000 random initializations and two solutions were obtained, one with the cost function of 107.3 and the other with a cost of 4.495. The solution with lowest cost function is presented bellow:

$$x = \begin{bmatrix} -5.9916 & -8.8767 & 4.2174 & -7.6749 & 0.7416 & 2.0511 & 10.9249 & -1.3723 \\ 9.1769 & -1.9330 & 0.4480 & 3.9664 & -4.0953 & 3.6757 & -0.8923 & -2.6921 \end{bmatrix}$$

In figure 52, it can be observed the anchor positions and two solutions obtained applying the algorithm.

The code that implements task 9 is the following:

```
getSelectionMatrices('datasets/lmdata2.mat');
load 'procesed_data.mat'

%constants for LM Algorithm
lambda=1;
e=10^-6;
max_iterations=100;
number_of_xinit=10000;

f_best=10^100;
i_max=0;
```

```
for n=1:number_of_xinit

    if n≠ 1 && f_k < f_best
        f_best=f_k;
        xk_best=xk;
        if i>i_max
            i_max=i
        end
    end

    %initial conditions
    xk=randi([-10^5 10^5],2,8);
    [f_k,g_k,h1_k,h2_k,dh1_k,dh2_k,n1,n2,norm1,norm2]=getFunctions(xk);

    %LM algorithm equal to Task 8

end
```

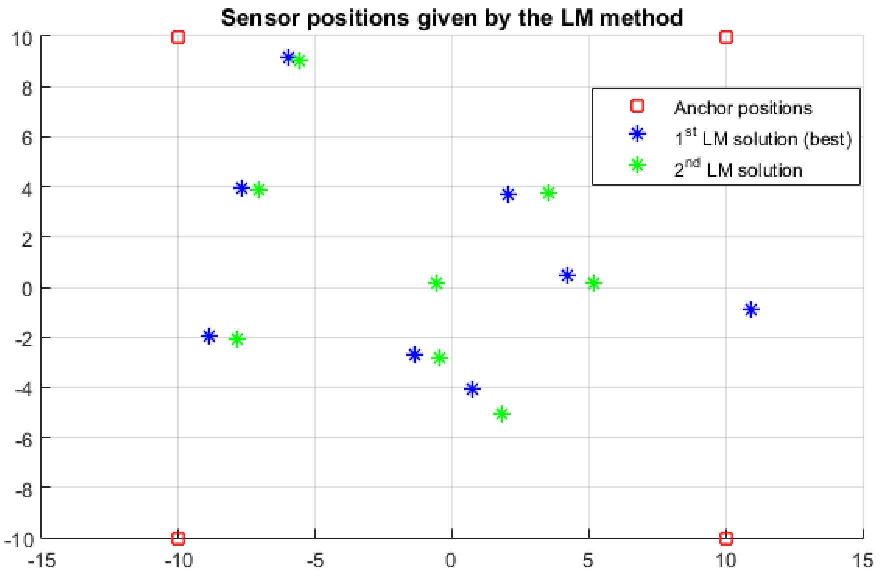


Figure 52: Network localization positions obtained, when the LM method is applied to problem 80 with the data from the file lmdata2.mat

3 Part 3

In this part we were asked to solve two problems that were also talked about in the first part of the project.In the first problem, the objective was to find a closed-form expression for the distance from a point p to a disk $D(c, r) = \{x : ||x - c||_2 \leq r\}$.

The second one, the objective was to find a closed-form solution to one of problems that were discussed in the first part of the project, where we had to solve the problem so the robot passes, as close as possible, to a series of intermediate points along its journey, using different regularizers.

3.1 Task 1

In this task we were asked to solve the optimization problem 86 bellow, that finds a closed-form expression for the distance from a point p to a disk $D(c, r) = \{x : ||x - c||_2 \leq r\}$

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p - y\|_2 \\ \text{subject to} & y \in D(c, r). \end{array} \tag{86}$$

This problem can be transformed so we have a close form of the expression $y \in D(c, r)$, as shown in the equation 87 bellow:

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p - y\|_2 \\ \text{subject to} & \|y - c\|_2 \leq r \end{array} \tag{87}$$

To solve this problem using the KKT System we had to transform the functions so we could get rid of the norm. What we did was to put the square norm instead of the norm so, when we compute the gradient, we have an expression that we can work with to find a closed-form for the distance from the point to the disk. The problem 87 can now be transformed in the problem 88:

$$\begin{array}{ll} \underset{y \in \mathbf{R}^n}{\text{minimize}} & \|p - y\|_2^2 \\ \text{subject to} & \|y - c\|_2^2 \leq r^2 \end{array} \tag{88}$$

$$\begin{cases} \nabla F(y) + \mu \nabla g(y) = 0 \\ g(y) \leq 0 \\ \mu \geq 0 \\ \mu g(y) = 0 \end{cases} \tag{89}$$

Using the last three conditions of the system we can transform it in:

$$\begin{cases} \nabla F(y) + \mu \nabla g(y) = 0 \\ g(y) = 0 \\ \mu \geq 0 \end{cases} \tag{90}$$

That makes sense because we know that the point of the disk that is closest to the point p is on the boundary of the disk. We know that,

$$\begin{cases} F(y) = \|p - y\|_2^2 \\ g(y) = \|y - c\|_2^2 - r^2 \end{cases} \tag{91}$$

So using the KKT System we have:

$$\begin{cases} \nabla F(y) + \mu \nabla g(y) = 0 \\ g(y) = 0 \end{cases} \Leftrightarrow \begin{cases} -2(p - y) + \mu 2(y - c) = 0 \\ \|y - c\| = R \end{cases} \Leftrightarrow \tag{92}$$

$$\begin{cases} 2(y - p) + \mu 2(y - c) = 0 \\ \|y - c\| = R \end{cases} \Leftrightarrow \begin{cases} (y - p) + \mu(y - c) = 0 \\ \|y - c\| = R \end{cases} \tag{93}$$

Solving the first equation of the system we have:

$$(y - p) + \mu(y - c) = 0 \Leftrightarrow y - p + \mu y - \mu c = 0 \Leftrightarrow y(1 + \mu) = p + \mu c \tag{94}$$

Isolating y we have

$$y = \frac{p + \mu c}{1 + \mu} \tag{95}$$

Solving now the second equation of the system, substituting y solved in the equation 95 we have:

$$\|y - c\| = R \Leftrightarrow \left\| \frac{p + \mu c}{1 + \mu} - c \right\| = R \Leftrightarrow \left\| \frac{p + \mu c - c - \mu c}{1 + \mu} \right\| = R \Leftrightarrow \tag{96}$$

$$\left\| \frac{p - c}{1 + \mu} \right\| = R \Leftrightarrow \frac{\|p - c\|}{|1 + \mu|} = R \Leftrightarrow |1 + \mu| = \frac{\|p - c\|}{R} \tag{97}$$

Isolating μ we have:

$$\mu = -1 \pm \frac{\|p - c\|}{R} \tag{98}$$

Substituting the μ of the equation 98 on the y of the equation 95 we have:

$$y = \frac{p + \mu c}{1 + \mu} \Leftrightarrow \frac{p + (-1 + \frac{\|p-c\|}{R})c}{1 + (-1 + \frac{\|p-c\|}{R})} \Leftrightarrow \frac{p - c + c(\frac{\|p-c\|}{R})}{\frac{\|p-c\|}{R}} \tag{99}$$

From the equation 99 we can now solve it so we can have a closed-form solution for $y*$, shown in the equation 100:

$$y* = c + \frac{R}{\|p - c\|}(p - c) \tag{100}$$

3.2 Task 2

In this task, we were given 3 problems from which we had to choose one to find the closed-form solution. In order to apply the *KKT* theorem, the function $f(x)$ has to be differentiable, so we chose the Problem A, because it is the only one with squared l_2 norms.

- Problem A:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \sum_{k=1}^{K} \|Ex(\tau_k) - w_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 \\ \text{subject to} & x(0) = x_{\text{initial}} \\ & x(T) = x_{\text{final}} \\ & x(t+1) = Ax(t) + Bu(t), \quad \text{for } 0 \leq t \leq T-1; \end{array} \tag{101}$$

In the first part of the project, $x(t)$ is a vector of the position and velocity of the robot and E is the matrix the selects the position of $x(t)$. So, $x = \begin{bmatrix} p(t) & v(t) \end{bmatrix}^T$ and $p(t) = Ex(t)$.

To get solve the *KKT* System as before, we have to reformulate the expressions.

For $x = \begin{bmatrix} x(0) & \dots & x(T) \end{bmatrix}^T$ and $u = \begin{bmatrix} u(0) & \dots & u(T) \end{bmatrix}^T$, we can make a change of variable y:

$$y = \begin{bmatrix} x \\ u \end{bmatrix} \tag{102}$$

As it was done before with the matrix E, we are going to need matrices in order to select only the values we want. So we created 7 selecting matrices:

- X:

$$Xy = x \tag{103}$$

- U:

$$Uy = u \tag{104}$$

- X_M:

$$X_M y = \begin{bmatrix} Ex(1) & \dots & Ex(T-1) \end{bmatrix}^T \tag{105}$$

- U_{M1}:

$$U_{M1} y = \begin{bmatrix} u(1) & \dots & u(T-2) \end{bmatrix}^T \tag{106}$$

- U_{M2}

$$U_{M2} y = \begin{bmatrix} u(2) & \dots & u(T-1) \end{bmatrix}^T \tag{107}$$

- X_1

$$X_1 y = \begin{bmatrix} 0 & x(0) & \dots & x(T-2) & 0 \end{bmatrix}^T \tag{108}$$

- U_1

$$U_1 y = \begin{bmatrix} 0 & u(0) & \dots & u(T-2) & 0 \end{bmatrix}^T \tag{109}$$

For $w = \begin{bmatrix} w_1 & \dots & w_K \end{bmatrix}^T$ and for $w' = X_M^{-1} w$, we can put $f(x)$ in the form of 111.

$$f(Y) = ||X_M y - w||^2 + \lambda || \underbrace{(U_{M2} - U_{M1})}_{U_{M3}} y||^2 \tag{110}$$

$$= (y - w')^T X_M^T X_M (y - w') + \lambda y^T U_{M3} y \tag{111}$$

Proceeding in similar way to the constrains, we get, for $d = \begin{bmatrix} x_{initial} & \dots & x_{final} \end{bmatrix}^T$, we get:

$$h(y) = Xy - AX_1 y - BU_1 y - d \tag{112}$$

For simplicity, we can attribute different names to the matrices:

$$X_M^T X_M = M \tag{113}$$

$$U_{M3}^T U_{M3} = M_3 \tag{114}$$

$$X - AX_1 - BU_1 = C \tag{115}$$

With the equations in the proper form, the gradients are easily computed as we can see in 116 and 117.

$$\nabla f(y) = 2M(y - w') + \lambda 2 M_3 y \tag{116}$$

$$\nabla h(y) = C^T \tag{117}$$

Solving the *KKT* System:

$$\begin{cases} 2M(y - w') + \lambda 2 M_3 y + C^T \beta = 0 \\ Cy - d = 0 \end{cases} \Leftrightarrow \tag{118}$$

$$\Leftrightarrow \begin{cases} 2 \underbrace{(M + \lambda M_3)}_{L} y = 2Mw' - C^T \beta \\ Cy = d \end{cases} \Leftrightarrow \tag{119}$$

$$\Leftrightarrow \begin{cases} y = L^{-1} M w' - \frac{1}{2} L^{-1} C^T \beta \\ CL^{-1} M w' - \frac{1}{2} C L^{-1} C^T \beta = d \end{cases} \Leftrightarrow \tag{120}$$

$$\Leftrightarrow \begin{cases} y = L^{-1} M w' - \frac{1}{2} L^{-1} - C^T \beta \\ \beta = 2(C^T)^{-1}(Mw' - LCd) \end{cases} \tag{121}$$

Then we substitute β into the first equation of the system , and we get the solution:

$$y^* = L^{-1} M w' - M w' - LCd = (L^{-1} - I) M w' - LCd \tag{122}$$