Optimization and Algorithms
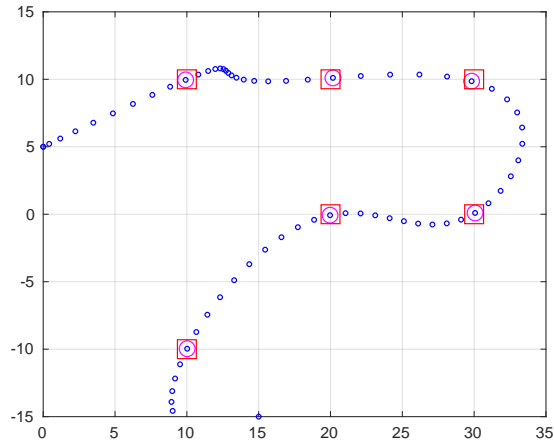## Project report

Group 2

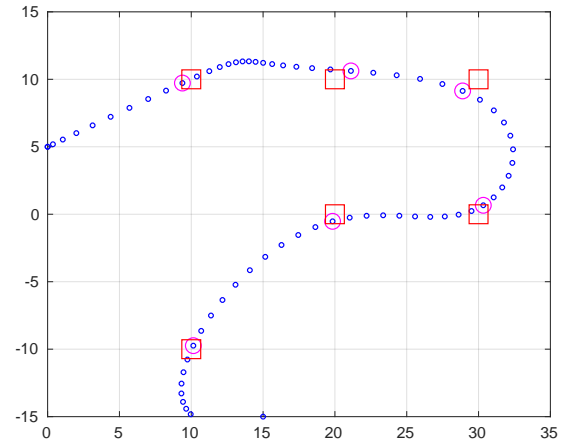Miguel Pinho 80826, Pedro Mendes 81046, Duarte Dias 81356
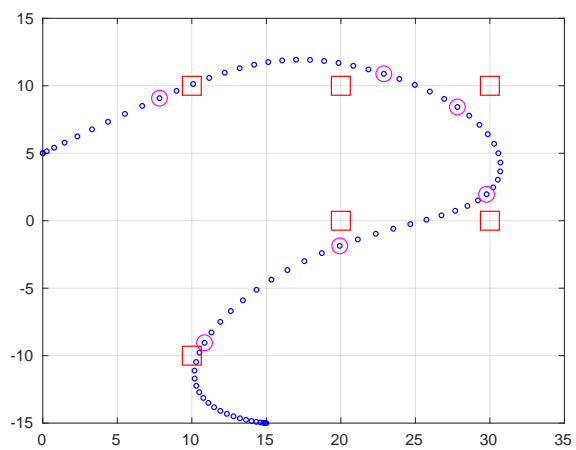
Part 1

Task 1

a)

In Figure 1 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times τ_k, for the different values of λ parameter, when the cost function uses ℓ_2^2 regularizer.

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

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

(d) $\lambda = 10^0$

(e) $\lambda = 10^1$

(f) $\lambda = 10^2$

2

(g) $\lambda = 10^3$

Figure 1: Positions of the robot from $t = 0$ to $t = T$ for the different values of λ with the ℓ_2^2 regularizer.

b)

In Figure 2 are represented the plots of the optimal control signal $u(t)$, from $t = 0$ to $t = T - 1$, for the different values of λ.
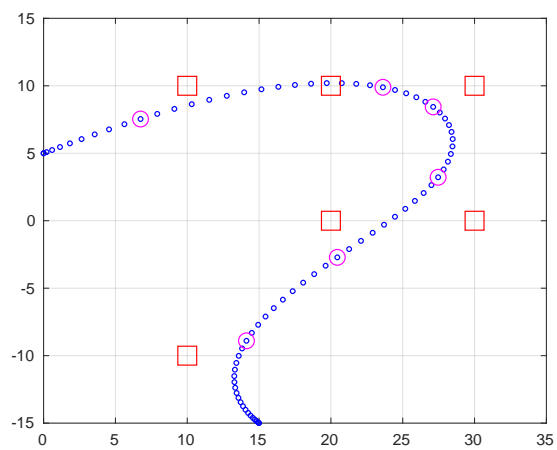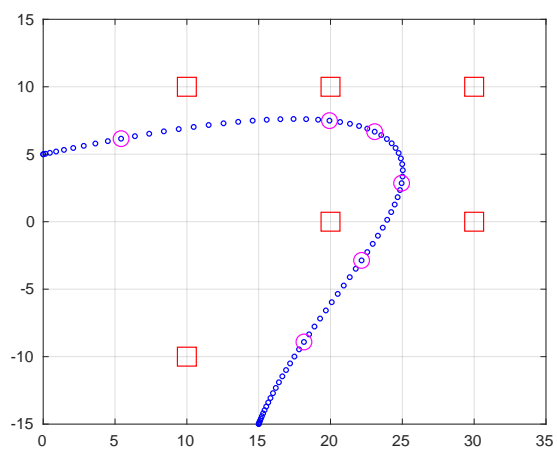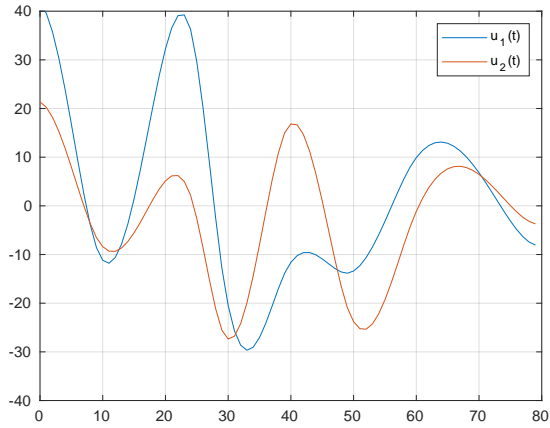

(a) $\lambda = 10^{-3}$

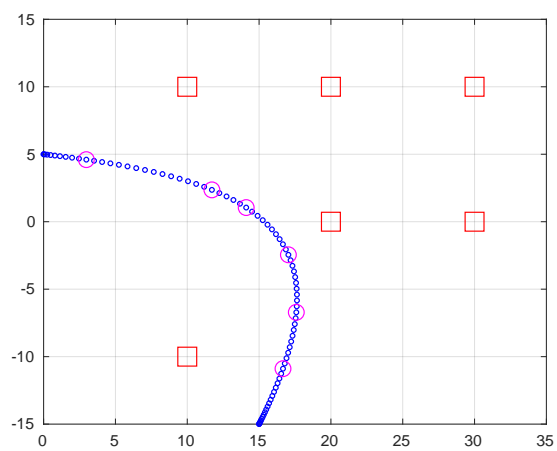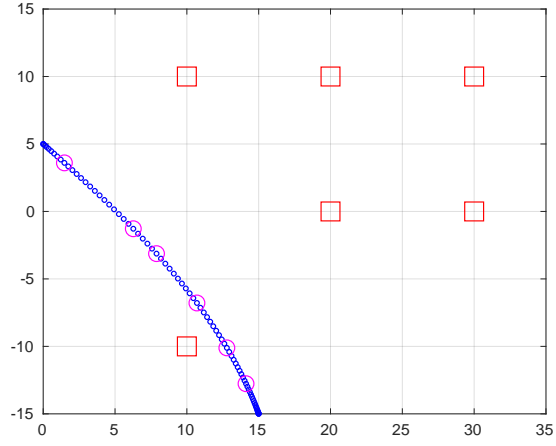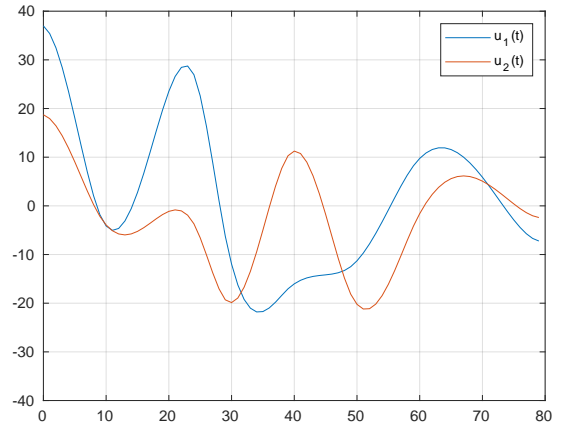
(b) $\lambda = 10^{-2}$
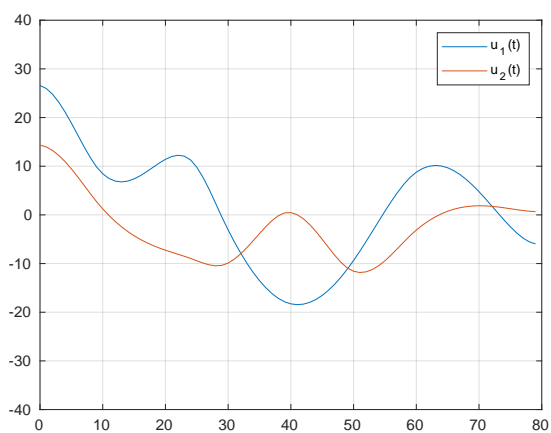
(c) $\lambda = 10^{-1}$
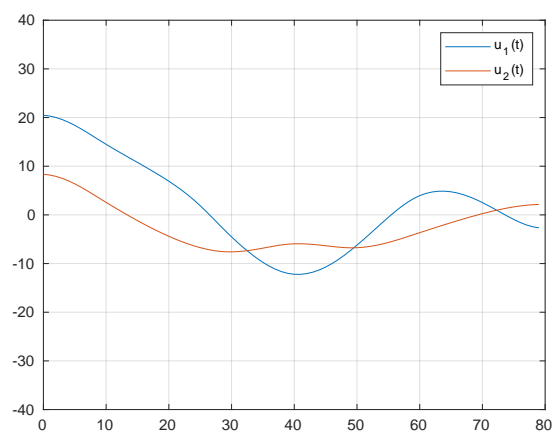
(d) $\lambda = 10^{0}$
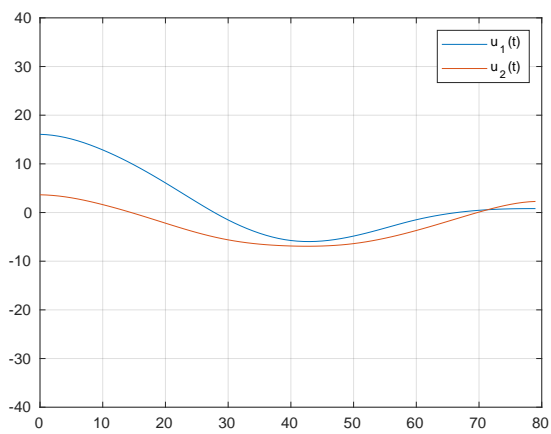
(e) $\lambda = 10^{1}$
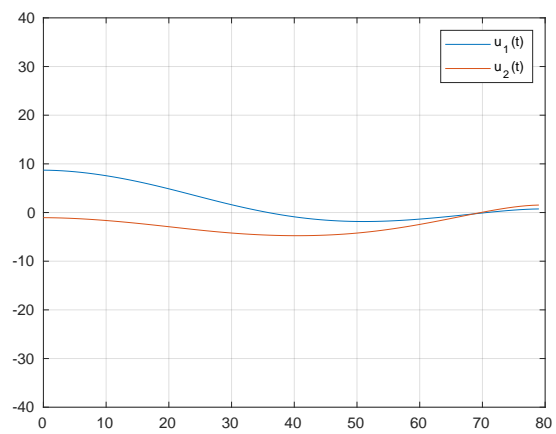
(f) $\lambda = 10^{2}$
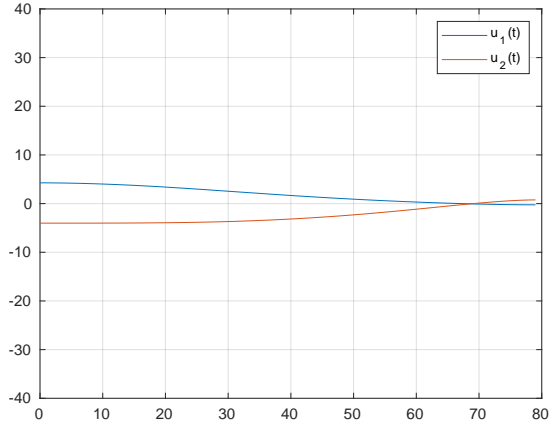
(g) $\lambda = 10^3$

Figure 2: Optimal control signal $u(t)$ from $t = 0$ to $t = T - 1$ for the different values of λ, with the ℓ_2^2 regularizer.

c)

The Table 1 contains the number of changes in the control signal from $t = 0$ to $t = T - 1$, for the different values of λ.

λ	Number of changes in control signal
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 changes of the optimal control signal from $t = 1$ to $t = T - 1$ for the different values of λ, with the ℓ_2^2 regularizer.

d)

The mean deviations from the waypoints, which is given by

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

, are determined in Table 2, for the different values of λ.

λ	Mean deviation
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 for the different values of λ, with the ℓ_2^2 regularizer.

```matlab
clear all;
close all;

%load the workspace
load('dataA.mat');

for L=1:1:length(lambda)

    % solve optimization problem for this lambda
    cvx_begin quiet
        variable x(4, T+1);% columns are R^4 state vectors
        variable u(2,   T); % columns are R^2 control signal

        % cost function
        f_waypoints = 0;
        for i=1:1:k
            f_waypoints = f_waypoints + square_pos( norm(E * x(:, tau(i) +
                1) - w(:, i), 2) );
        end

        f_regularizer = 0;
        for i=2:1:T
            f_regularizer = f_regularizer + square_pos ( norm(u(:, i)-u(:,
                i-1), 2) );
        end

        f = f_waypoints + lambda(L) * f_regularizer;
        minimize( f );

        % subject to
        x(:,1)   == initialx;
        x(:,T+1) == finalx;
```

```matlab
32          for t = 1:T
33              norm( u(:,t)) <= Umax;
34          end
35
36          for t = 1:T
37              x(:, t+1) == A * x(:, t) + B * u(:, t);
38          end
39      cvx_end;
40
41      % plot the results
42      plot_graphs(x, u, tau+1, w);
43
44      % save plots
45      str = num2str(lambda(L));
46      save_str = strrep(str,'.',',');
47      saveas(figure(1), strcat('Figures/task1/lambda_',save_str , '_position.
            ↪ pdf'));
48      saveas(figure(2), strcat('Figures/task1/lambda_', save_str, '_control.
            ↪ pdf'));
49
50      % changes in control signal
51      counter = 0;
52      for t=2:1:T
53          if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
54              counter = counter + 1;
55          end
56      end
57
58      %calculation of mean deviation
59      m=0;
60      for i=1:1:k
61          m = m + norm(E * x(:, tau(i)+1) - w(:, i), 2);
62      end
63      mean_deviation = m/k;
64
65      str1 = ['Number of changes of the control signal = ', num2str(counter)
            ↪ ];
66      disp(str1);
67      str2 = ['Mean deviation = ', num2str(mean_deviation)];
68      disp(str2);
69
70      close all;
71  end
```

Listing 1: Script Task 1

Task 2

a)

In Figure 3 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times τ_k, for the different values of λ parameter, when the cost function uses ℓ_2 regularizer.
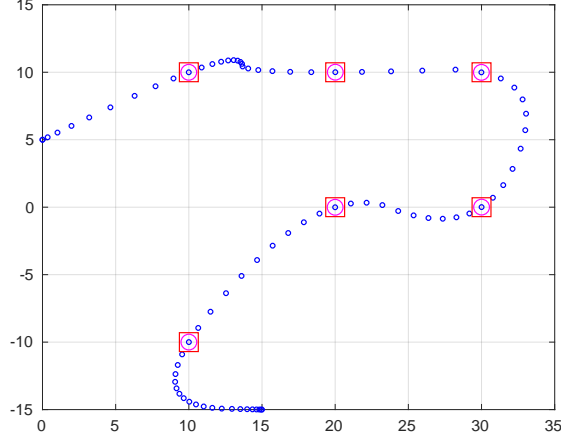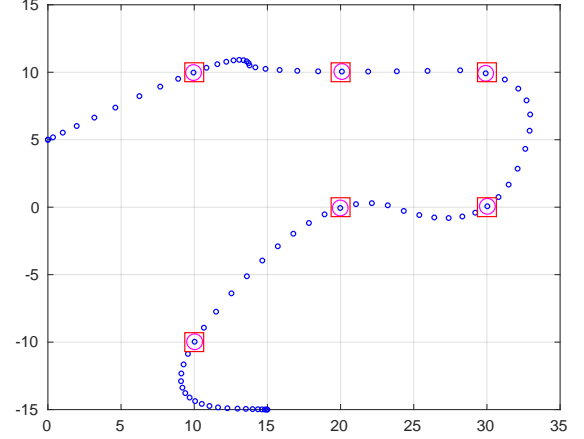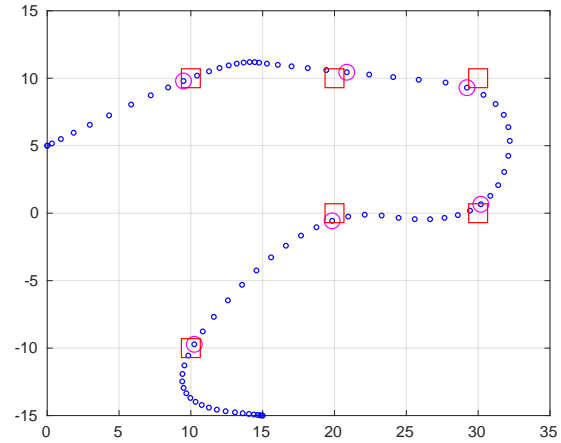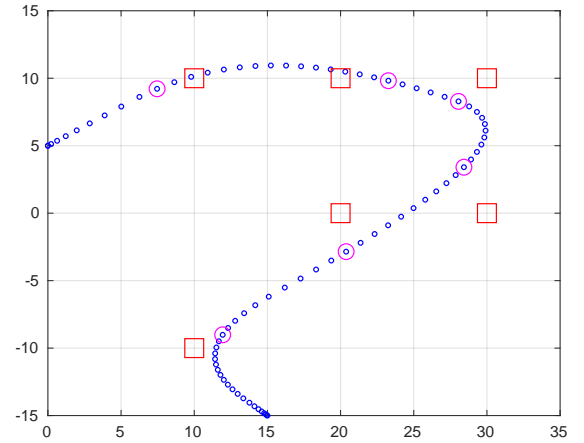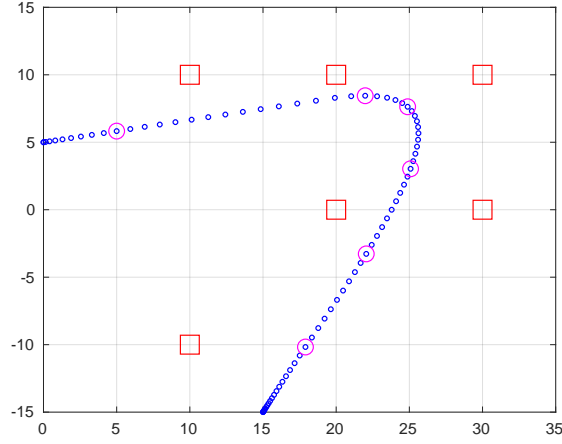
(a) $\lambda = 10^{-3}$
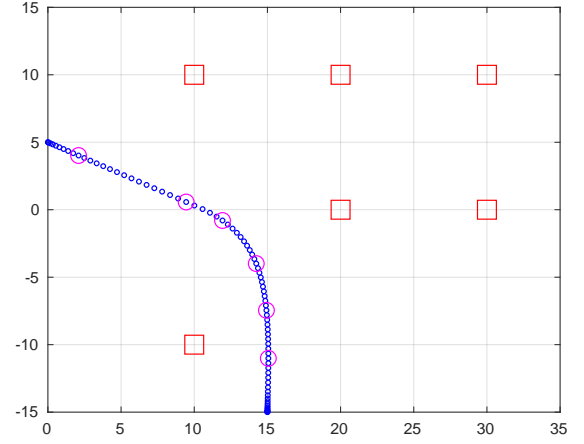
(b) $\lambda = 10^{-2}$
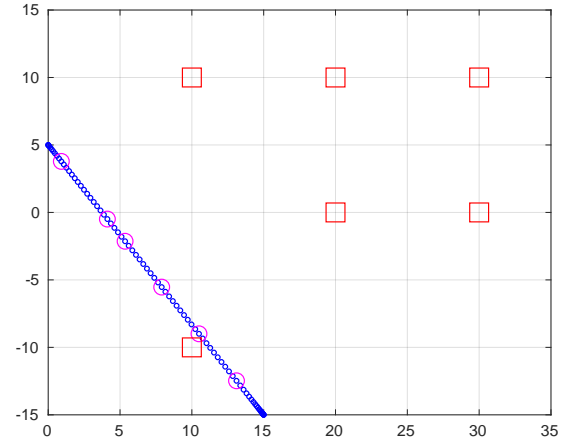
(c) $\lambda = 10^{-1}$

(d) $\lambda = 10^0$

(e) $\lambda = 10^1$

(f) $\lambda = 10^2$

(g) $\lambda = 10^3$

Figure 3: Positions of the robot from $t = 0$ to $t = T$ for the different values of λ, with the ℓ_2 regularizer.

b)

In Figure 4 are represented the plots of the optimal control signal $u(t)$, from $t = 0$ to $t = T - 1$, for the different values of λ.
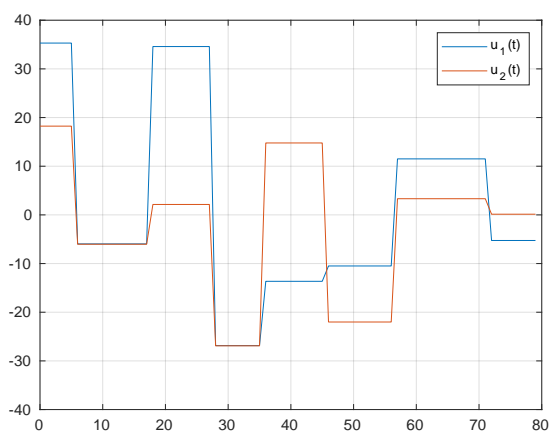
(a) $\lambda = 10^{-3}$
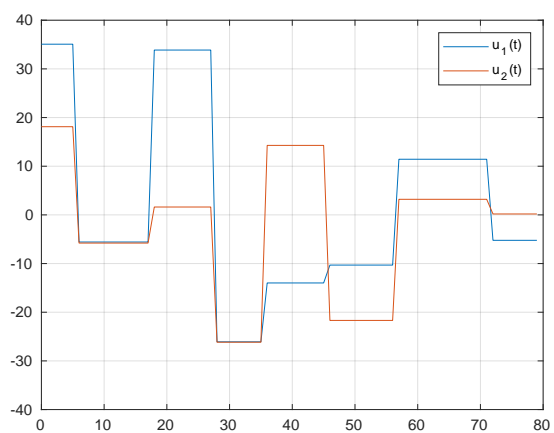
(b) $\lambda = 10^{-2}$
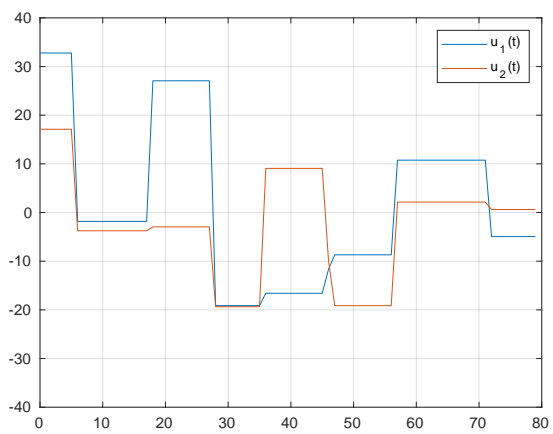
(c) $\lambda = 10^{-1}$
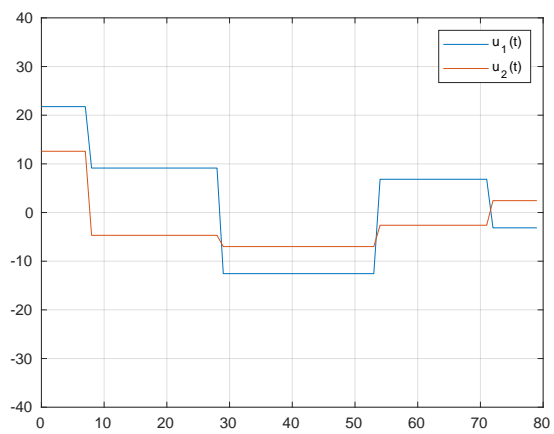
(d) $\lambda = 10^0$
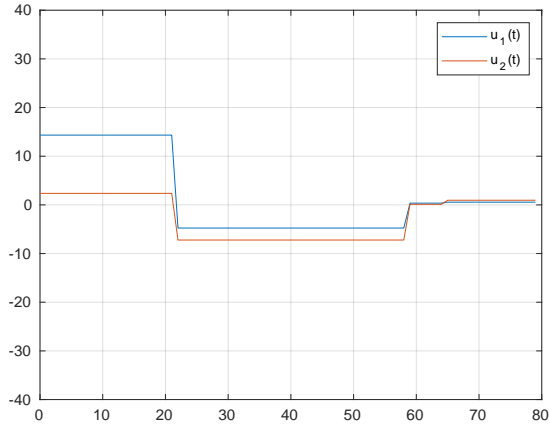
(e) $\lambda = 10^1$
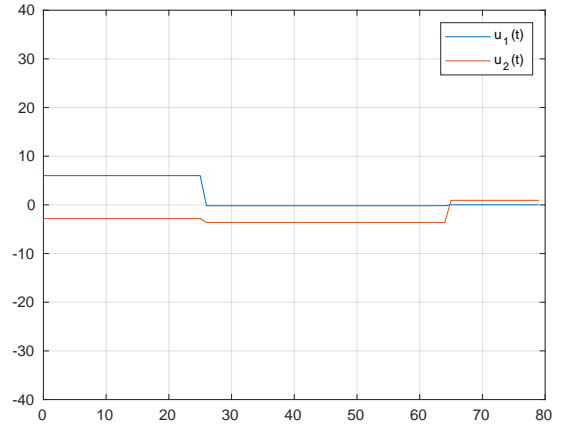
(f) $\lambda = 10^2$
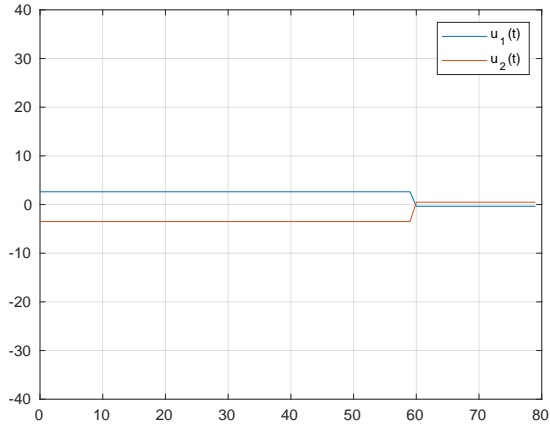
(g) $\lambda = 10^3$

Figure 4: Optimal control signal $u(t)$ from $t = 0$ to $t = T - 1$ for the different values of λ, with the ℓ_2 regularizer.

c)

The Table 3 contains the number of changes in the control signal from $t = 0$ to $t = T - 1$, for the different values of λ.

λ	Number of changes in control signal
10^{-3}	9
10^{-2}	8
10^{-1}	11
10^{0}	5
10^{1}	4
10^{2}	4
10^{3}	2

Table 3: Number of changes of the optimal control signal from $t = 1$ to $t = T - 1$ for the different values of λ, with the ℓ_2 regularizer.

d)

The mean deviations from the waypoints, for the different values of λ, are determined in Table 4.

λ	Mean deviation
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 for the different values of λ, with the ℓ_2 regularizer.

```matlab
clear all;
close all;

%load the workspace
load('dataA.mat');

for L=1:1:length(lambda)

    % solve optimization problem
    cvx_begin quiet
        variable x(4, T+1); % columns are R^4 state vectors
        variable u(2,   T); % columns are R^2 control signal vectors

        %cost function
        f_waypoints = 0;
```

```matlab
16          for i=1:1:k
17              f_waypoints = f_waypoints + square_pos( norm(E * x(:, tau(i) +
       ↪ 1) − w(:, i), 2) );
18          end
19
20          f_regularizer = 0;
21          for i=2:1:T
22              f_regularizer = f_regularizer + norm(u(:, i)−u(:, i−1), 2);
23          end
24
25          f = f_waypoints + lambda(L) * f_regularizer;
26          minimize( f );
27
28          %subject to
29          x(:,1) == initialx;
30          x(:,T+1) == finalx;
31
32          for t = 1:T
33              norm( u(:,t)) ≤ Umax;
34          end
35
36          for t = 1:T
37              x(:,t+1) == A*x(:,t) + B*u(:,t);
38          end
39
40      cvx_end;
41
42      %plot the results
43      plot_graphs(x, u, tau+1, w);
44
45      % save plots
46      str = num2str(lambda(L));
47      save_str = strrep(str,'.',',');
48      saveas(figure(1), strcat('Figures/task2/lambda_', save_str , '_position
       ↪ .pdf'));
49      saveas(figure(2), strcat('Figures/task2/lambda_', save_str , '_control.
       ↪ pdf'));
50
51      %changes in control signal
52      counter = 0;
53      for t=2:1:T
54          if norm(u(:,t)−u(:,t−1), 2) > power(10,−6)
55              counter = counter + 1;
56          end
57      end
58
59      %calculation of mean deviation
60      m=0;
61      for i=1:1:k
62          m = m + norm(E * x(:, tau(i)+1) − w(:, i), 2);
63      end
```

```
64      mean_deviation = m/k;

65

66      str1 = ['Number of changes of the control signal = ', num2str(counter)
            ↪ ];
67      disp(str1);
68      str2 = ['Mean deviation = ', num2str(mean_deviation)];
69      disp(str2);

70

71      close all
72  end
```

Listing 2: Script Task 2

Task 3

a)

In Figure 5 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times τ_k, for the different values of λ parameter, when the cost function uses ℓ_1 regularizer.
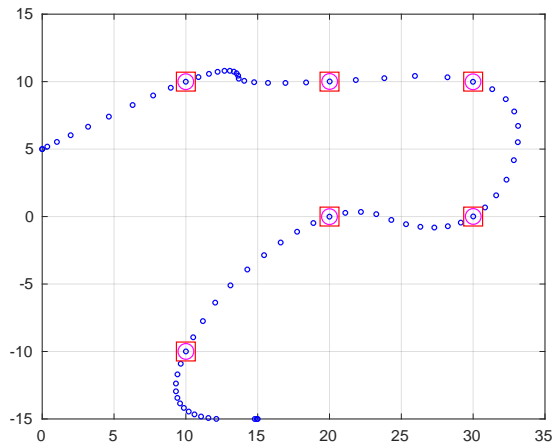
(a) $\lambda = 10^{-3}$
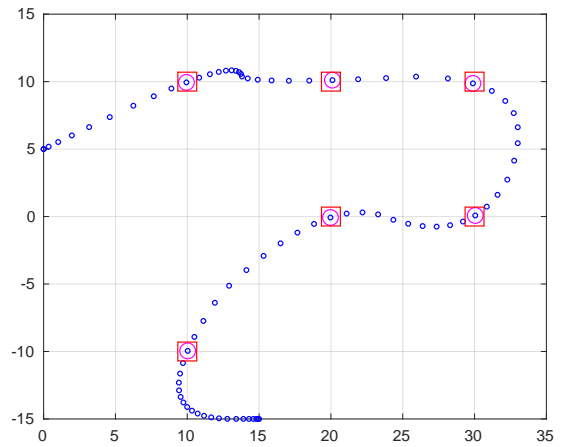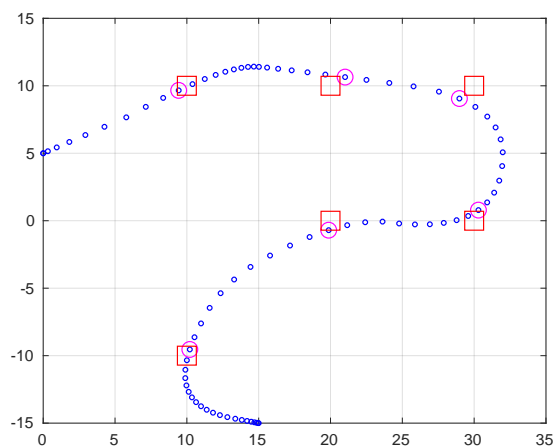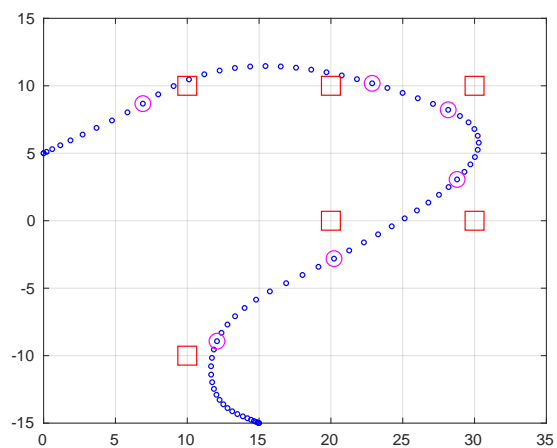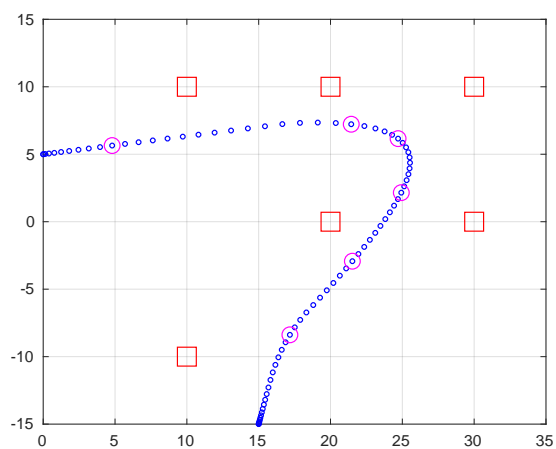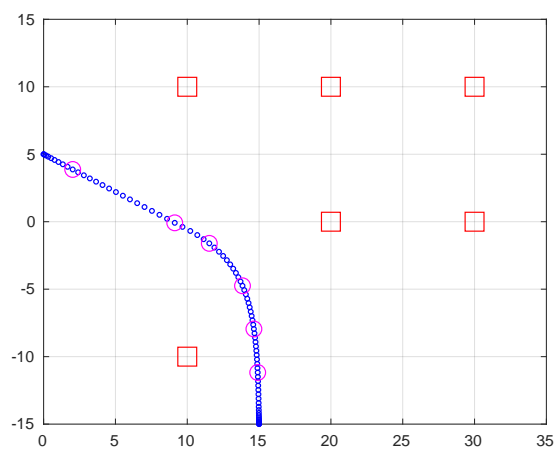
(b) $\lambda = 10^{-2}$
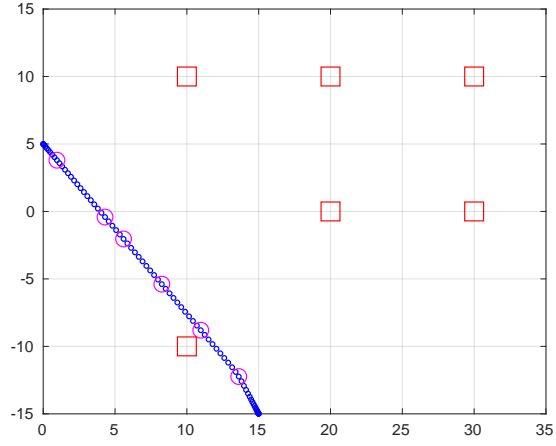
(c) $\lambda = 10^{-1}$

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

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

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

(g) $\lambda = 10^3$

Figure 5: Positions of the robot from $t = 0$ to $t = T$ for the different values of λ, with the ℓ_1 regularizer.

b)

In Figure 6 are represented the plots of the optimal control signal $u(t)$, from $t = 0$ to $t = T - 1$, for the different values of λ.

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

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

16
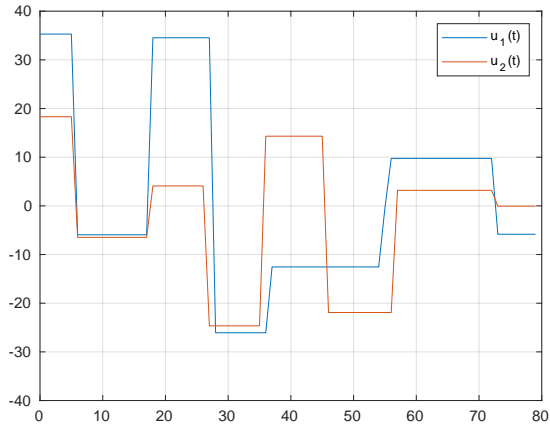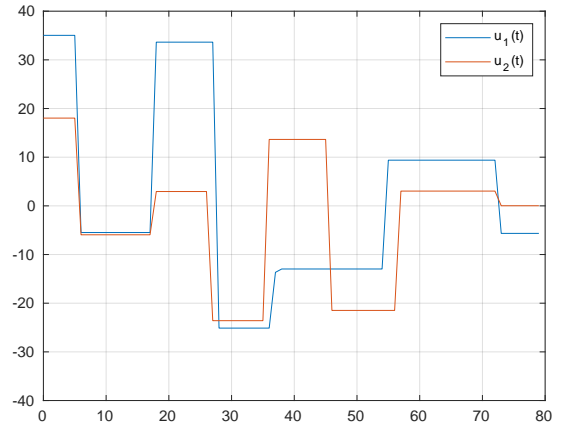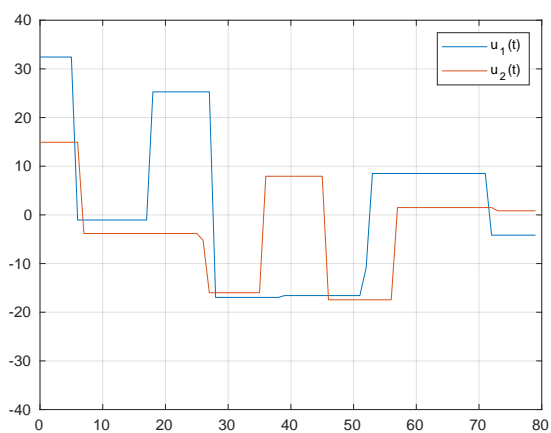
(c) $\lambda = 10^{-1}$
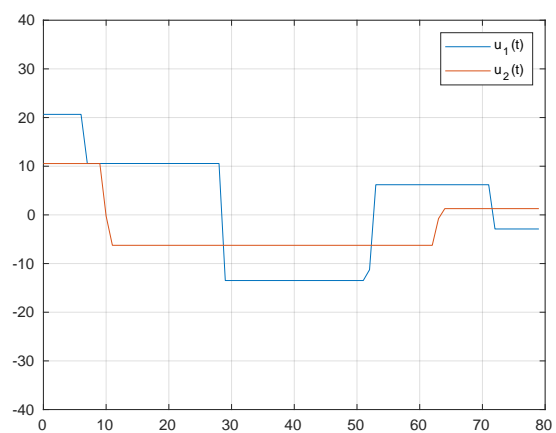
(d) $\lambda = 10^{0}$
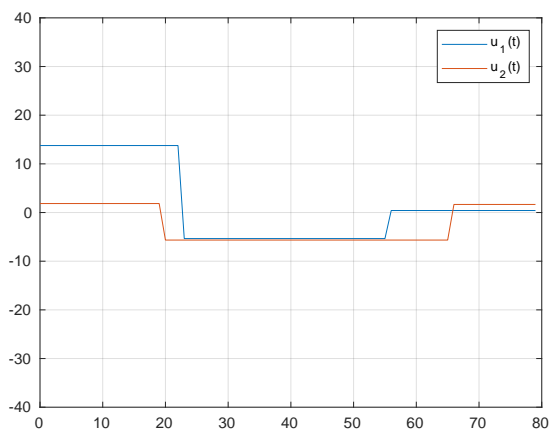
(e) $\lambda = 10^{1}$
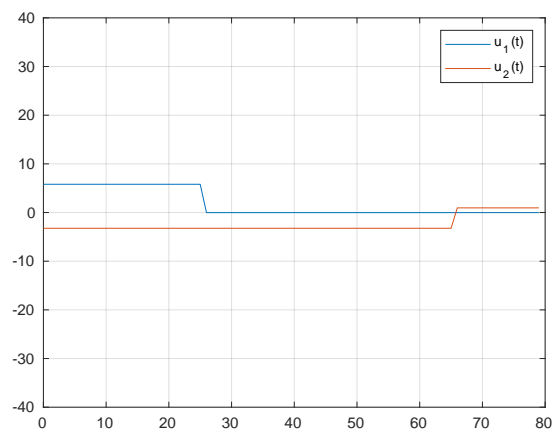
(f) $\lambda = 10^{2}$
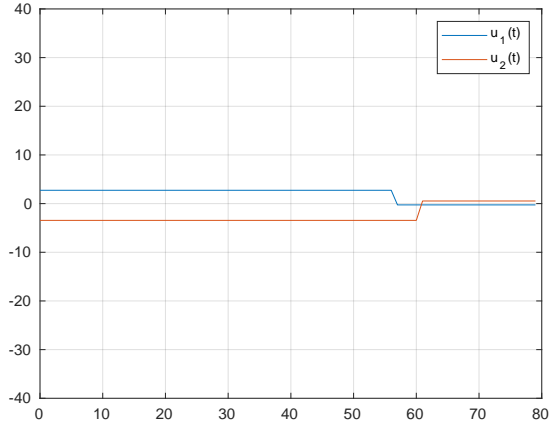
(g) $\lambda = 10^3$

Figure 6: Optimal control signal $u(t)$ from $t = 0$ to $t = T - 1$ for the different values of λ, with the ℓ_1 regularizer.

c)

The Table 5 contains the number of changes in the control signal from $t = 0$ to $t = T - 1$, for the different values of λ.

λ	Number of changes in control signal
10^{-3}	12
10^{-2}	12
10^{-1}	14
10^{0}	11
10^{1}	5
10^{2}	3
10^{3}	2

Table 5: Number of changes of the optimal control signal from $t = 1$ to $t = T - 1$ for the different values of λ, with the ℓ_1 regularizer.

d)

The mean deviations from the waypoints, for the different values of λ, are determined in Table 6.

18

λ	Mean deviation
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 for the different values of λ, with the ℓ_1 regularizer.

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('dataA.mat');
6
7   for L=1:1:length(lambda)
8
9       % solve optimization problem
10      cvx_begin quiet
11          variable x(4, T+1); % columns are R^4 state vectors
12          variable u(2,   T); % columns are R^2 control signal vectors
13
14          %cost function
15          f_waypoints = 0;
16          for i=1:1:k
17              f_waypoints = f_waypoints + square_pos( norm(E * x(:, tau(i) +
                  ↪ 1) - w(:, i), 2) );
18          end
19
20          f_regularizer = 0;
21          for i=2:1:T
22              f_regularizer = f_regularizer + norm(u(:, i)-u(:, i-1), 1);
23          end
24
25          f = f_waypoints + lambda(L) * f_regularizer;
26          minimize( f );
27
28          %subject to
29          x(:,1) == initialx;
30          x(:,T+1) == finalx;
31
32          for t = 1:T
33              norm( u(:,t)) <= Umax;
34          end
35
```

```matlab
36            for t = 1:T
37                x(:,t+1) == A*x(:,t) + B*u(:,t);
38            end
39
40        cvx_end;
41
42        %plot the results
43        plot_graphs(x, u, tau+1, w);
44
45        % save plots
46        str = num2str(lambda(L));
47        save_str = strrep(str,'.',',');
48        saveas(figure(1), strcat('Figures/task3/lambda_', save_str , '_position
             ↪ .pdf'));
49        saveas(figure(2), strcat('Figures/task3/lambda_', save_str , '_control.
             ↪ pdf'));
50
51        %changes in control signal
52        counter = 0;
53        for t=2:1:T
54            if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
55                counter = counter + 1;
56            end
57        end
58
59        %calculation of mean deviation
60        m=0;
61        for i=1:1:k
62            m = m + norm(E * x(:, tau(i)+1) - w(:, i), 2);
63        end
64        mean_deviation = m/k;
65
66        str1 = ['Number of changes of the control signal = ', num2str(counter)
             ↪ ];
67        disp(str1);
68        str2 = ['Mean deviation = ', num2str(mean_deviation)];
69        disp(str2);
70
71        close all
72    end
```

Listing 3: Script Task 3

Task 4

The cost function is given by

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

where the second term is called the regularizer, where p is the norm function used. In Task 1, $p = 2$ and $a = 2$, in Task 2, $p = 2$ and $a = 1$, and in Task 3, $p = 1$ and $a = 1$.

Analyzing the equation, we see that is composed by two terms, the first one that tries to minimize the distance of the robot to the target positions (waypoints), and the second one that tries to minimize the changes in the control signal, i.e., the signal is $u(t) - u(t - 1) = 0$ most of the time.

When the value of λ increases, the weight of the second term of the equation also increases, and the robot gives more importance to the changes in the control signal than the goal reaching the waypoints in time. So, when the value of λ increases, the mean deviation also increases (for the same regularizer), as we can see in Table 2, Table 3 and Table 5.

When the value of the λ parameter is high, the control signal is too constant to enable the possibility of the robot cross all the waypoints, as can be seen in Figure 1, Figure 3 and Figure 5, even ignoring them for the highest values. For lower values of λ, the robot has much more precision at passing in the waypoints, but at the cost of more complex control signals (Figure 2, Figure 4 and Figure 6).

The optimizer tries to minimize each value of the second sum of the cost function, which are the differences of the control signal between consecutive instants. When the ℓ_2^2 regularizer is used, however, these differences are all minimized but never reach zero, which is why the control signal changes for every instant (Table 1). With the ℓ_2 and ℓ_1 norms, however, several of these values are minimized exactly to zero, as can be seen by the small number of changes of the control signals (Figure 3 and Figure 5), which are piecewise constant signals as wanted ($u(t) = u(t - 1)$ for most of the values of t).

The derivative at a point of the square of the Euclidean norm decreases when the algorithm is converging to zero. So, using the ℓ_2^2 regularizer, the algorithm will never decrease below the given threshold (10^{-6}) and never reach 0. However, using the ℓ_2^2 regularizer, comparing to the other two regularizers used in this project, can be computed analytically, as explored in Part 3 Task 2, which is faster and more efficient to compute.

When the cost function uses the ℓ_2 regularizer, the algorithm can minimize the second term until zero (the same happens when using the ℓ_1 regularizer). The derivative is constant and during minimization, the algorithm decrements always the same value. The minimization can reach values very close to zero (below the threshold), which explains the small number of changes.

When the cost function uses the ℓ_1 regularizer, we observed a slight increase in the number of changes of the control signal. This regularizer can also minimize the function to zero. However, this increase in the number of changes in the control signal can be explain due to the fact that using the $norm1$, the components x and y of the each signal $u(t)$ are individually minimized. So, even if the x component of the signal maintains, if the y component of the signal changes, this provokes a change on the control signal. However, in practise, these changes in only one of the components of the motor signal are not as costly as changing both, so this norm can give better results.

The ℓ_1 regularizer is not so computational efficient as the previous two because the $norm - 1$ can have multiple solutions, instead of Euclidean norm that only has one solution.

The wish of transfer (wish 1) and bounded control (wish 2) are always respected, as they correspond to constraints of the optimization. The wish of passing close to the waypoints (wish 3), depends on the value of λ: the lower it is, the better it will be satisfied. The wish of simpler control (wish 4) is only really satisfied for norms ℓ_2 and ℓ_1, and the higher the λ the better.

Task 5

The minimum for the expression

$$min\|p - y\|_2 : y \in D(c, r)$$

is found for the y inside the disk that is closer to the point p. If p is inside the disk, it is that point itself, and the result is zero. If the point is outside, y can be found geometrically by drawing a line connecting c and p, and finding the intersection to the border of the disk. From Figure 7 the distance to the disk can be easily derived.

In order to calculate the distance from a given point p to a disk D(c, r) the following expression is used:

$$d(p, D(c, r)) = (\|p - c\|_2 - r)_+$$

Where the euclidean distance from the point p to the centre of the disk c is calculated and subtracted by the radius. In case the point is already inside the disk, that value is negative, but the $(\ldots)_+$ operator will give a result of zero, as the distance to the disk is null.
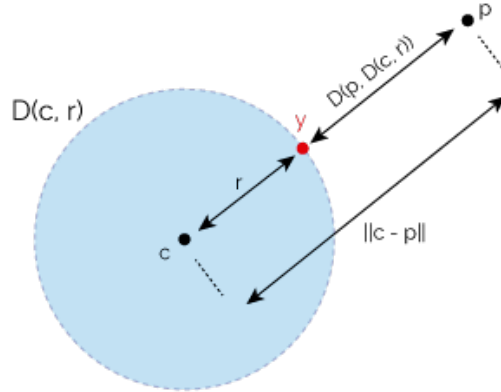
Figure 7: Distance of point to the disk, if it is outside it. The point y found by the minimization is highlighted.

Task 6

By solving problem (8) in CVX we obtain the following results:

a)
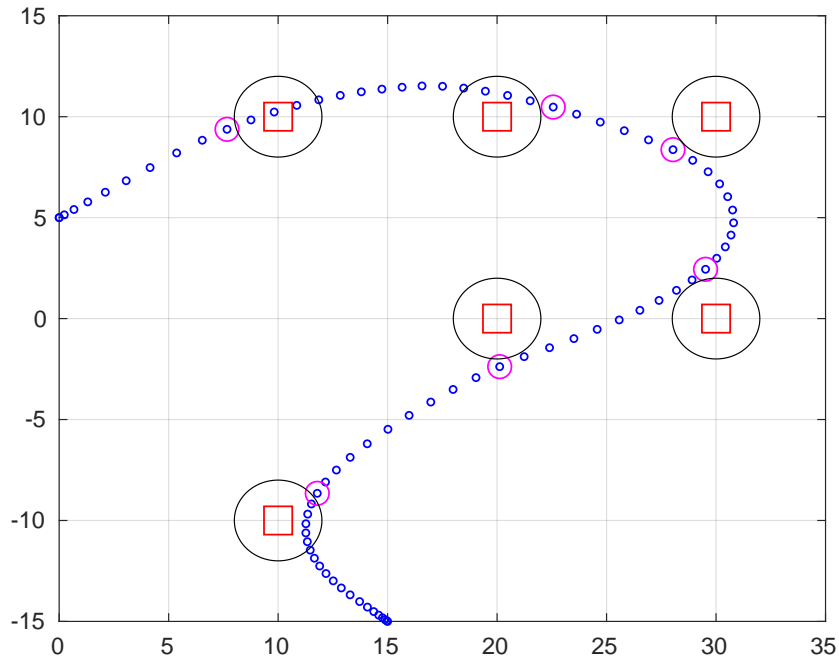
Figure 8: Positions of the robot from $t = 0$ to $t = T$, positions at the appointed times τ_k for $1 \leq k \leq K$ are marked in blue. The waypoints are signaled as red squares and the disk is drawn in black.

b)
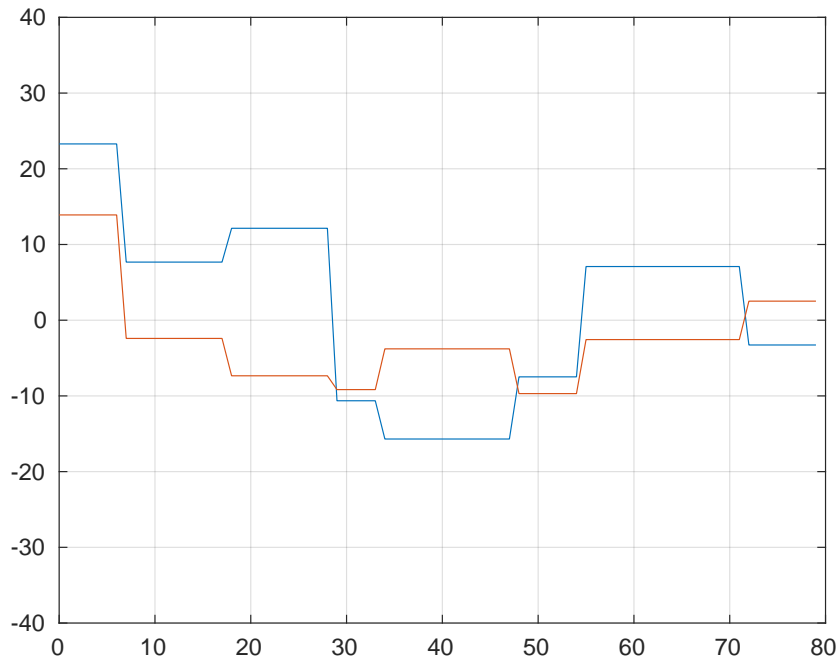
Figure 9: Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$.

23

c)

The optimal control signal changes from $t = 1$ to $t = T - 1$ a total of 9 times.

d)

The mean deviation to the center of the disks obtained was 2.796.

For comparison, the mean deviation to the border of the disks is also calculated, using:

$$\frac{1}{K} \sum_{k=1}^{K} \|d(Ex(\tau_k), D(c_k, r_k))\|, \tag{3}$$

and the value of 0.8451 was obtained.

e)

The results obtained are compared with lambda $= 0.1$ from Task 2, as that formulation could be seen as setting a radius of 0 for the disks. With a radius of 2, we obtained a smaller number of changes to the optimal control signal, which can be explained by considering that increasing the radius from 0 to 2 relaxes the restriction to the optimal path. The robot can now pass in any point of the disk to minimize the waypoint objective, so it has a much boarder solution space to explore for a simpler signal. The mean deviation to the center of the disk obtained increased, but this can be understood as a side effect of relaxing the objective. The points in the appointed times are close to the border of the disk, which matches with an increase of the mean deviation near the value of the radius.

It is interesting to note that the positions at the appointed times (τ_k) are never inside the circles or exactly in the border, which is a direct consequence of the way the optimization is formulated. The cost function is the sum of a term corresponding to the distance to the disk and a term corresponding to how simple the signal. The optimizer is trying the balance those two objectives, so when the points are already close to border of the disk, the distance term is already close to zero, priority is given to simplifying the signal. That is, there is no advantage for the optimizer to actually put the points inside the disk.

The mean deviation to the border of the disk is also computed, and interestingly that value is close to the mean deviation of the problem of Task 2, with radius 0, which shows the optimization converged to a point with a similar value for the first distance term of the cost function.

```matlab
1  clear all;
2  close all;
3
4  radius = 2;
5
6  %load the workspace
7  load('dataB.mat');
```

```matlab
8
9  %chosse the value of lambda
10 lambda1 = 0.1;
11
12 % solve optimization problem
13 cvx_begin quiet
14     variable x(4, T+1);% columns are R^4 state vectors
15     variable u(2,   T); % columns are R^2 control signal
16
17     %cost function
18     f_waypoints = 0;
19     for i=1:1:k
20         f_waypoints = f_waypoints + square_pos(norm((E * x(:, tau(i) + 1))
            ↪ − c(:, i), 2) − radius);
21     end
22
23     f_regularizer = 0;
24     for i=2:1:T
25         f_regularizer = f_regularizer + norm(u(:, i)−u(:, i−1), 2);
26     end
27
28     f = f_waypoints + lambda1 * f_regularizer;
29     minimize( f );
30
31     %subject to
32     x(:,1) == initialx;
33     x(:,T+1) == finalx;
34
35     for t = 1:T
36         norm( u(:,t)) ≤ Umax;
37     end
38
39     for t = 1:T
40         x(:, t+1) == A * x(:, t) + B * u(:, t);
41     end
42
43 cvx_end;
44
45 %plot the results
46 plot_graphs_disks(x, u, tau+1, c, radius);
47
48
49 %changes in control signal
50 counter = 0;
51 for t=2:1:T
52     if norm(u(:,t)−u(:,t−1), 2) > power(10,−6)
53         counter = counter + 1;
54     end
55 end
56
57 %calculation of mean deviation
```

```
58  m = 0;
59  m_ = 0;
60  for i=1:1:k
61      m = m + norm((E * x(:, tau(i))) - c(:, i));
62      m_ = m_ + pos(norm((E * x(:, tau(i))) - c(:, i) ) - radius);
63  end
64  mean_deviation = m/k;
65  mean_desviation_disk = m_/k;
66
67  disp(counter);
68  disp(mean_deviation);
69  disp(mean_desviation_disk);
```

Listing 4: Script Task 6

Task 7

When trying to solve problem (10) in CVX the software is unable to provide a valid solution and defines that the optimimzation problem is infeasible, for the given parameters. The motor of the robot is limited by U_{max}, which is lower than the previous tasks, and it turns out it is too weak for the robot to be able to pass exactly in all waypoints in the required time instants.

```
1   clear all;
2   close all;
3
4   %load the workspace
5   load('dataC.mat');
6
7   % solve optimization problem
8   cvx_begin
9       variable x(4, T+1);% columns are R^4 state vectors
10      variable u(2,   T); % columns are R^2 control signal
11
12      f = 0;
13      minimize( f );
14
15      %subject to
16      x(:,1) == initialx;
17      x(:,T+1) == finalx;
18
19      for t = 1:T
20          norm( u(:,t)) <= Umax;
21      end
22
23      for i=1:1:k
24          E * x(:, tau(i) + 1) == w(:, i);
25      end
```

```
26
27       for t = 1:T
28            x(:, t+1) == A * x(:, t) + B * u(:, t);
29       end
30
31   cvx_end;
32
33
34   captured=0;
35   for i=1:1:k
36       dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);
37
38       %disp(dw);
39       if dw ≤ 10^-6
40            captured = captured + 1;
41       end
42   end
43
44   disp(captured);
45
46
47   plot_graphs(x, u, tau+1, w);
```

Listing 5: Script Task 7

Task 8

Function ϕ is proved nonconvex both graphically and by definition.

$$\phi(x_1, x_2) = \begin{cases} 0, & \text{if } (x_1, x_2) = 0 \\ 1, & \text{if } (x_1, x_2) \neq 0 \end{cases}$$

In figure 10, the function and one of its chords are plotted, with $x_2 = 0$. Part of the function graph is above that chord, for example for the $x = (-1, 0)$ marked in green, so we can conclude the function is nonconvex.
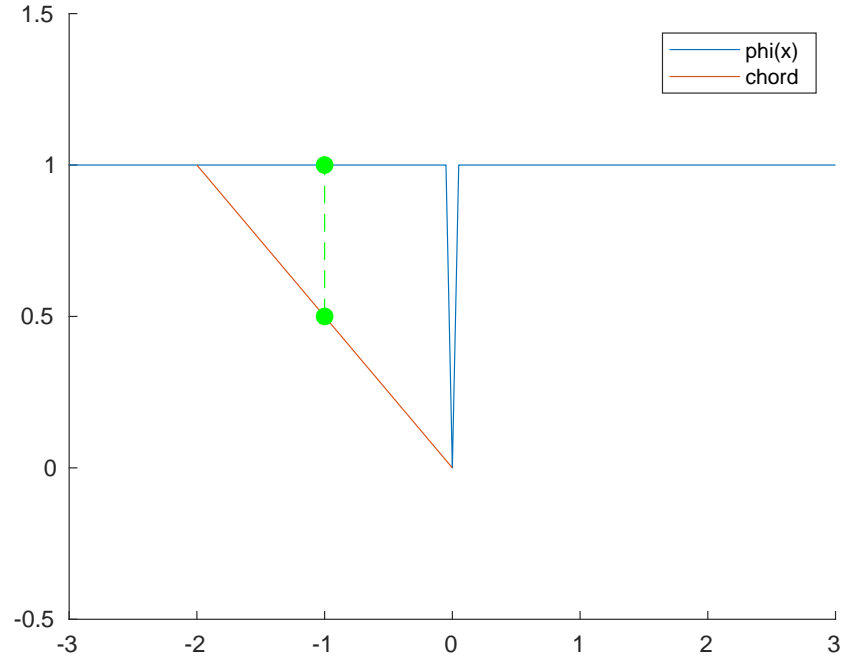
Figure 10: Graphical proof hat ϕ is nonconvex

By definition, function ϕ has to satisfy, for all $x_a, x_b \in \mathbb{R}$ and $0 \leq \alpha \leq 1$:

$$\phi((1-\alpha)x_a + \alpha x_b) \leq (1-\alpha)\phi(x_a) + \alpha\phi(x_b)$$

Using the same example, this inequality does not hold for $x_a = (-2,0)$, $x_b = (0,0)$ and $\alpha = 0.5$:

$$\phi(-1,0) \not\leq 0.5\phi(-2,0) + 0.5\phi(0,0)$$
$$1 \not\leq 0.5$$

So we can conclude the function ϕ is nonconvex.

```matlab
1   clear all;
2   close all;
3
4   x = -3:0.05:3;
5   phi = x;
6
7   for i=1:1:length(x)
8       if phi(i) ≠ 0
9           phi(i) = 1;
10      end
11  end
12
13  x_hat = -2:0.05:0;
```

```
14  chord = −0.5 * x_hat;
15
16  figure(1);
17  hold on;
18
19  f = plot(x, phi);
20  g = plot(x_hat, chord);
21  line([−1 −1], [0.5 1], 'Color', 'green', 'LineStyle', '−−');
22  scatter([−1 −1], [0.5 1], 'MarkerEdgeColor', 'green', 'MarkerFaceColor', '
        ↪ green', 'LineWidth', 1.5);
23
24  axis([−3 3 −0.5 1.5]);
25  xlabel('x');
26  legend([f g], 'phi(x)', 'chord');
27
28  saveas(figure(1), 'Figures/task8/nonconvex.pdf');
```

Listing 6: Script Task 8

Task 9

a)

Figure 11: Positions of the robot from $t = 0$ to $t = T$ (blue dots) for the new formulation with the ℓ_2^2 formulation. Waypoints ω_k are marked as red square and positions at the appointed times τ_k are marked with red circles.

b)

Figure 12: Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$, for the formulation with the ℓ_2^2 norm.

c)

The robot did not capture any waypoint.

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('dataC.mat');
6
7  % solve optimization problem
8  cvx_begin quiet
9      variable x(4, T+1);% columns are R^4 state vectors
10     variable u(2,   T); % columns are R^2 control signal
11
12     %cost function
13     f = 0;
14     for i=1:1:k
15         f = f + square_pos( norm(E * x(:, tau(i) + 1) - w(:, i), 2) );
16     end
17
18     minimize( f );
```

30

```matlab
19
20      %subject to
21      x(:,1) == initialx;
22      x(:,T+1) == finalx;
23
24      for t = 1:T
25          norm( u(:,t)) <= Umax;
26      end
27
28      for t = 1:T
29          x(:, t+1) == A * x(:, t) + B * u(:, t);
30      end
31
32  cvx_end;
33
34  % plot postions and control signals
35  plot_graphs(x, u, tau+1, w);
36
37  % save plots
38  saveas(figure(1), 'Figures/task9/position.pdf');
39  saveas(figure(2), 'Figures/task9/control.pdf');
40
41  captured=0;
42  for i=1:1:k
43      dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);
44
45      %disp(dw);
46      if dw <= 10^-6
47          captured = captured + 1;
48      end
49  end
50
51  str1 = ['Number of waypoints capture by the robot = ', num2str(captured)];
52  disp(str1);
53
54  close all;
```

Listing 7: Script Task 9
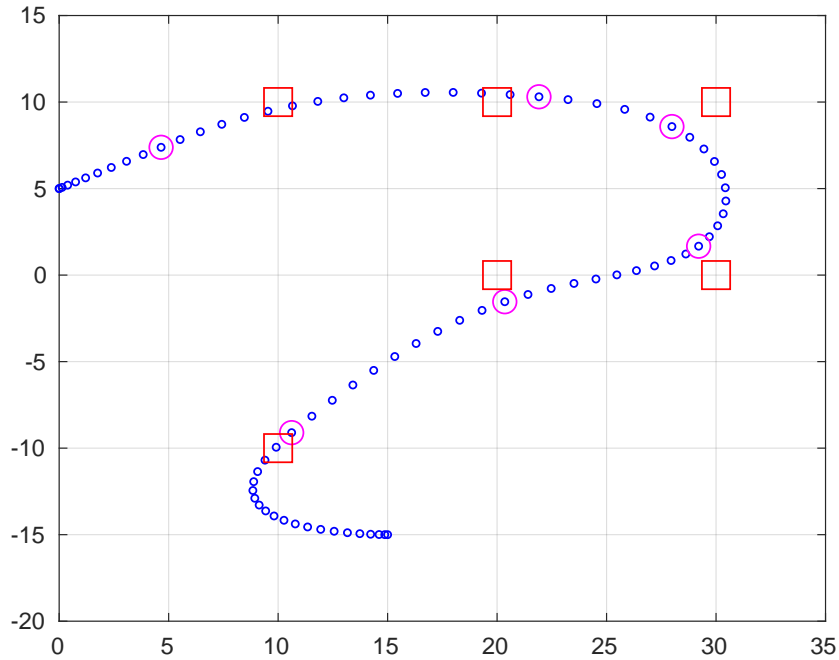
Task 10

a)
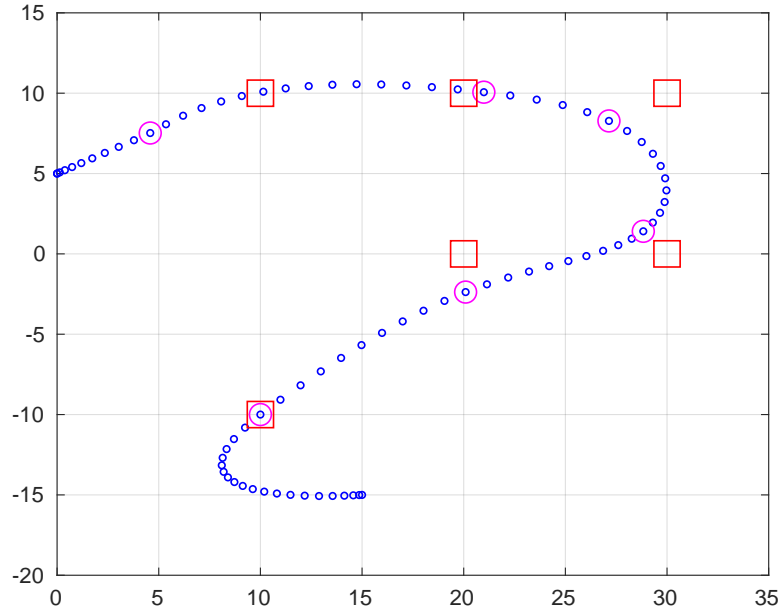
Figure 13: Positions of the robot from $t = 0$ to $t = T$ (blue dots) for the new formulation with the ℓ_2 formulation. Waypoints ω_k are marked as red square and positions at the appointed times τ_k are marked with red circles.
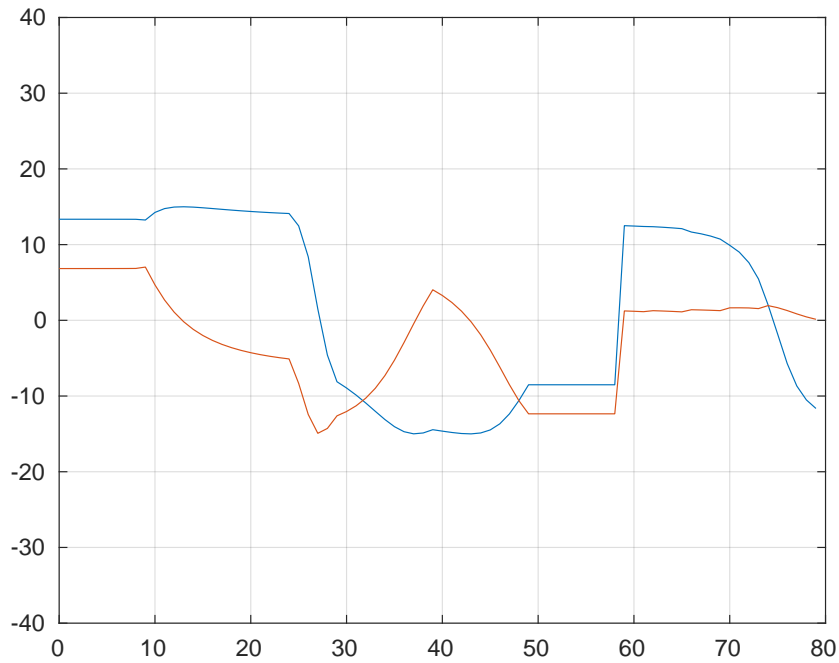
b)
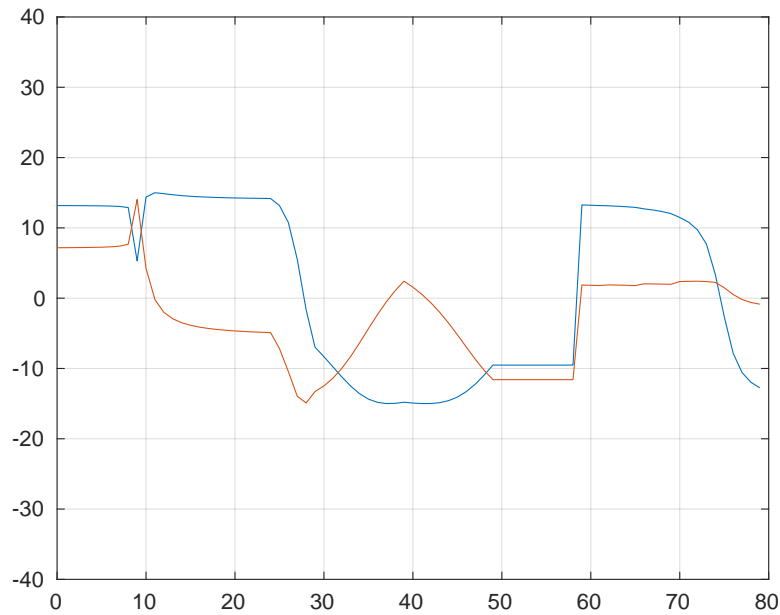
Figure 14: Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$, for the formulation with the ℓ_2 norm.

c)

The robot captured 1 waypoint.

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('dataC.mat');
6
7  % solve optimization problem
8  cvx_begin quiet
9      variable x(4, T+1);% columns are R^4 state vectors
10     variable u(2,   T); % columns are R^2 control signal
11
12     %cost function
13     f = 0;
14     for i=1:1:k
15         f = f + norm(E * x(:, tau(i) + 1) - w(:, i), 2);
16     end
17
18     minimize( f );
19
20     %subject to
21     x(:,1) == initialx;
22     x(:,T+1) == finalx;
23
24     for t = 1:T
25         norm( u(:,t)) <= Umax;
26     end
27
28     for t = 1:T
29         x(:, t+1) == A * x(:, t) + B * u(:, t);
30     end
31
32 cvx_end;
33
34 % plot postions and control signals
35 plot_graphs(x, u, tau+1, w);
36
37 % save plots
38 saveas(figure(1), 'Figures/task10/position.pdf');
39 saveas(figure(2), 'Figures/task10/control.pdf');
40
41 captured=0;
42 for i=1:1:k
43     dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);
44
45     %disp(dw);
46     if dw <= 10^-6
```

```
47            captured = captured + 1;
48        end
49  end
50
51  str1 = ['Number of waypoints capture by the robot = ', num2str(captured)];
52  disp(str1);
53
54  close all;
```

Listing 8: Script Task 10

Task 11

a)
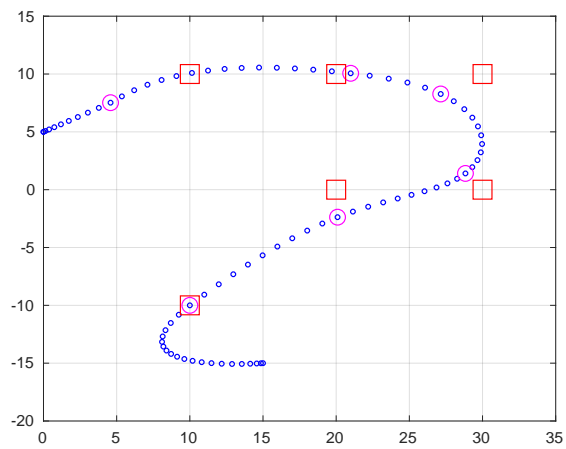
(a) $m = 0$
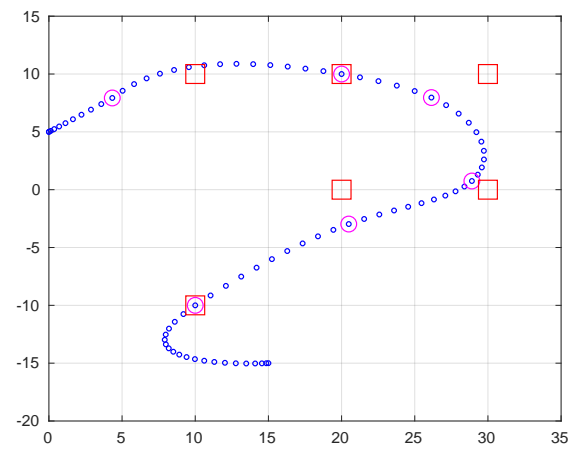
(b) $m = 1$
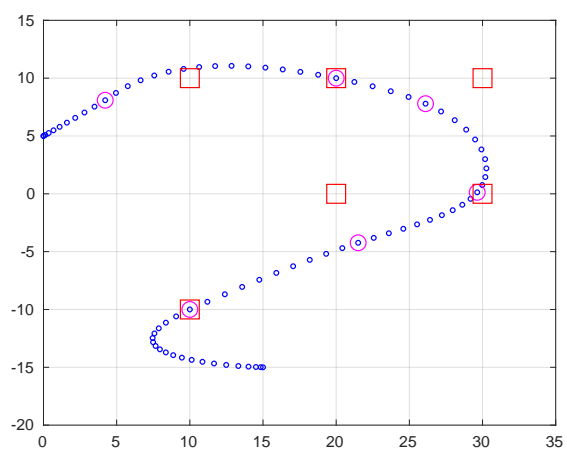
(c) $m = 2$
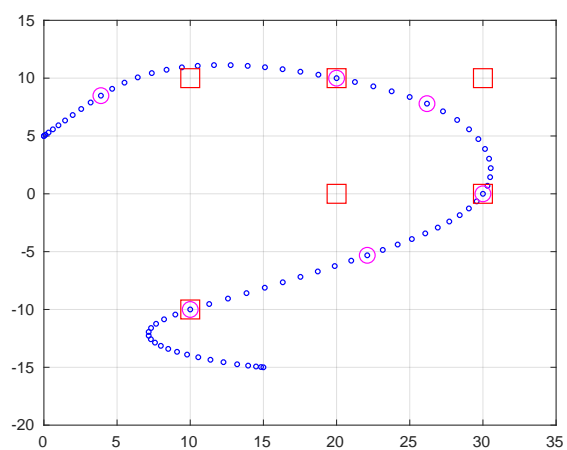
(d) $m = 3$
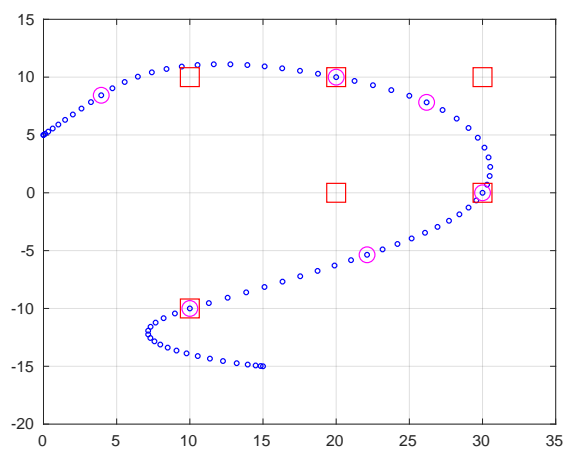
(e) $m = 4$
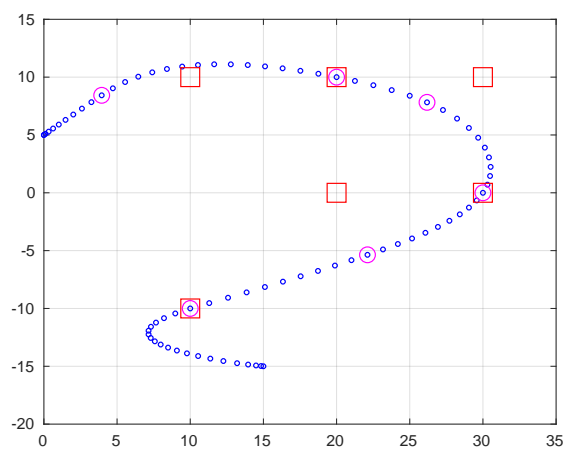
(f) $m = 5$
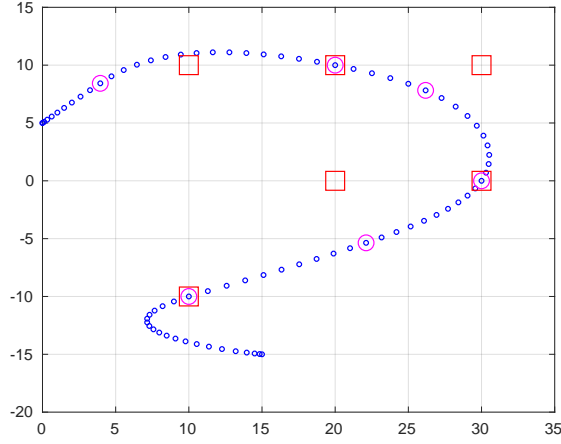
(g) $m = 6$
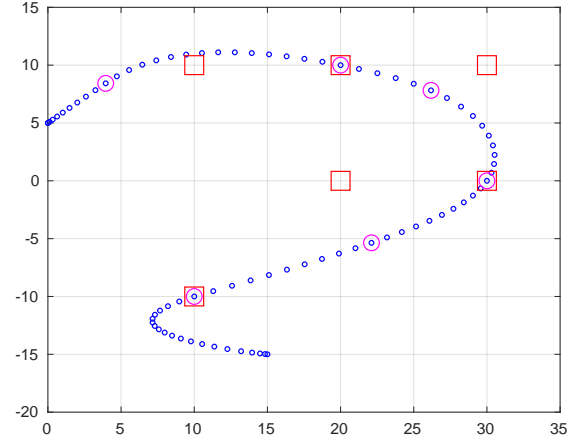
(h) $m = 7$
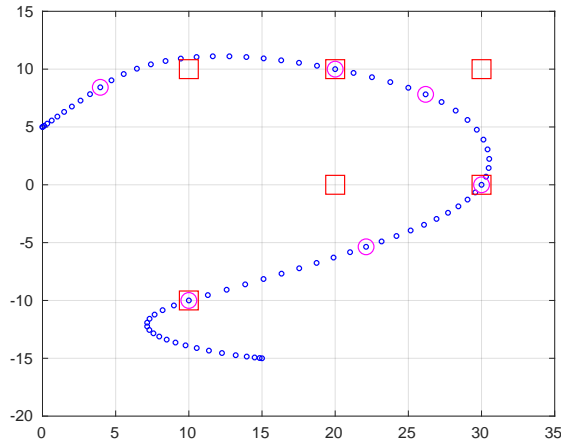
(i) $m = 8$
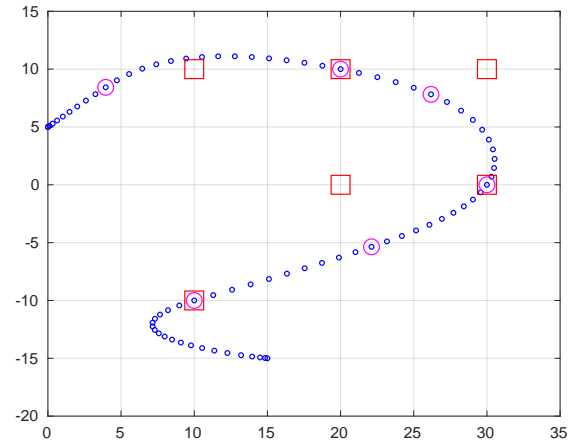
(j) $m = 9$

Figure 15: Positions of the robot $x^{(m)}(t)$ from $t = 0$ to $t = T$ (blue dots) for each iteration m. Waypoints ω_k are marked as red square and positions at the appointed times τ_k are marked with red circles.

b)

(a) $m = 0$

(b) $m = 1$

(c) $m = 2$

(d) $m = 3$

37

(e) $m = 4$

(f) $m = 5$

(g) $m = 6$

(h) $m = 7$

38
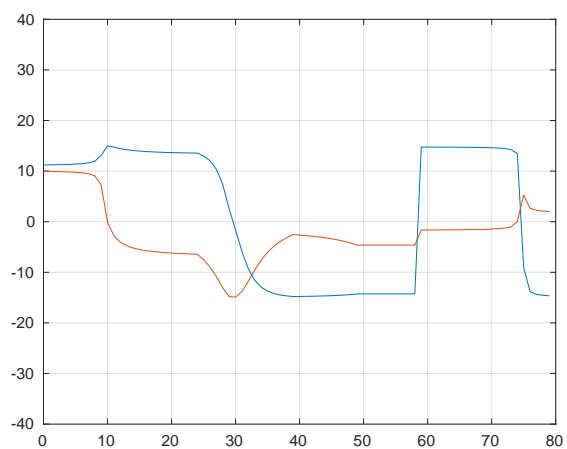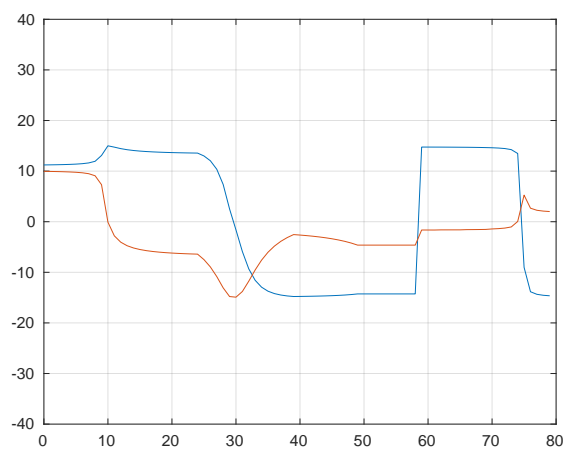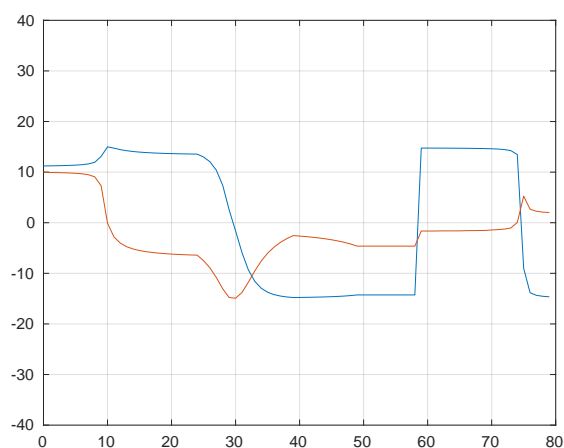
(i) $m = 8$
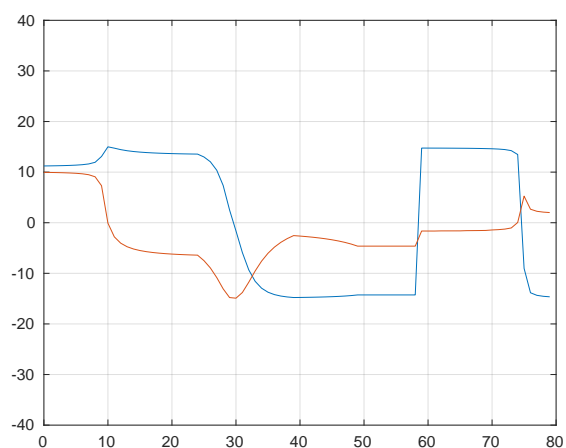
(j) $m = 9$

Figure 16: Optimal control signal $u^{(m)}(t)$ from $= 0$ to $= T$ for each iteration m.

c)

m	Number of captured waypoints
0	1
1	2
2	2
3	3
4	3
5	3
6	3
7	3
8	3
9	3

Table 7: Number of waypoints captured by the robot.

```
1  clear all;
2  close all;
3
4  %load the workspace
5  load('dataC.mat');
6
```

```matlab
7  weights = ones(k, 1);
8  epson = 10^-6;
9  iter = 10;
10
11 for m=1:1:iter
12     % solve iterative optimization problem
13     cvx_begin quiet
14         variable x(4, T+1);% columns are R^4 state vectors
15         variable u(2,   T); % columns are R^2 control signal
16
17         %cost function
18         f = 0;
19         for i=1:1:k
20             f = f + weights(i) * norm(E * x(:, tau(i) + 1) - w(:, i), 2);
21         end
22
23         minimize( f );
24
25         %subject to
26         x(:,1) == initialx;
27         x(:,T+1) == finalx;
28
29         for t = 1:T
30             norm( u(:,t)) <= Umax;
31         end
32
33         for t = 1:T
34             x(:, t+1) == A * x(:, t) + B * u(:, t);
35         end
36     cvx_end;
37
38     % plot postions and control signals
39     plot_graphs(x, u, tau+1, w);
40
41     % save plots
42     saveas(figure(1), strcat('Figures/task11/iter_', num2str(m - 1), '
           ↪ _position.pdf'));
43     saveas(figure(2), strcat('Figures/task11/iter_', num2str(m - 1), '
           ↪ _control.pdf'));
44
45     % calc new weights
46     for i=1:1:k
47         weights(i) = 1 / (norm(E * x(:, tau(i)+1) - w(:, i), 2) + epson);
48     end
49
50     %disp(weights);
51
52     captured=0;
53     for i=1:1:k
54         dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);
55
```

```
56          if dw ≤ 10^−6
57              captured = captured + 1;
58          end
59      end
60
61
62      str1 = ['Number of waypoints capture on iteration ', num2str(m − 1), '
            ↪ = ', num2str(captured)];
63      disp(str1);
64
65      close all;
66  end
```

Listing 9: Script Task 11

Task 12

The weights introduced in the last formulation change the importance that the optimizer gives to minimizing the distance to each of the different waypoints, as a bigger weight translates to a more significant term in the minimization.

In the first iteration the weights are all set to one (similar to Task 10), so that all waypoints have the same importance. As the goal is to set some of these distances to zero, between each iteration, all points that are already close to the objective will have their weight increased, in comparison to the other weights, and the points that are further are penalized. This can be seen from the weight expression:

$$weight_k^{(m)} = \frac{1}{\|Ex^{(m)}(\tau_k) - \omega_k\|_2 + \epsilon} \tag{4}$$

Over the successive iterations, the waypoints that are easier to capture start getting bigger and bigger weights and the trajectory is fixed to pass through them. In the next iteration the optimizer tries to capture more waypoints, but with the constraint of passing through the already captured points. Waypoints that are far from the trajectory get smaller and smaller weights, and start to be ignored by the optimizer.

This successive capturing of new waypoints can be seen by analyzing the trajectories by iteration, in Figure 15. First the last waypoint is fixed ($m = 0$), then the second ($m = 1$) and finally the fourth ($m = 3$). The other waypoints cannot be captured with these 3 already fixed.

As expected removing the term for simple control from the cost function, the control signal has more sudden changes. The trajectory of the robot is only limited by the strength of the motor and the laws of movement.

Comparing Task 8 and Task 9, with the ℓ_2 norm formulation one waypoint is captured. The optimizer is minimizing the sum of distances to the waypoints, and replacing the ℓ_2^2 with the ℓ_2 allows some of the those components to be exactly minimized to zero (or very

very close), as is discussed in Task 4. This is exactly what is wanted in this problem, for as many of those distances to be zero as possible, so it makes sense that norm ℓ_2 is used for the iterative formulation.

The robot trajectory (and number of captured waypoints) converges for a very early iteration, as it almost does not change from iteration $m = 4$ on-wards. Could an automatic stop criterion be implemented?

Part 2

1 Logistic Regression

To simplify the computation of the function to minimize and its gradient and hessian, a variable change is performed. The variable t is introduced:

$$t = \begin{bmatrix} s \\ r \end{bmatrix} \in \mathbb{R}^3$$

The model expression can be evaluated for a point, by a vector multiplication:

$$s^T x_k - r = \hat{x}_k^T t$$

By introducing:

$$\hat{x}_k = \begin{bmatrix} x_k \\ -1 \end{bmatrix} \qquad \hat{X} = \begin{bmatrix} x_1 & x_2 & \dots & x_K \\ -1 & -1 & \dots & -1 \end{bmatrix} = \begin{bmatrix} X \\ -\mathbf{1} \end{bmatrix}$$

The optimization problem becomes:

$$\underset{t \in \mathbb{R}}{\text{minimize}} \quad \hat{f}(t) = \frac{1}{K} \sum_{k=1}^{K} (\log(1 + \exp(\hat{x}_k^T t)) - y_k(\hat{x}_k^T t))$$

The function to minimize, can be expressed as:

$$f(t) = \frac{1}{K} \sum_{k=1}^{K} \phi(\hat{x}_k^T t)$$

Where, $\phi(u) : \mathbb{R} \to \mathbb{R}$:

$$\phi(u) = \log(1 + \exp(u)) - y_k u$$

$$\dot{\phi}(u) = \frac{\exp(u)}{1 + \exp(u)} - y_k$$

$$\ddot{\phi}(u) = \frac{\exp(u)}{(1 + \exp(u))^2}$$

Using the results of task5, the gradient and hessian can be expressed as:

$$\nabla f(t) = \frac{1}{K} \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_K \end{bmatrix} \begin{bmatrix} \dot{\phi}(\hat{x}_1^T t) \\ \dot{\phi}(\hat{x}_2^T t) \\ \dots \\ \dot{\phi}(\hat{x}_K^T t) \end{bmatrix} = \frac{1}{K} \hat{X} v$$

$$\mathbf{H}f(t) = \frac{1}{K} \begin{bmatrix} \hat{x}_1 & \hat{x}_2 & \dots & \hat{x}_K \end{bmatrix} \begin{bmatrix} \ddot{\phi}(\hat{x}_1^T t) & & & \\ & \ddot{\phi}(\hat{x}_2^T t) & & \\ & & \dots & \\ & & & \ddot{\phi}(\hat{x}_K^T t) \end{bmatrix} \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \\ \dots \\ \hat{x}_K \end{bmatrix} = \frac{1}{K} \hat{X} D \hat{X}^T$$

These expressions can be evaluated very efficiently in MATLAB, using matrix multiplication.

Task 1

The convexity of function $f(t)$ can be proved by dividing it into smaller functions known to be convex, composed by operations that preserve convexity.

The cost function,

$$f(t) = \frac{1}{K} \sum_{k=1}^{K} (\log(1 + \exp(\hat{x}_k^T t)) - y_k(\hat{x}_k^T t)) \, ,$$

is the sum of functions

$$h_k(t) = \log(1 + \exp(\hat{x}_k^T t)) - y_k(\hat{x}_k^T t)$$

multiplied by a positive constant. If the functions $h_k(t)$ are convex, the result is also convex.

Function $h_k(t)$ can be seen as the composition $\phi(g_k(t))$ of function

$$\phi(u) = \log(1 + \exp(u)) - y_k u$$
$$g_k(t) = \hat{x}_k^T \, .$$

Function $\phi(u)$ is the sum of the logistic function $\phi_1(u) = \log(1 + \exp(u))$ and an affine function $\phi_2(u) = y_k u$, which are both convex, as studied in the lectures, so it is convex. Function $h_k(t)$ is an affine map, and the composition of a convex function and an affine map was studied to be convex.

Then, the convexity of function $f(t)$ is proven, by recursion, as shown in Figure 17.
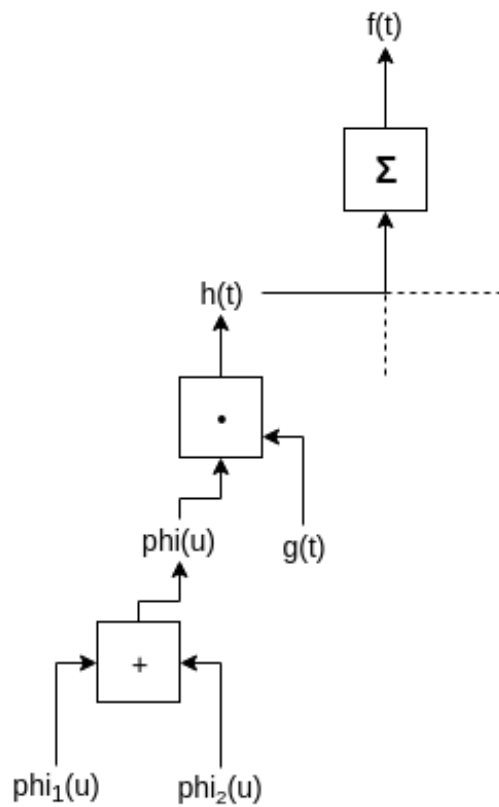
Figure 17: Recursion tree proving function's $f(t)$ convexity.

Task 2

The following matlab functions were used in order to calculate the wanted function, it's gradient and it's hessian respectively:

```matlab
1 function [ result ] = f_hat( t, X_hat, Y, k )
2 result = (1/k) * sum(log(1 + exp(X_hat'*t)) - (Y.').*(X_hat'*t));
3 end
```

Listing 10: Script to calculate function value

```matlab
1 function [ result ] = gradient_f_hat( t, X_hat, Y, k )
2
3 result = 1/k * X_hat * (exp(X_hat'*t)./(1 + exp(X_hat'*t)) - Y');
4
5 end
```

Listing 11: Script to calculate gradient of function

```matlab
1 function [ result ] = hessian_f_hat( t, X_hat, Y, k )
2
3 result = 1/k * X_hat * diag(exp(X_hat'*t)./((1 + exp(X_hat'*t)).^2)) *
     ↪ X_hat';
4
5 end
```

Listing 12: Script to calculate hessian of function
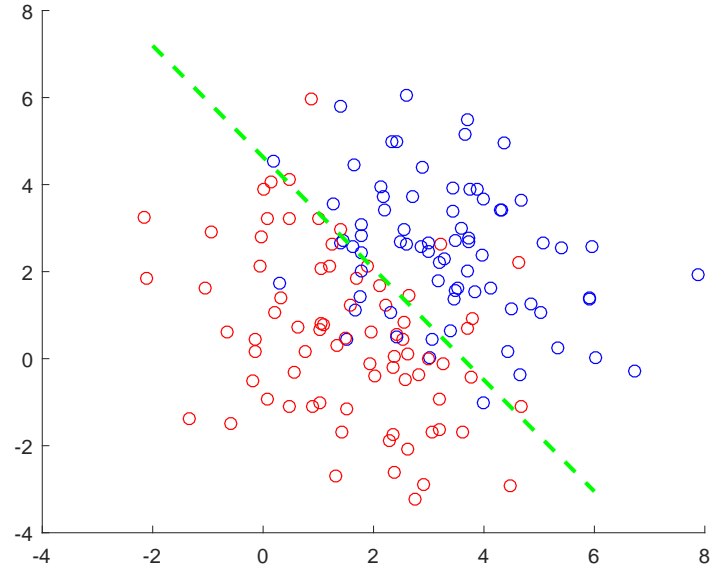
Figure 18: Gradient method result for data1.m.

$$s = (1.3495, 1.0540), \quad r = 4.8815$$

The total number of iterations was: 1125.
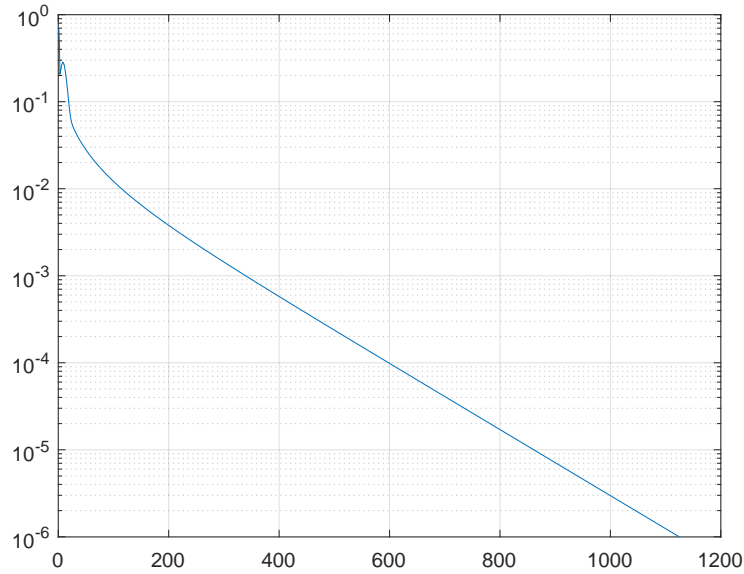
Figure 19: Norm of the gradient along iterations of the gradient method for data1.m.

Task 3
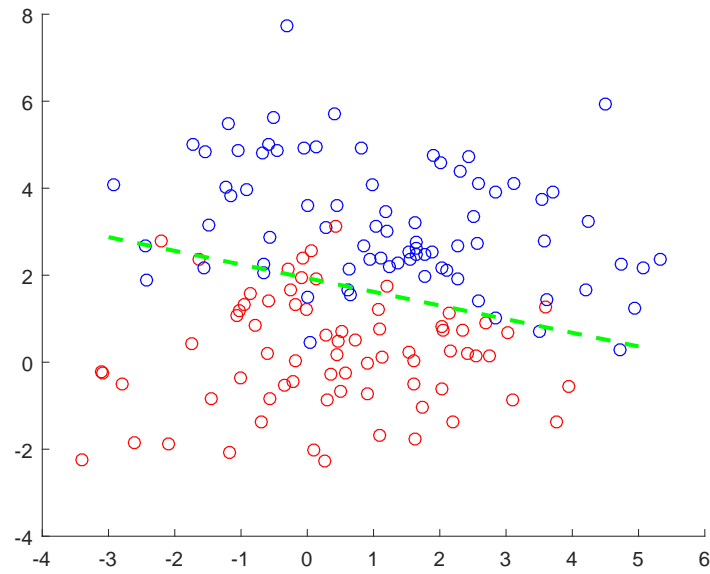
Figure 20: Gradient method result for data2.m.

$$s = (0.7402, 2.3577), \quad r = 4.5553$$

The total number of iterations was: 1362.
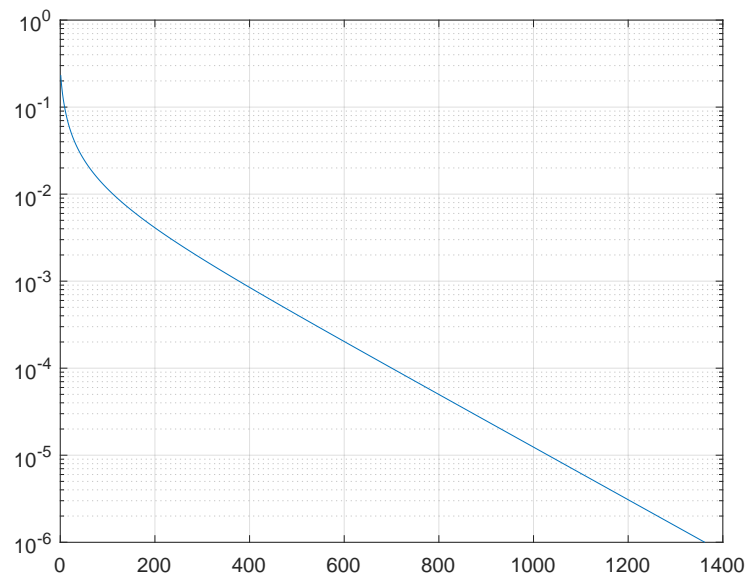
Figure 21: Norm of the gradient along iterations of the gradient method for data2.m.

The following code was used for calculating the solutions for both task2 and task3 (in line 5 we specify which dataset we are using):

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('data1.mat');
6
7  %%%%%%%%%%%%%%%%
8  %Gradient method
9  %%%%%%%%%%%%%%%%
10
11 %Amount of input features
12 k = 150;
13
14 %Stopping criterion constants
15 s0 = [-1 -1];
16 r0 = 0;
17 epslon = 10^(-6);
18
19 %Initial point for gradient descent
20 t0 = [s0 r0]'
21
22 %Backtracking parameters
23 alpha0 = 1;
24 y = 10^(-4);
25 beta = 0.5;
26
27 %Transformation of X
28 X_hat = [X; -ones(length(X), 1).'];
29
30 %Algorithm - Gradient Descent
31 t = t0
32 alpha = alpha0;
33 gradients = [];
34 while norm(gradient_f_hat(t, X_hat, Y, k)) >= epslon
35     d = -gradient_f_hat(t, X_hat, Y, k);
36     alpha = alpha0;
37     while f_hat(t + alpha.*d, X_hat, Y, k) >= f_hat(t, X_hat, Y, k) + (y.*
            ↪ gradient_f_hat(t, X_hat, Y, k)'*(alpha.*d))
38         alpha = beta .* alpha;
39     end
40     t = t + (alpha .* d)
41     %f_hat(t, X_hat, Y, k)
42     gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
43 end
44
45 %plot figure with logarithmic y-axis
46 figure;
```

```matlab
47  semilogy(gradients);
48  grid on;
49
50  iter = length(gradients)
51
52  s0 = t(1)
53  s1 = t(2)
54  r = t(3)
55
56
57  figure;
58  %Plot data points
59  for i = 1:150
60      if Y(i) == 0
61          scatter(X(1, i), X(2, i), [], 'red');
62          hold on
63      else
64          scatter(X(1, i), X(2, i), [], 'blue');
65          hold on
66      end
67  end
68
69  %Plot resulting line
70  xResult = -3:0.001:5;
71  yResult = (r/s1) - (s0/s1)*xResult;
72  plot(xResult, yResult,['--', 'g'], 'LineWidth', 2)
```

Listing 13: Script used to calculate solutions for task2 and task3

Task 4

data3.m
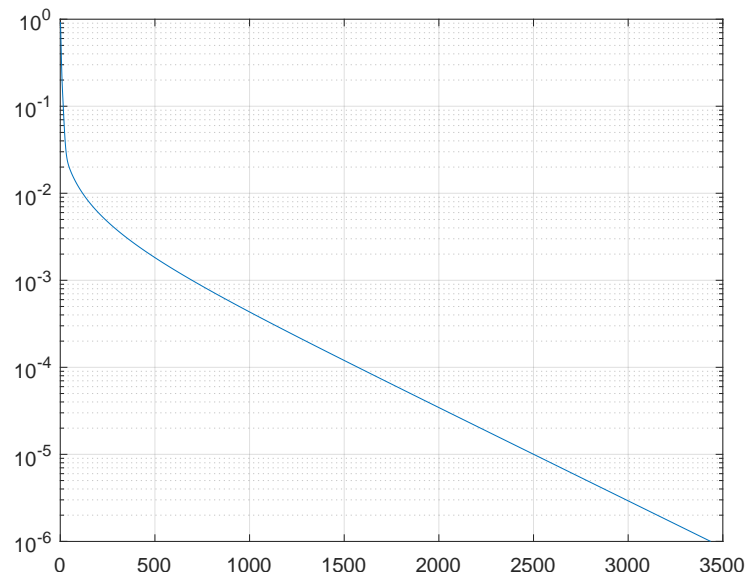
Figure 22: Norm of the gradient for the gradient method along iterations for data3.m.

$$
s = \big(-1.308227, \quad 1.407825, \quad 0.804854, \quad -1.002446, \quad 0.554806,
$$
$$
-0.548903, \quad -1.199742, \quad 0.079202, \quad -1.827920, \quad -0.148425,
$$
$$
1.924119, \quad -0.358574, \quad -0.289965, \quad 0.192455, \quad 1.061395,
$$
$$
0.210723, \quad -0.092877, \quad 1.0476061, \quad -1.124826, \quad -1.331059,
$$
$$
0.766117, \quad -0.272855, \quad -0.534893, \quad 0.999579, \quad -0.419137,
$$
$$
-0.313312, \quad 0.407509, \quad -0.196521, \quad -0.737937, \quad -0.981437 \big)
$$

$$
r = -4.7984
$$

The total number of iterations was: 3436.

dataset4.m
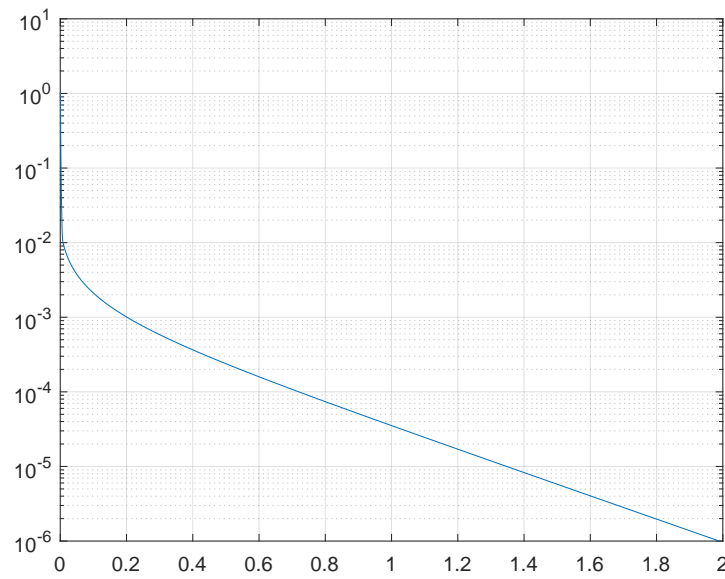
Figure 23: Norm of the gradient for the gradient method along iterations for dataset4.m. The horizontal axis is multiplied by 10^4.

$$s = \big(0.1098, \quad -0.6423, \quad 0.1019, \quad 1.2428, \quad -1.6431, \qquad\qquad ,$$
$$1.0244, \quad 0.0512, \quad 0.8271, \quad 0.3136, \quad 0.7449,$$
$$-0.5858, \quad 0.6267, \quad 1.3611, \quad 0.1534, \quad 2.3234,$$
$$-0.0840, \quad -0.9489, \quad 2.4699, \quad -0.8678, \quad -1.6516,$$
$$0.6460, \quad -0.4779, \quad 1.6397, \quad 0.9034, \quad -1.2293,$$
$$-0.7587, \quad -0.4887, \quad 1.0306, \quad 0.0888, \quad -1.0917,$$
$$-1.2717, \quad -2.0333, \quad -0.2505, \quad -0.3518, \quad -0.3486,$$
$$-2.5610, \quad -0.3132, \quad -0.4902, \quad 0.7258, \quad 0.5774,$$
$$-1.0528, \quad 0.6400, \quad 0.3759, \quad -0.1547, \quad 0.0298,$$
$$0.9547, \quad -0.2863, \quad 0.6364, \quad 0.7859, \quad 0.7584,$$
$$0.2880, \quad 0.1648, \quad 0.6776, \quad 2.0550, \quad 1.0996,$$
$$0.5261, \quad -0.5770, \quad 1.1454, \quad -0.5617, \quad 0.0065,$$
$$0.4768, \quad -2.3677, \quad -1.1561, \quad -2.6619, \quad 0.0622,$$
$$0.1037, \quad -0.6237, \quad 0.1913, \quad 0.6672, \quad -1.0493,$$
$$-0.3240, \quad -0.3207, \quad -1.0904, \quad -0.8293, \quad -0.3104,$$
$$-0.4879, \quad -0.1060, \quad -0.1646, \quad 2.2683, \quad -1.2380,$$
$$-0.8575, \quad -2.4781, \quad -0.4158, \quad 0.1660, \quad 0.7931,$$
$$0.3685, \quad -0.0524, \quad -0.9997, \quad -0.5732, \quad 0.3971,$$
$$1.1911, \quad 1.8318, \quad -1.7287, \quad 0.2329, \quad -1.1921,$$
$$1.6558, \quad 0.4612, \quad -0.6431, \quad 0.8295, \quad 0.2975\big)$$
$$r = 7.6701\,,$$

The total number of iterations was: 19892.

The following code was used for calculating the solutions for both datasets in task 4 (in line 5 we specify which dataset we are using):

```
1  clear all;
2  close all;
3
4  %load the workspace
5  load('data3.mat');
6
7  %%%%%%%%%%%%%%%
8  %Gradient method
9  %%%%%%%%%%%%%%%%
10
11 %Amount of input features
12 [n, k] = size(X);
13
14 %Stopping criterion constants
15 s0 = -ones(1, n);
16 r0 = 0;
17 epslon = 10^(-6);
```

```
18
19  %Initial point for gradient descent
20  t0 = [s0 r0]'
21
22  %Backtracking parameters
23  alpha0 = 1;
24  y = 10^(−4);
25  beta = 0.5;
26
27  %Transformation of X
28  X_hat = [X; −ones(length(X), 1)'];
29
30  %Algorithm − Gradient Descent
31  t = t0;
32  alpha = alpha0;
33  gradients = [];
34  while norm(gradient_f_hat(t, X_hat, Y, k)) ≥ epslon
35      d = −gradient_f_hat(t, X_hat, Y, k);
36      alpha = alpha0;
37      while f_hat(t + alpha.*d, X_hat, Y, k) ≥ f_hat(t, X_hat, Y, k) + (y.*
          ↪ gradient_f_hat(t, X_hat, Y, k)'*(alpha.*d))
38          alpha = beta .* alpha;
39      end
40      t = t + (alpha .* d);
41      %f_hat(t, X_hat, Y, k)
42      gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
43  end
44
45  %plot figure with logarithmic y−axis
46  figure;
47  semilogy(gradients);
48  grid on;
49
50  iter = length(gradients)
51
52  s = t(1:n)
53  r = t(n+1)
```

Listing 14: Script used to calculate solutions for task4

Task 5

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

a)

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

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

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

That can be expressed as matrix multiplication:

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

b)

$$\mathbf{H}p(x) = \begin{bmatrix} \frac{\partial^2 p(x)}{\partial x_1^2} & \frac{\partial^2 p(x)}{\partial x_1 \partial x_2} & \frac{\partial^2 p(x)}{\partial x_1 \partial x_3} \\ \frac{\partial^2 p(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 p(x)}{\partial x_2^2} & \frac{\partial^2 p(x)}{\partial x_2 \partial x_3} \\ \frac{\partial^2 p(x)}{\partial x_3 \partial x_1} & \frac{\partial^2 p(x)}{\partial x_3 \partial x_2} & \frac{\partial^2 p(x)}{\partial x_3^2} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{K} \frac{\partial^2}{\partial x_1^2} \phi(a_k^T x) & \dots & \dots \\ & \dots & \\ & \dots & \end{bmatrix}$$

From the chain rule, $\frac{\partial^2}{\partial x_i \partial x_j} \phi(a_k^T x) = \ddot{\phi}(a_k^T) a_{ki} a_{kj}$:

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

$$\mathbf{H}p(x) = \sum_{k=1}^{K} a_k^{\ddot{T}} x a_k a_k^T = \sum_{k=1}^{K} a_k a_k^{\ddot{T}} x a_k^T$$

Which can be expressed using matrix multiplication:

$$\mathbf{H}p(x) = \begin{bmatrix} a_1 & a_2 & \dots & a_K \end{bmatrix} \begin{bmatrix} \ddot{\phi}(a_1^T x) & & \mathbf{0} \\ & \dots & \\ \mathbf{0} & & \ddot{\phi}(a_K^T x) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_K \end{bmatrix} = ADA^T$$

Task 6

data1.m
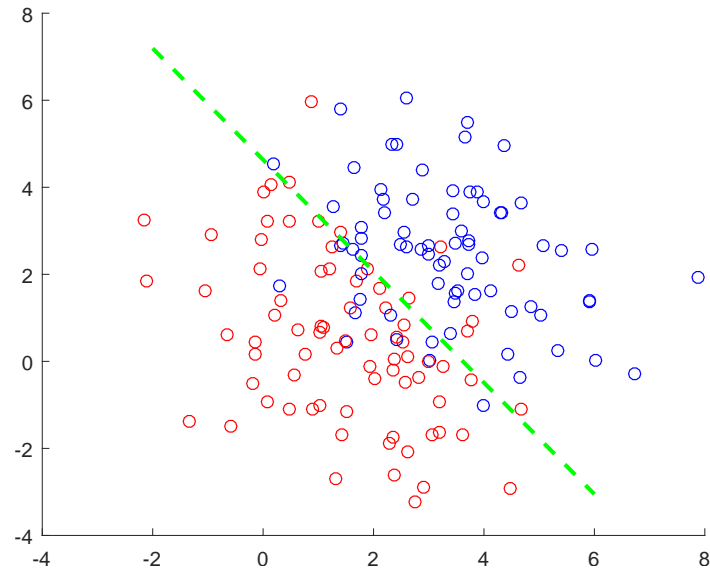
Figure 24: Newton method result for data1.m

$$s = (1.3496, 1.0540), \quad r = 4.8817$$

Total number of iterations: 8.
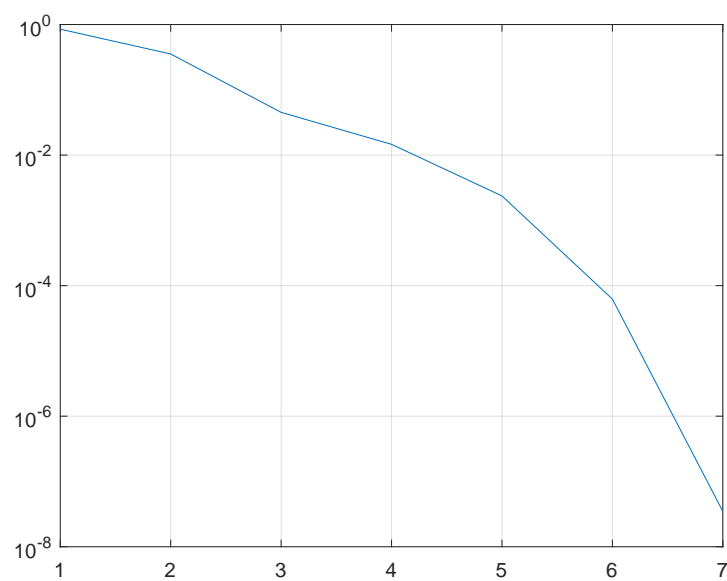
Figure 25: Norm of the gradient along iterations for data1.m
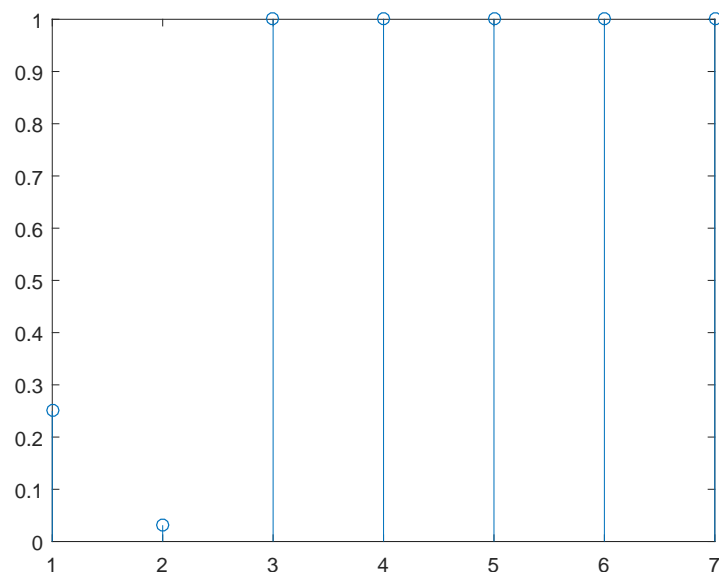
Figure 26: Value of alpha along iterations for data1.m

data2.m
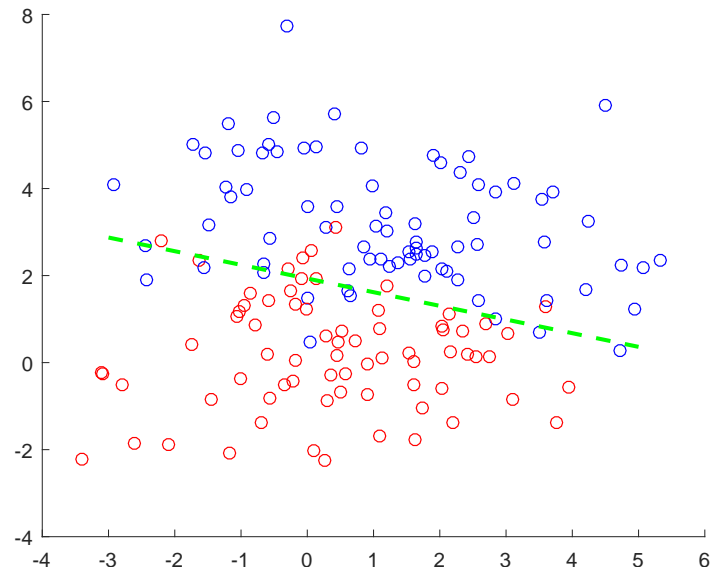
Figure 27: Newton method result for data2.m

$$s = (0.7402, 2.3577), \quad r = 4.5554$$

Total number of iterations: 8.
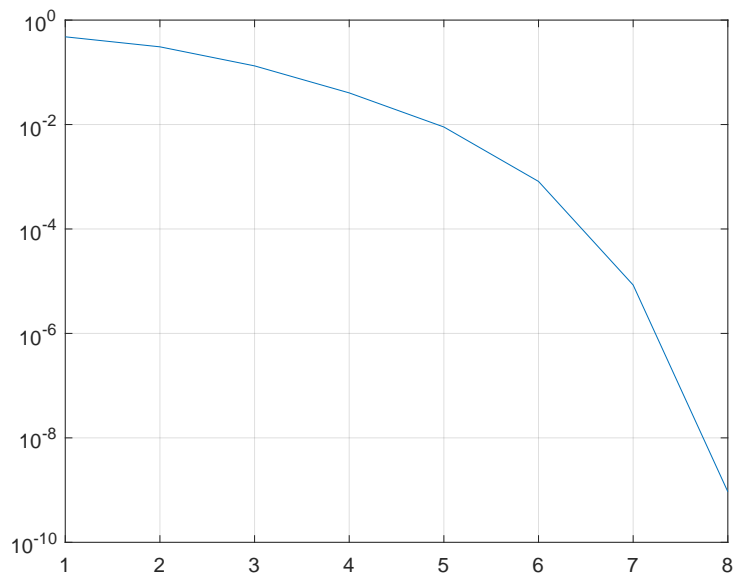
Figure 28: Norm of the gradient along iterations for data2.m.

Figure 29: Value of alpha along iterations for data2.m.

The following code was used for calculating the solutions for both datasets 1 and 2 (in line 5 we specify which dataset we are using):

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('data2.mat');
6
7  %%%%%%%%%%%%%%%%
8  %Gradient method
9  %%%%%%%%%%%%%%%%
10
11  %Amount of input features
12  k = 150;
13
14  %Stopping criterion constants
15  s0 = [-1 -1];
16  r0 = 0;
17  epslon = 10^(-6);
18
19  %Initial point for gradient descent
20  t0 = [s0 r0]'
21
22  %Backtracking parameters
23  alpha0 = 1;
24  y = 10^(-4);
25  beta = 0.5;
```

```matlab
26
27   %Transformation of X
28   X_hat = [X; −ones(length(X), 1).'];
29
30   %Algorithm − Newton Method
31   t = t0;
32   alpha = alpha0;
33   alphas = [];
34   gradients = [];
35   while norm(gradient_f_hat(t, X_hat, Y, k)) ≥ epslon
36       g = gradient_f_hat(t, X_hat, Y, k);
37       d = −(hessian_f_hat(t, X_hat, Y, k)\g);
38       alpha = alpha0;
39       while f_hat(t + alpha.*d, X_hat, Y, k) ≥ f_hat(t, X_hat, Y, k) + (y.*g
            ↪ '*(alpha.*d))
40           alpha = beta .* alpha;
41       end
42       t = t + (alpha .* d)
43       %f_hat(t, X_hat, Y, k)
44       gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
45       alphas = [alphas alpha];
46   end
47
48   %plot alpha for each iteration
49   figure;
50   stem(alphas);
51
52   %plot figure with logarithmic y−axis
53   figure;
54   semilogy(gradients);
55   grid on;
56
57   s0 = t(1)
58   s1 = t(2)
59   r = −t(3)
60
61   figure;
62   %Plot data points
63   for i = 1:150
64       if Y(i) == 0
65           scatter(X(1, i), X(2, i), [], 'red');
66           hold on
67       else
68           scatter(X(1, i), X(2, i), [], 'blue');
69           hold on
70       end
71   end
72
73   %Plot resulting line
74   xResult = −3:0.001:5;
75   yResult = (r/s1) − (s0/s1)*xResult;
```

```
76  plot(xResult, yResult,['--', 'g'], 'LineWidth', 2)
```

Listing 15: Script used to calculate solutions for datasets data1.m and data2.m

data3.m
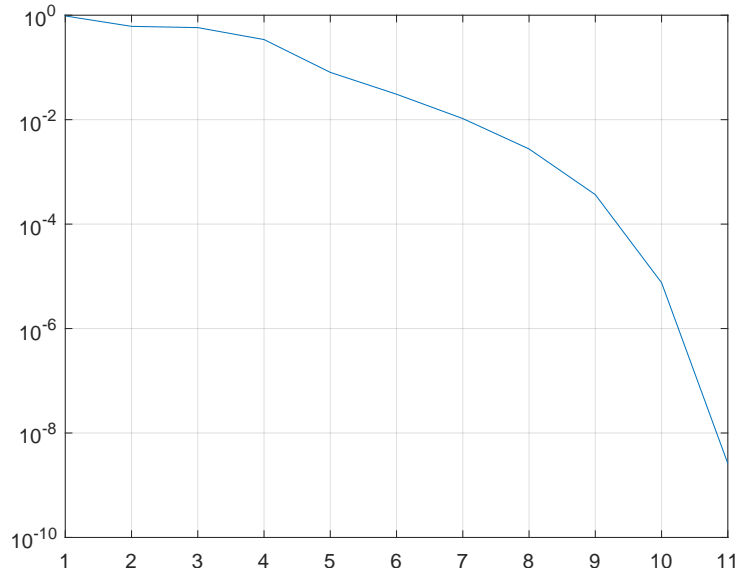
Figure 30: Norm of the gradient along iterations for data3.m.

$$
\begin{aligned}
s = \big(& 1.308302, & 1.407906, & 0.804898, & -1.002499, & 0.554835, & & , \\
& -0.548936, & -1.199810, & 0.079208, & -1.828021, & -0.148437, \\
& 1.924226, & -0.358591, & -0.289980, & 0.192471, & 1.061458, \\
& 0.210731, & -0.092887, & 1.047664, & -1.124890, & -1.331140, \\
& 0.766161, & -0.272865, & -0.534926, & 0.999634, & -0.419161, \\
& -0.313326, & 0.407533, & -0.196529, & -0.737980, & -0.981490 \big)
\end{aligned}
$$

$r = 4.7987$

The total number of iterations was: 11.
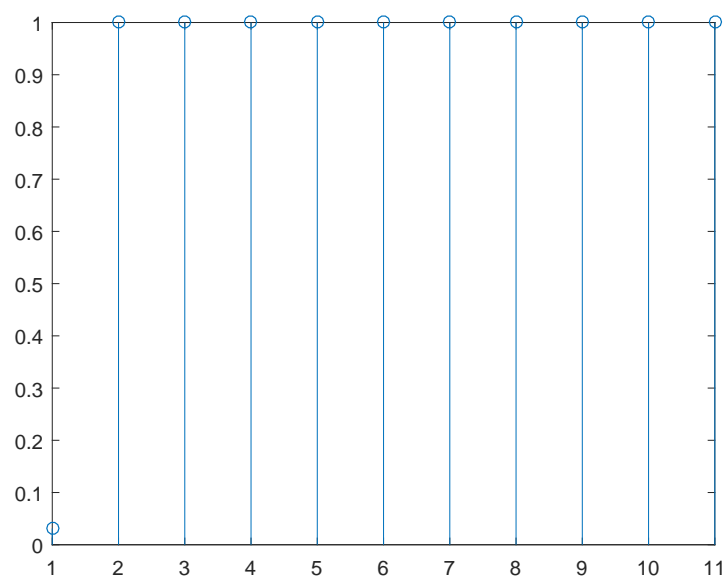
Figure 31: Value of alpha along iterations for data3.m

dataset4.m
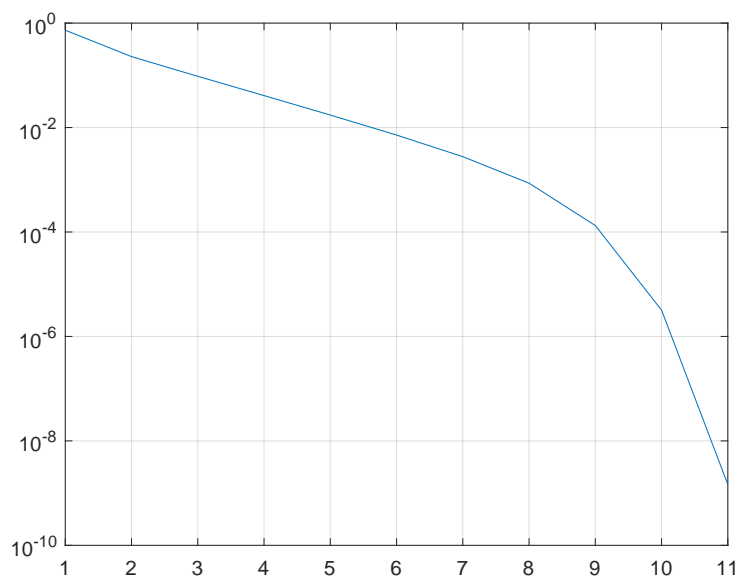
Figure 32: Norm of the gradient along iterations for dataset4.m.

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

$$r = 7.6718$$

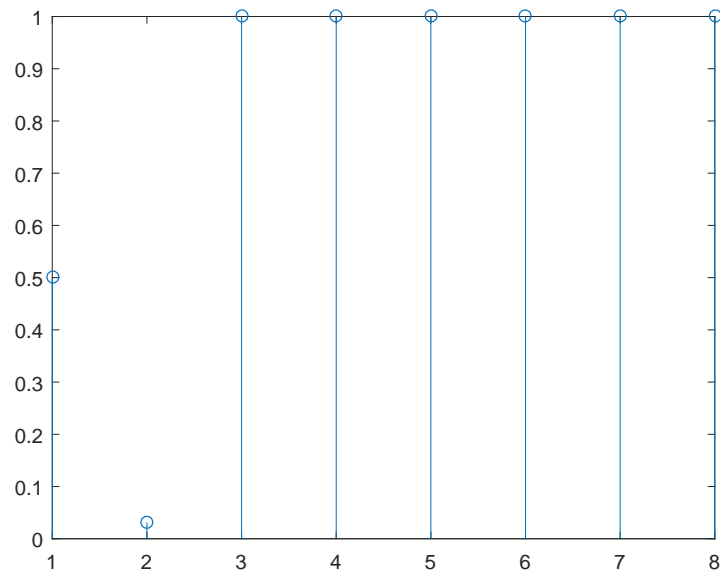The total number of iterations was: 11.
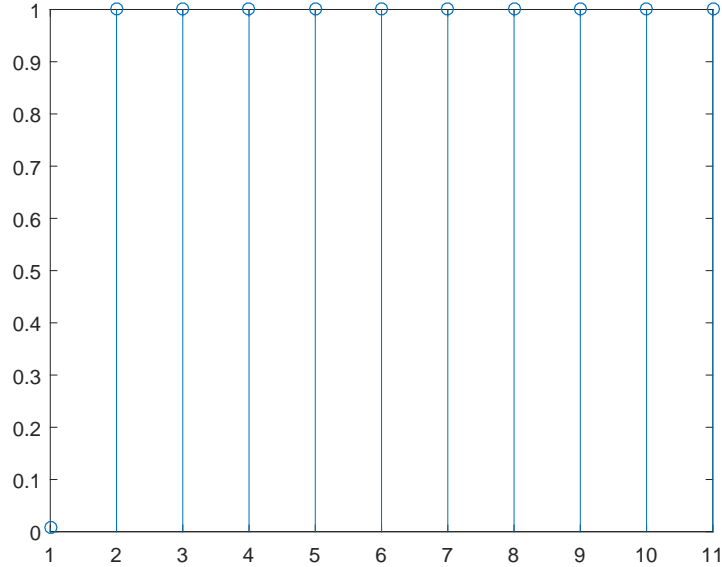
Figure 33: Value of alpha along iterations for dataset4.m

The following code was used for calculating the solutions for both datasets 3 and 4 (in

line 5 we specify which dataset we are using):

```matlab
1  clear all;
2  close all;
3
4  %load the workspace
5  load('data3.mat');
6
7  %%%%%%%%%%%%%%%%
8  %Gradient method
9  %%%%%%%%%%%%%%%%
10
11 %Amount of input features
12 [n, k] = size(X);
13
14 %Stopping criterion constants
15 s0 = -ones(1, n);
16 r0 = 0;
17 epslon = 10^(-6);
18
19 %Initial point for gradient descent
20 t0 = [s0 r0]'
21
22 %Backtracking parameters
23 alpha0 = 1;
24 y = 10^(-4);
25 beta = 0.5;
26
27 %Transformation of X
28 X_hat = [X; -ones(length(X), 1)'];
29
30 %Algorithm - Newton Method
31 t = t0;
32 alpha = alpha0;
33 alphas = [];
34 gradients = [];
35 while norm(gradient_f_hat(t, X_hat, Y, k)) >= epslon
36     g = gradient_f_hat(t, X_hat, Y, k);
37     d = -(hessian_f_hat(t, X_hat, Y, k)\g);
38     alpha = alpha0;
39     while f_hat(t + alpha.*d, X_hat, Y, k) >= f_hat(t, X_hat, Y, k) + (y.*g
           ↪ '*(alpha.*d))
40         alpha = beta .* alpha;
41     end
42     t = t + (alpha .* d);
43     %f_hat(t, X_hat, Y, k)
44     gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
45     alphas = [alphas alpha];
46 end
47
```

```
48  %plot alpha for each iteration
49  figure;
50  stem(alphas);
51
52  %plot figure with logarithmic y-axis
53  figure;
54  semilogy(gradients);
55  grid on;
56
57  s = t(1:n)
58  r = t(n+1)
```

Listing 16: Script used to calculate solutions for data3.m and dataset4.m

Task 7

Both the gradient and Newton methods converge to a very similar solution, for all datasets, as expected, as the cost function was proven to be convex, so there is only one minimum, the global minimum.

The newton method requires a very small number of iterations to converge, almost fixed in the order of 10, especially when compared to the gradient method, which required thousands of iterations for the datasets in small dimensions and tens of thousands for the dataset in medium dimensions.

This can be explained by the role of the Hessian matrix in the computation, that corrects the direction and norm of the the gradient for the iteration step. Analyzing the value of the backtracking variable alpha along the iterations (Figures 26, 29, 31 and 33), after the second or third iteration it is always 1, that is, the there is no need to reduce the iteration step, to get a better estimation of the minimum.

However, in the Newton method each iteration requires more computation, as the computation of the Hessian is of complexity $O(n^3)$, where n is the dimension of the problem, so it becomes too slow for higher dimensions. Although the gradient method requires much more iterations, each iteration is simpler, so it scales much better.

The choice between the minimization methods depends on the dimension of the problem. For problems in lower dimensions the Newton method is faster, but has the number of dimensions increase and the Hessian computations become too costly, the gradient method becomes more advantageous.

2   Network localization

Task 8

LM method addresses a nonlinear least-squares problems. The optimization problem to solve is

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

$$= \underset{x}{\text{minimize}} \quad \underbrace{\sum_{(m,p)\in\mathcal{A}} (g_{mp}(x))^2 + \sum_{(p,q)\in\mathcal{S}} (h_{pq}(x))^2}_{f(x)} . \tag{5}$$

The variable to minimize is

$$x = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_P \end{bmatrix},$$

that corresponds to each sensor position.

The function $g_{mp}(x)$ is given by $g_{mp}(x) = \|a_m - s_p\| - y_{mp}$, where m and p correspond to a pair of anchor and sensor in the set $\mathcal{A}$ and the function $h_{pq}(x)$ is given by $h_{pq}(x) \|s_p - s_q\| - z_{pq}$, where p and q corresponds to a sensor index of the pair of two sensor in the set $\mathcal{S}$. All these function are affine and differentiable, except when a_m is equals to s_p and s_p is equals to s_q. However, this does not represent a problem, because, physically the sensor and the anchors cannot have the same position.

To compute the LM algorithm, firstly, the gradient of the cost function and the gradient of all $g_{mp}(x)$ and $h_{pq}(x)$ functions are determined. Then, a least square problems is solved.

To simplify the problem, the functions are written in other notation, using the variable x to minimize.

$$g_{mp}(x) = \|a_m - s_p\| - y_{mp} = \|a_m - B_p x\| - y_{mp} \tag{6}$$

$$h_{pq}(x) = \|s_p - s_q\| - z_{pq} = \|E_{pq} x\| - y_{mp} \tag{7}$$

The matrix B_p has the goal to enable only one sensor s_p of each time. This matrix has 2 rows and 16 columns and corresponds to a matrix with all entries equal to zero and an identity matrix 2 by 2 placed in the respective columns of the sensor. For example, for the sensor 1 $(p = 1)$, this matrix is

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The matrix E_{pq} of equation 7 enables the connections between sensor s_p and s_q. This matrix is given by

$$E_{pq} = B_p - B_q, \tag{8}$$

so the matrix that enables the connection between sensor 1 $(p = 1)$ and sensor 8 $(p = 8)$, for example, is

$$E_{18} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Using the chain rule and the formula of the derivative of the composite function, applied to matrix differential calculus, the gradients of g_{mp} and h_{pq} are given by

$$\nabla g_{mp}(x) = -B_p^T \frac{a_m - B_p x}{\|a_m - B_p x\|} ,\tag{9}$$

$$\nabla h_{pq}(x) = -E_{pq}^T \frac{E_{pq} x}{\|E_{pq} x\|} .\tag{10}$$

The gradient of the cost function is given by

$$\nabla f(x) = \sum_{(m,p)\in\mathcal{A}} (\nabla g_{mp}(x))^2 + \sum_{(p,q)\in\mathcal{S}} (\nabla h_{pq}(x))^2 \tag{11}$$

$$= \sum_{(m,p)\in\mathcal{A}} 2 \left(\|a_m - B_p x\| - y_{mp} \right) (-B_p^T) \frac{a_m - B_p x}{\|a_m - B_p x\|} + \sum_{(p,q)\in\mathcal{S}} 2 \left(\|E_{pq} x\| - z_{pq} \right) (-E_{pq}^T) \frac{E_{pq} x}{\|E_{pq} x\|}$$

Th LM method tries to solve the optimization problem

$$\operatorname*{minimize}_{x} \sum_{p=1}^{P} (f_p(x_k) + \nabla f_p(x_k)^T (x - x_k))^2 + \lambda \|x - x_k\|^2 .\tag{12}$$

This equation can be rearrange as the following least square problem

$$\operatorname*{minimize}_{x} \|Ax - b\|^2$$

where

$$A = \begin{bmatrix} \nabla f_1(x_k)^T \\ \nabla f_2(x_k)^T \\ \vdots \\ \nabla f_p(x_k)^T \\ \sqrt{\lambda_k} I \end{bmatrix}$$

and

$$b = \begin{bmatrix} \nabla f_1(x_k)^T x_k - f_1(x_k) \\ \nabla f_2(x_k)^T x_k - f_2(x_k) \\ \vdots \\ \nabla f_p(x_k)^T x_k - f_p(x_k) \\ \sqrt{\lambda_k} x_k \end{bmatrix} .$$

With these matrix, the solution is obtained by solving a linear system of equations ($Ax = b$).

The LM method needs a initialization that corresponds to the initial position of the sensors. It also needs the initial value of the regularizer parameter (λ) and the minimum threshold, i.e, the value that is considered that the cost function is sufficiently minimized.

After running this algorithm, the final position of the sensor obtained are

$$s_1 = \begin{bmatrix} -2.0205 \\ 6.3478 \end{bmatrix}, \qquad s_2 = \begin{bmatrix} -2.5704 \\ 9.1908 \end{bmatrix}, \qquad s_3 = \begin{bmatrix} 3.7052 \\ -1.7748 \end{bmatrix}, \qquad s_4 = \begin{bmatrix} -4.4309 \\ -2.3808 \end{bmatrix},$$

$$s_5 = \begin{bmatrix} 1.6285 \\ -11.8744 \end{bmatrix}, \qquad s_6 = \begin{bmatrix} 3.3785 \\ 1.1180 \end{bmatrix}, \qquad s_7 = \begin{bmatrix} 4.2787 \\ 9.5108 \end{bmatrix}, \qquad s_8 = \begin{bmatrix} -1.9070 \\ -5.2535 \end{bmatrix}$$

The plot of the network obtained is represented in Figure 34.
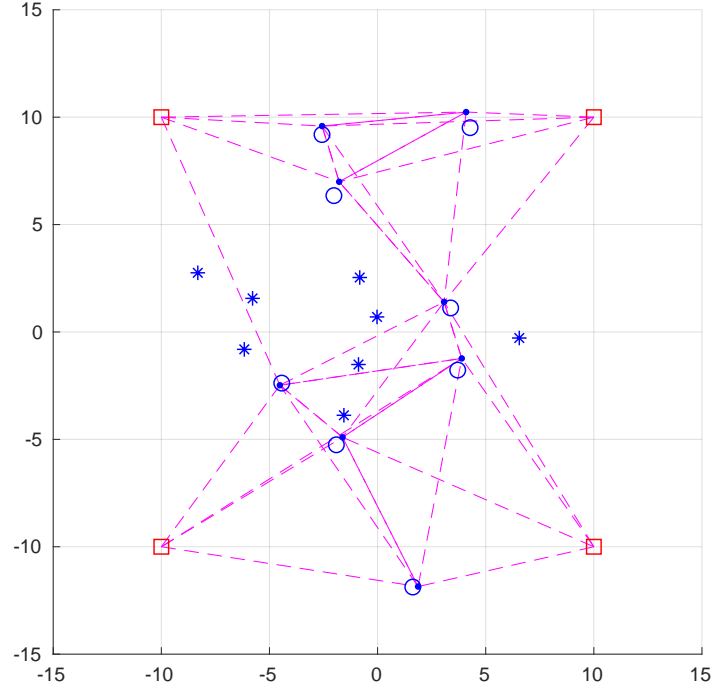
Figure 34: Network Localization

Using the LM algorithm to minimize the optimization problem (5), 17 iterations are needed until the criterion of minimization is accomplish, i.e., when the norm of the gradient of the cost function is less than 10^{-6}. The value of the cost function at the minimum found by the algorithm is $f(x_k) = 4,1651$ (with $k = 17$). The plot of the norm of the gradient for each iteration are represented in Figure 35.
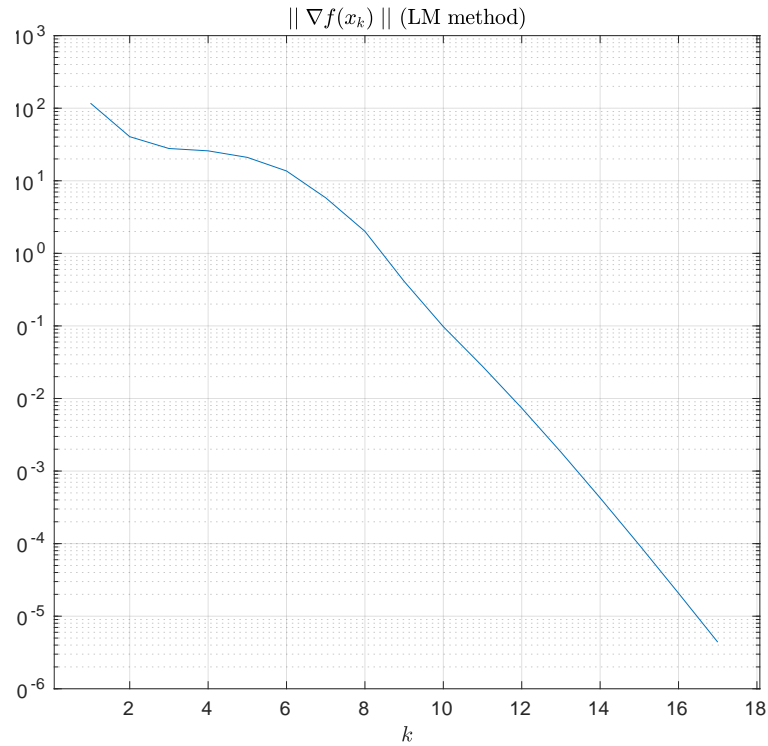
Figure 35: Norm of the gradient along the iterations of the LM method

```matlab
clear all;
close all;

%load the workspace
load('lmdata1.mat');

lambda = 1;
tolerance = power(10, -6);

iter = 0;
plot_n_grad = [];

%indentity matrix 16x16
I = eye(2);
I_a = eye(16);

B = zeros(2, 16, 8);
E = zeros(2, 16, 24);

for i=1:1:8
    B(:, 2*i-1:2*i, i) = I;
end
```

69

```matlab
23  for j=1:1:size(iS)
24      E(:,:,j) = B(:,:,iS(j,1))-B(:,:,iS(j,2));
25  end
26
27  xk = xinit;
28  n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );
29
30
31  while n_grad > tolerance
32
33      gradi_fp = gradient_fp(iA, A, iS, B, E, xk);
34      func_fp = fp(A, iA, iS, B, y, z, xk, E);
35
36      b_aux = gradi_fp'*xk  - func_fp;
37      v_lambda = sqrt(lambda).*xk;
38      b = [b_aux; v_lambda];
39
40      A_aux = sqrt(lambda) .* I_a;
41      A_ = [gradi_fp'; A_aux];
42
43      belief = A_\b;
44
45      f_belief = f(A, iA, iS, B, y, z, belief, E);
46      f_xk = f(A, iA, iS, B, y, z, xk, E);
47
48      if f_belief < f_xk
49          xk = belief;
50          lambda = 0.7* lambda;
51      else
52          lambda = 2* lambda;
53      end
54
55      iter = iter+1;
56      plot_n_grad(iter) = n_grad;
57      %data for the next iteration
58      n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );
59
60
61  end
62
63  sensor = zeros(2,8);
64  for i=1:1:8
65      sensor(:,i) = xk(2*i-1:2*i);
66  end
67  plotgraph(A, iA, iS, sensor, xinit, S, plot_n_grad, iter);
```

Listing 17: Script Task 8

```matlab
1  function [func] = f(A, iA, iS, B, y, z, x, E)
2      %returns a number
3      f_ = fp(A, iA, iS, B, y, z, x, E).^2;
4      func = sum(f_);
5
6  end
```

Listing 18: Function to calculate $f(x)$

```matlab
1  function [f_] = fp(A, iA, iS, B, y, z, x, E)
2      %returns a (px1) vector (p=size(iA)+size(iS))
3      sA =  size(iA, 1);
4      sS =  size(iS, 1);
5      f_ = zeros(sA+sS, 1);
6
7      for j=1:1:sA
8          f_(j) = norm(A(:,iA(j,1)) - B(:,:,iA(j,2))*x) - y(j);
9      end
10
11     for i=1:1:sS
12         f_(i+sA) = norm(E(:,:,i)*x) - z(i);
13     end
14
15 end
```

Listing 19: Function to calculate $g_{mp}(x)$ and $h_{pq}(x)$

```matlab
1  function [gradientf] = gradient_f(A, iA, iS, B, y, z, x, E)
2  %return a column of 16x1 with all the parcial derivatives of f
3      sA =  size(iA, 1);
4      sS =  size(iS, 1);
5      grad_f = zeros(16, sS+sA);
6
7      for i=1:1:sA
8          grad_f(:,i) = 2*(norm(A(:,iA(i,1))-B(:,:,iA(i,2))*x)-y(i))*(-B(:,:,
                ↪ iA(i,2))'*(A(:,iA(i,1))-B(:,:,iA(i,2))*x))./(norm(A(:,iA(i,1)
                ↪ )-B(:,:,iA(i,2))*x));
9      end
10     for i=1:1:sS
11         grad_f(:,i+sA) = 2*(norm(E(:,:,i)*x)-z(i))*(E(:,:,i)'*(E(:,:,i)*x))
                ↪ ./(norm(E(:,:,i)*x));
12     end
13     %sum of the rows
14     gradientf = sum(grad_f, 2);
15 end
```

Listing 20: Function to calculate the gradient of $f(x)$

```matlab
function [gradientfp] = gradient_fp(iA, A, iS, B, E, x)
    %return the gradient of fp (16xp)

    sA =  size(iA, 1);
    sS =  size(iS, 1);
    gradientfp = zeros(16, sS+sA);

    for i=1:1:sA
        gradientfp(:,i) = (-B(:,:,iA(i,2))' * (A(:,iA(i,1))- B(:,:,iA(i,2))
            ↪ *x))./(norm(A(:,iA(i,1)) - B(:,:,iA(i,2))*x));
    end
    for  i=1:1:sS
        gradientfp(:,i+sA) = (E(:,:,i)'*( E(:,:,i)*x))./(norm(E(:,:,i)*x));
    end

end
```

Listing 21: Function to calculate the gradient of $g_{mp}(x)$ and $h_{pq}(x)$

```matlab
function [] = plotgraph(A, iA, iS, s, xinit, S, n_grad, iter)

sinit = zeros(2,8);
for i=1:1:8
    sinit(:,i) = xinit(2*i-1:2*i);
end

figure(1);
hold on
xlim([-15 15])
ylim([-15 15])


for i=1:1:size(iA,1)
    plot([A(1,iA(i,1)) S(1,iA(i,2))], [A(2,iA(i,1)) S(2,iA(i,2))], '--', '
        ↪ Color', 'm');
end
for i=1:1:size(iS,1)
    plot([S(1,iS(i,1)) S(1,iS(i,2))], [S(2,iS(i,1)) S(2,iS(i,2))], '--', '
        ↪ Color', 'm');
end

plot(A(1,:),A(2,:), 's', 'MarkerSize', 10, 'MarkerEdgeColor', 'r');
plot(sinit(1,:), sinit(2,:), '*', 'MarkerSize',7, 'MarkerEdgeColor', 'b');
plot(S(1,:), S(2,:), '.', 'MarkerSize',10, 'MarkerEdgeColor', 'b');
plot(s(1,:),s(2,:), 'o', 'MarkerSize',8, 'MarkerEdgeColor', 'b');
grid on

v_i = 1:1:iter;
figure(2);
```

```
29  semilogy(v_i, n_grad);
30  xlabel('$$k$$','Interpreter','latex');
31  title('$$\mid\mid \nabla f(x_k) \mid\mid$$ (LM method)','Interpreter','
      ↪ latex');
32  grid on
33  end
```

Listing 22: Function plot the network and the norm of the gradient

Task 9

The LM algorithm always needs a initialization. This initialization is very important and it will affect the way that the function is minimize. For example, if the function to minimize has several local minimums, and if the initialization is close to one of this global minimums, there is a high probability that the function will be minimize to this local minimum, instead of the global minimum. To avoid these situations, the LM algorithm is computed for several times from different initial positions.

Using the same LM algorithm as in the previous task, the LM method is started from five million different random initializations. The best solution obtained, i.e., the smallest value of the cost function, is $f(x_k) = 4.4945$. The plots of the network obtained and the norm of the gradient along the iterations are represented in Figure 36 and Figure 37.
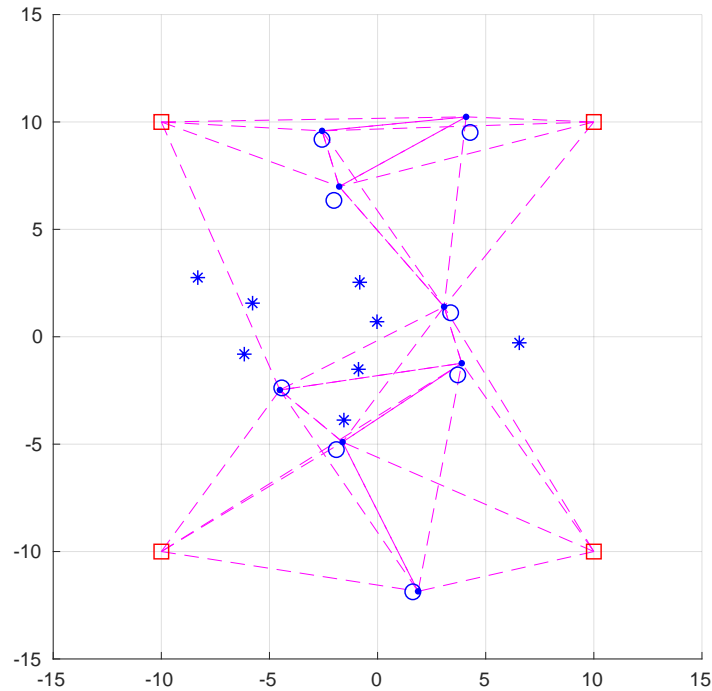
Figure 36: Network Localization
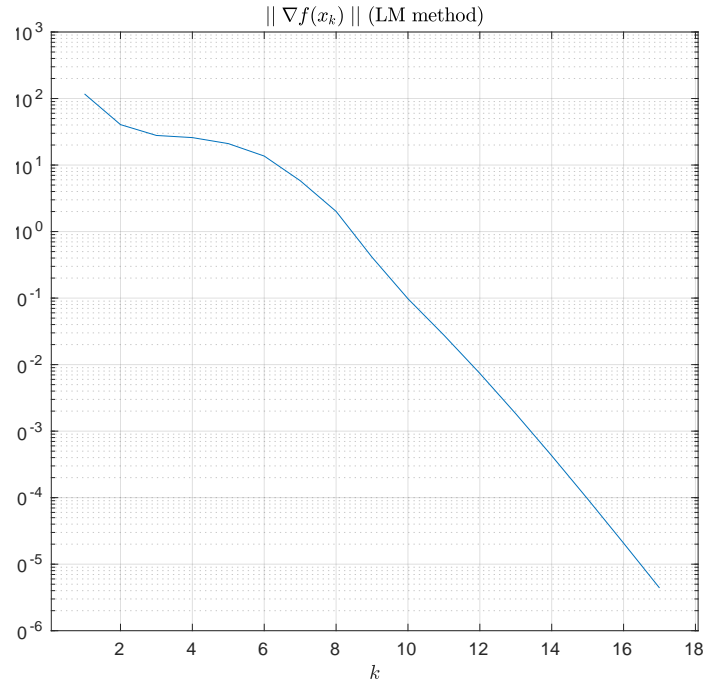
Figure 37: Norm of the gradient along the iterations of the LM method

```matlab
1  clear all;
2  close all;
3  cost_function = 1000000;
4
5
6  for it=1:1:5000000
7
8      clearvars −except cost_function x_best iter_best plot_n_grad_best it
           ↪ x_bestinit
9
10     %load the workspace
11     load('lmdata2.mat');
12
13     lambda = 1;
14     tolerance = power(10, −6);
15
16     iter = 0;
17     plot_n_grad = [];
18
19     %generate random vector of 16x1 in the interval of [−11 11]
20     xinit = −11 + (11+11).*rand(16,1);
21
22     %indentity matrix 16x16
23     I = eye(2);
```

```matlab
24        I_a = eye(16);
25
26        B = zeros(2, 16, 8);
27        E = zeros(2, 16, 24);
28
29        for i=1:1:8
30            B(:, 2*i-1:2*i, i) = I;
31        end
32        for j=1:1:size(iS)
33            E(:,:,j) = B(:,:,iS(j,1))-B(:,:,iS(j,2));
34        end
35
36        xk = xinit;
37        n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );
38
39
40        while n_grad > tolerance
41
42            gradi_fp = gradient_fp(iA, A, iS, B, E, xk);
43            func_fp = fp(A, iA, iS, B, y, z, xk, E);
44
45            b_aux = gradi_fp'*xk  - func_fp;
46            v_lambda = sqrt(lambda).*xk;
47            b = [b_aux; v_lambda];
48
49            A_aux = sqrt(lambda) .* I_a;
50            A_ = [gradi_fp'; A_aux];
51
52            belief = A_\b;
53
54            f_belief = f(A, iA, iS, B, y, z, belief, E);
55            f_xk = f(A, iA, iS, B, y, z, xk, E);
56
57            if f_belief < f_xk
58                xk = belief;
59                lambda = 0.7* lambda;
60            else
61                lambda = 2* lambda;
62            end
63
64            iter = iter+1;
65            plot_n_grad(iter) = n_grad;
66            %data for the next iteration
67            n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );
68
69            if iter > 1000
70                %diverge
71                break
72            end
73        end
74
```

```
75
76     f_final = f(A, iA, iS, B, y, z, xk, E);
77
78     if f_final < cost_function
79         cost_function = f_final;
80         x_bestinit = xinit;
81         x_best = xk;
82         iter_best = iter;
83         plot_n_grad_best = plot_n_grad;
84     end
85 end
86
87
88 sensor = zeros(2,8);
89 for i=1:1:8
90     sensor(:,i) = x_best(2*i-1:2*i);
91 end
92 plotgraph_task9(A, iA, iS, sensor,x_bestinit, plot_n_grad_best,iter_best);
```

Listing 23: Script of task 9

```
1  function [] = plotgraph(A, iA, iS, s, xinit, S, n_grad, iter)
2
3  sinit = zeros(2,8);
4  for i=1:1:8
5      sinit(:,i) = xinit(2*i-1:2*i);
6  end
7
8  figure(1);
9  hold on
10 xlim([-15 15])
11 ylim([-15 15])
12
13
14 for i=1:1:size(iA,1)
15     plot([A(1,iA(i,1)) S(1,iA(i,2))], [A(2,iA(i,1)) S(2,iA(i,2))], '--', '
        ↪ Color', 'm');
16 end
17 for i=1:1:size(iS,1)
18     plot([S(1,iS(i,1)) S(1,iS(i,2))], [S(2,iS(i,1)) S(2,iS(i,2))], '--', '
        ↪ Color', 'm');
19 end
20
21 plot(A(1,:),A(2,:), 's', 'MarkerSize', 10, 'MarkerEdgeColor', 'r');
22 plot(sinit(1,:), sinit(2,:), '*', 'MarkerSize',7, 'MarkerEdgeColor', 'b');
23 plot(S(1,:), S(2,:), '.', 'MarkerSize',10, 'MarkerEdgeColor', 'b');
24 plot(s(1,:),s(2,:), 'o', 'MarkerSize',8, 'MarkerEdgeColor', 'b');
25 grid on
26
27 v_i = 1:1:iter;
```

```
28  figure(2);
29  semilogy(v_i, n_grad);
30  xlabel('$$k$$','Interpreter','latex');
31  title('$$\mid\mid \nabla f(x_k) \mid\mid$$ (LM method)','Interpreter','
       ↪ latex');
32  grid on
```

Listing 24: Function plot the network and the norm of the gradient

Part 3

Task 1

We want to determine a closed form solution to the problem

$$\begin{aligned} \underset{y \in \mathbf{R}^n}{\text{minimize}} \quad & \|p - y\|_2 \\ \text{subject to} \quad & y \in D(c, r). \end{aligned} \tag{13}$$

To solve this problem, we used the KKT conditions. Considering a generic problem where we want to minimize the cost function $f(x)$.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & h(x) = 0; g(x) \leq 0. \end{aligned} \tag{14}$$

This theorem says if x^* is a regular local minimum, there exist $\lambda^* \in R^p, \mu^* \in R^m$ such that

$$\begin{cases} \nabla f(x^*) + \nabla h(x^*)\lambda^* + \nabla g(x^*)\mu^* = 0 \\ h(x^*) = 0 \\ g(x^*) \leqslant 0 \\ \mu^* \geqslant 0 \\ g(x^*)^T \mu^* = 0 \end{cases}$$

We need to rewrite the problem 13 as the previous system of equation. The cost function can be write as $\frac{1}{2}\|p - y\|_2^2$, to be easy to determine the gradient. The constrain can be written as $D(c, r) = \{y\colon \|y - c\|_2 \leq r\} = \{y\colon \frac{1}{2}\|y - c\|_2^2 - \frac{1}{2}r^2 \leq 0\}$

A new problem can be rewrite.

$$\begin{aligned} \underset{y \in \mathbf{R}^n}{\text{minimize}} \quad & \tfrac{1}{2}\|p - y\|_2^2 \\ \text{subject to} \quad & \tfrac{1}{2}\|y - c\|_2^2 - \tfrac{1}{2}r^2 \leq 0. \end{aligned} \tag{15}$$

Doing a parallelism with equation 14, we obtain

$$f(y) = \frac{1}{2}\|p - y\|_2^2 \tag{16}$$

$$g(y) = \frac{1}{2}\|y - c\|_2^2 - \frac{1}{2}r^2 \tag{17}$$

Now, we need to determine the gradient of $f(y)$ and $g(y)$.

$$\nabla f(y) = \nabla(\frac{1}{2}\|p - y\|_2^2) = \frac{1}{2}\nabla\|p - y\|_2^2 = \frac{1}{2}\nabla(p - y)^2 = y - p \tag{18}$$

$$\nabla g(y) = \nabla(\frac{1}{2} \|y - c\|_2^2 - \frac{1}{2}r^2) = \nabla(\frac{1}{2} \|y - c\|_2^2) - \nabla(\frac{1}{2}r^2) \tag{19}$$

$$= \nabla(\frac{1}{2} \|y - c\|_2^2) - 0 = \nabla\frac{1}{2}(y - c)^2 = y - c$$

The KKT system is

$$\begin{cases} (y - p) + \mu^*(y - c) = 0 \\ \frac{1}{2} \|y - c\|_2^2 - \frac{1}{2}r^2 \leqslant 0 \\ \mu^* \geqslant 0 \\ \mu^*(\frac{1}{2} \|y - c\|_2^2 - \frac{1}{2}r^2)^T = 0 \end{cases}$$

Using the first equation, we can determine y in function of μ.

$$(y - p) + \mu^*(y - c) = 0 \Leftrightarrow y - p + \mu^* y - \mu^* c \Leftrightarrow y(\mu^* + 1) = p + \mu^* c \Leftrightarrow y = \frac{p + \mu^* c}{\mu^* + 1} \tag{20}$$

The result obtain in 20, we apply in the last equation of the KKT system.

$$\mu^*(\frac{1}{2} \|y - c\|_2^2 - \frac{1}{2}r^2)^T = 0 \Leftrightarrow \mu^*(\frac{1}{2} \left\|\frac{p + \mu^* c}{\mu^* + 1} - c\right\|_2^2 - \frac{1}{2}r^2)^T = 0 \Leftrightarrow \tag{21}$$

$$\Leftrightarrow \mu^* = 0 \vee (\frac{1}{2} \left\|\frac{p + \mu^* c}{\mu^* + 1} - c\right\|_2^2 - \frac{1}{2}r^2)^T = 0 \Leftrightarrow \mu^* = 0 \vee \frac{1}{2} \left\|\frac{p + \mu^* c}{\mu^* + 1} - c\right\|_2^2 - \frac{1}{2}r^2 = 0 \Leftrightarrow$$

$$\Leftrightarrow \mu^* = 0 \vee \frac{1}{2} \left\|\frac{p + \mu^* c}{\mu^* + 1} - c\right\|_2^2 = \frac{1}{2}r^2 \Leftrightarrow \mu^* = 0 \vee \left\|\frac{p + \mu^* c}{\mu^* + 1} - c\right\|_2^2 = r^2 \Leftrightarrow$$

$$\Leftrightarrow \mu^* = 0 \vee (\frac{p + \mu^* c}{\mu^* + 1} - c)^T(\frac{p + \mu^* c}{\mu^* + 1} - c) = r^2 \Leftrightarrow \mu^* = 0 \vee \frac{1}{(\mu^* + 1)^2}(p - c)^T(p - c)^T = r^2 \Leftrightarrow$$

$$\Leftrightarrow \mu^* = 0 \vee \frac{1}{(\mu^* + 1)^2} \|p - c\|_2^2 = r^2 \Leftrightarrow \mu^* = 0 \vee (\mu^* + 1)^2 = \frac{\|p - c\|_2^2}{r^2} \Leftrightarrow$$

$$\Leftrightarrow \mu^* = 0 \vee \mu^* + 1 = \pm\frac{\|p - c\|_2}{r} \Leftrightarrow \mu^* = 0 \vee \mu^* = -1 \pm \frac{\|p - c\|_2}{r}$$

However, using the inequality $\mu^* \geqslant 0$, we conclude that if $\|p - c\|_2 \geqslant r$, that is, if the point p is outside of the circle with center in c and radius r, μ^* is equal to $\mu^* = -1 + \frac{\|p - c\|_2}{r}$. If the distance of point p to the center c is less than the radius (point p is inside of the circle with center in c and radius r), i.e., $\|p - c\|_2 < r$, $\mu^* = 0$

$$\begin{cases} \mu^* = 0, & \text{if } \|p - c\|_2 < r \\ \mu^* = -1 + \frac{\|p-c\|_2}{r}, & \text{if } \|p - c\|_2 \geqslant r \end{cases}$$

This result can be write as a closed form solution.

$$\mu^* = (-1 + \frac{\|p - c\|_2}{r})_+ \tag{22}$$

Replacing this result in equation 20, it's obtained a closed form solution for y^*.

$$y^* = \frac{p + (-1 + \frac{\|p-c\|_2}{r})_+}{(-1 + \frac{\|p-c\|_2}{r})_+ + 1} \tag{23}$$

Developing the equation 23, we obtain the following result.

$$\begin{cases} y = p, & \text{if } \|p - c\|_2 < r \\ y = \frac{p+c(-1+\frac{\|p-c\|_2}{r})}{1+\frac{\|p-c\|_2}{r}+1}, & \text{if } \|p - c\|_2 \geqslant r \end{cases} \Leftrightarrow \begin{cases} y = p, & \text{if } \|p - c\|_2 < r \\ y = \frac{p+c(-1+\frac{\|p-c\|_2}{r})}{\frac{\|p-c\|_2}{r}}, & \text{if } \|p - c\|_2 \geqslant r \end{cases}$$

$$\Leftrightarrow \begin{cases} y = p, & \text{if } \|p - c\|_2 < r \\ y = \frac{p-c+c\frac{\|p-c\|_2}{r}}{\frac{\|p-c\|_2}{r}}, & \text{if } \|p - c\|_2 \geqslant r \end{cases} \Leftrightarrow \begin{cases} y = p, & \text{if } \|p - c\|_2 < r \\ y = c + r\frac{p-c}{\|p-c\|_2}, & \text{if } \|p - c\|_2 \geqslant r \end{cases}$$

This result means that, when the point p is inside the disk centered in c and with radius of r, the distance is equal to zero, so the point itself is the closest to the disk.

$$\|p - y^*\| = \|p - p\| = 0, \text{if } \|p - c\|_2 \geqslant r \tag{24}$$

When the point p is outside the disk centered in c and with radius of r, the distance between p and the disk is given by

$$\|p - y^*\| = \left\| p - c + r\frac{p - c}{\|p - c\|_2} \right\|, \text{if } \|p - c\|_2 \geqslant r \tag{25}$$

Task 2

We chose problem A to solve

$$\begin{aligned} \underset{x,u}{\text{minimize}} \quad & \sum_{k=1}^{K} \|Ex(\tau_k) - \omega_k\|_2^2 + \lambda \sum_{t=1}^{T-1} \|u(t) - u(t-1)\|_2^2 \\ \text{subject to} \quad & x(0) = x_i; \\ & x(T) = x_f; \\ & x(t+1) = Ax(t) + Bu(t), \quad for\ 0 \leq t \leq T - 1; \end{aligned} \tag{26}$$

with $x(t) \in \mathbb{R}^4$ for $t = 0 \ldots T$ and $u(t) \in \mathbb{R}^2$ for $t = 0 \ldots T - 1$.

To solve this optimization analytically it is better to express it with only one vector variable, by concatenating all original variables

$$z = \begin{bmatrix} x(0) \\ \ldots \\ x(T) \\ u(0) \\ \ldots \\ u(T-1) \end{bmatrix} \in \mathbb{R}^n, \quad n = 4(T+1) + 2T.$$

The matrices X_t and Y_t are defined for index t to recover $x(t)$ and $y(t)$, respectively

$$
\begin{aligned}
x(t) = X_t z, \quad X_t = \begin{bmatrix} 0 & \ldots & \mathbf{I} & \ldots & 0 & 0 & \ldots & 0 & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{4 \times n}, \\
u(t) = Y_t z, \quad Y_t = \begin{bmatrix} 0 & \ldots & 0 & \ldots & 0 & 0 & \ldots & \mathbf{I} & \ldots & 0 \end{bmatrix} \in \mathbb{R}^{2 \times n}.
\end{aligned}
\tag{27}
$$

Using this new variable and expressing the ℓ_2^2 norm as matrix multiplication, the optimization problem with the new variable becomes:

$$
\begin{aligned}
\underset{z}{\text{minimize}} \quad & \sum_{k=1}^{K} (EX_{\tau_k} z - \omega_k)^T I (EX_{\tau_k} z - \omega_k) + \sum_{t=1}^{T-1} (Y_t z - Y_{t-1} z)^T \lambda I (Y_t z - Y_{t-1} z) \\
\text{subject to} \quad & X_0 z = x_i; \\
& X_T z = x_f; \\
& X_{T+1} z = A X_T z + B Y_T z, \quad for\ 0 \le t \le T - 1;
\end{aligned}
\tag{28}
$$

This optimization can be expressed in a more compact way, as the cost function is a sum of quadratic functions, which is itself a quadratic

$$
\sum_{k=1}^{K} (EX_{\tau_k} z - \omega_k)^T \mathbf{I} (EX_{\tau_k} z - \omega_k) + \sum_{t=1}^{T-1} (Y_t z - Y_{t-1} z)^T \lambda \mathbf{I} (Y_t z - Y_{t-1} z)
$$

$$
= \left(\begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \end{bmatrix} \right)^T \mathbf{I} \left(\begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \end{bmatrix} \right) + \left(\begin{bmatrix} Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z \right)^T \lambda \mathbf{I} \left(\begin{bmatrix} Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z \right)
$$

$$
= \left(\begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \\ Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \\ 0 \\ \ldots \\ 0 \end{bmatrix} \right)^T \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I} \end{bmatrix} \left(\begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \\ Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \\ 0 \\ \ldots \\ 0 \end{bmatrix} \right)
$$

$$
= (Rz - v)^T D (Rz - v)
$$

The constraints can also be stacked in matrix form, as an affine expression

$$
\begin{bmatrix} X_0 \\ X_T \\ AX_0 - X_1 + BY_0 \\ \dots \\ AX_{T-1} - X_T + BY_{T-1} \end{bmatrix} z = \begin{bmatrix} x_i \\ x_f \\ 0 \\ \dots \\ 0 \end{bmatrix} \Leftrightarrow Lz = t. \tag{29}
$$

Then, the optimization problem has the structure

$$
\begin{aligned}
\underset{z}{\text{minimize}} \quad & (Rz - v)^T S(Rz - v) \\
\text{subject to} \quad & Lz - t = 0
\end{aligned}, \tag{30}
$$

where

$$
R = \begin{bmatrix} EX_{\tau_1} \\ \dots \\ EX_{\tau_K} \\ Y_1 - Y_0 \\ \dots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad v = \begin{bmatrix} x_i \\ x_f \\ 0 \\ \dots \\ 0 \end{bmatrix} \in \mathbb{R}^m, \qquad D = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \lambda\mathbf{I} \end{bmatrix} \in \mathbb{R}^{m \times m},
$$

$$
L = \begin{bmatrix} X_0 \\ X_T \\ AX_0 - X_1 + BY_0 \\ \dots \\ AX_{T-1} - X_T + BY_{T-1} \end{bmatrix} \in \mathbb{R}^{w \times n}, \qquad t = \begin{bmatrix} x_i \\ x_f \\ 0 \\ \dots \\ 0 \end{bmatrix} \in \mathbb{R}^w,
$$

with

$$
n = 4(T+1) + 2T \qquad m = 2K + 2(T-1) \qquad w = 4(2+T). \tag{31}
$$

This constrained optimization is very similar with one of the problems studied in the course lectures, i.e. quadratic cost function with affine constraint, and can be solved using the Karush-Kuhn-Tucker (KTT) conditions.

The cost function is convex, as it is the composition of an affine mapping $(Rz - v)$ and a convex quadatric function. Matrix S is diagonal, with all the values, that correspond to the eigenvalues, positive, so it is definite positive.

As this optimization problem only has equalities as constraints, the conditions are

$$
\begin{cases} \nabla f(z^*) + \nabla h(z^*)\lambda^* = 0 \\ h(z^*) = 0 \end{cases}, \tag{32}
$$

where

$$f(z) = (Rz - v)^T D(Rz - v), \tag{33}$$
$$h(z) = Lz - t. \tag{34}$$

The gradients of these functions are, using matrix differential algebra and noting that matrix D is symmetric,

$$\nabla f(z) = 2R^T S(Rz - v), \tag{35}$$
$$\nabla h(z) = L^T. \tag{36}$$

The system to solve is

$$\begin{cases} 2R^T D(Rz^* - v) + L^T \lambda^* = 0 \\ Lz^* - t = 0 \end{cases}, \tag{37}$$

and can be solved by first isolating z^* in the first equation (note that $R^T SR$ is a square matrix)

$$\begin{aligned} & 2(R^T DR)z^* - 2R^T Dv = -L^T \lambda^* \\ \Leftrightarrow & 2(R^T DR)z^* = 2R^T Dv - L^T \lambda^* \\ \Leftrightarrow & z^* = (R^T DR)^{-1}(R^T Dv - \frac{1}{2}L^T \lambda^*). \end{aligned} \tag{38}$$

Replacing z^* in the second equation, λ^* can be obtained ($L(R^T DR)^{-1}L^T$ is also a square matrix),

$$\begin{aligned} & L(R^T DR)^{-1}(R^T Dv - \frac{1}{2}L^T \lambda^*) - t = 0 \\ \Leftrightarrow & L(R^T DR)^{-1}R^T Dv - \frac{1}{2}L(R^T DR)^{-1}L^T \lambda^* - t = 0 \\ \Leftrightarrow & \frac{1}{2}(L(R^T DR)^{-1}L^T)\lambda^* = L(R^T DR)^{-1}R^T Dv - t \\ \Leftrightarrow & \lambda^* = 2(L(R^T DR)^{-1}L^T)^{-1}(L(R^T DR)^{-1}R^T Dv - t), \end{aligned} \tag{39}$$

with which z^* can be obtained from (38)

$$z^* = (R^T DR)^{-1}[R^T Dv - L^T(L(R^T DR)^{-1}L^T)^{-1}(L(R^T DR)^{-1}R^T Dv - t)] \tag{40}$$

Defining the auxiliary vector and matrix

$$S = R^T DR, \quad p = R^T Dv, \tag{41}$$

the final closed form expression is very similar to the one presented in the lecture

$$z^* = S^{-1}(p + L^T(LS^{-1}L^T)^{-1}(t - LS^{-1}p)). \tag{42}$$

This is the only KTT point, so it is the solution.

Errata

On Part 2, Task 5 b) (page 55), in some of the intermediate steps in the Hessian calculation, there is a missing ϕ and some resulting bad formatting,

$$\mathbf{H}p(x) = \begin{bmatrix} \sum_{k=1}^{K} \ddot{\phi}(a_k{}^T x)a_{k1}a_{k1} & \dots & \dots \\ \dots & & \dots \\ \dots & & \end{bmatrix} = \sum_{k=1}^{K} \ddot{\phi}(a_k{}^T x) \begin{bmatrix} a_{k1}a_{k1} & a_{k1}a_{k1} & a_{k1}a_{k3} \\ a_{k2}a_{k1} & a_{k2}a_{k2} & a_{k2}a_{k3} \\ a_{k3}a_{k1} & a_{k3}a_{k2} & a_{k3}a_{k3} \end{bmatrix},$$

$$\mathbf{H}p(x) = \sum_{k=1}^{K} \ddot{\phi}(a_k{}^T x)a_k a_k{}^T = \sum_{k=1}^{K} a_k \ddot{\phi}(a_k{}^T x)a_k{}^T$$

On Part 2, Task 9 (page 73), in the Figure 36 and Figure 37 should be
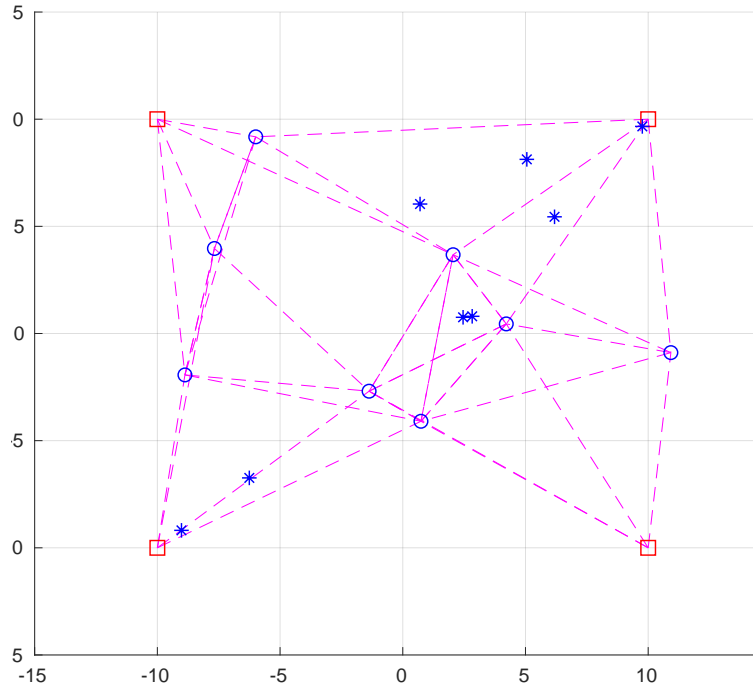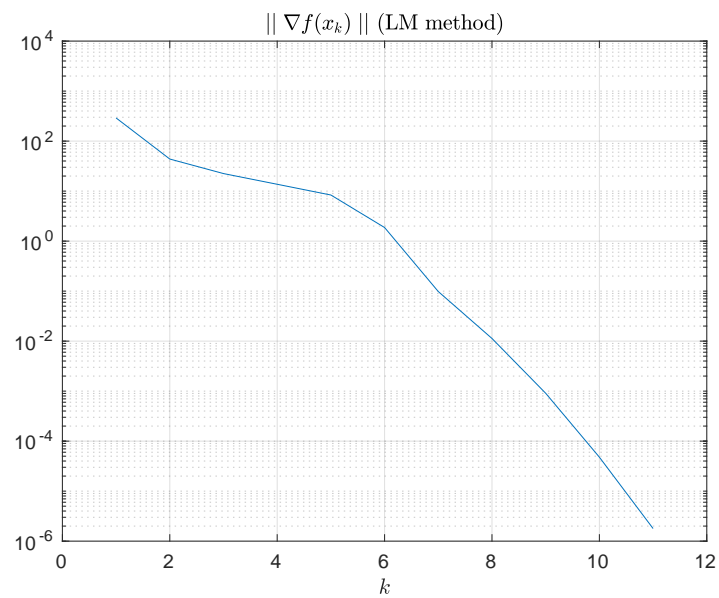

Figure 36: Network Localization

Figure 37: Norm of the gradient along the iterations of the LM method