# Optimization and Algorithms
# Project report

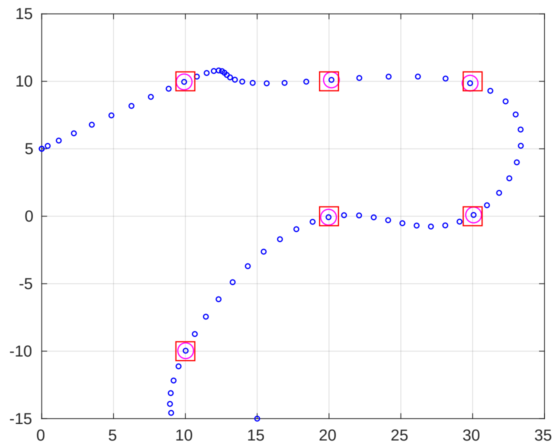
Group 2
Miguel Pinho 80826, Pedro Mendes 81046, Duarte Dias 81356

# Part 1

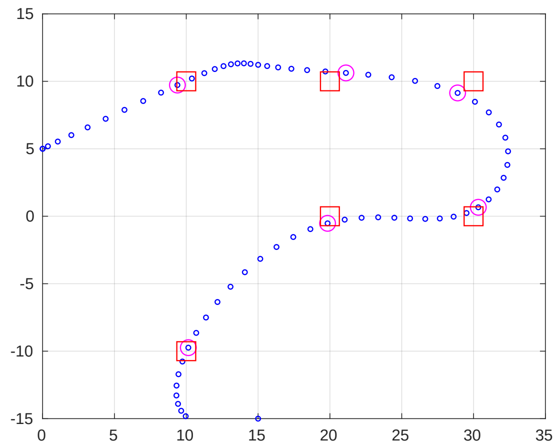
## Task 1

### a)

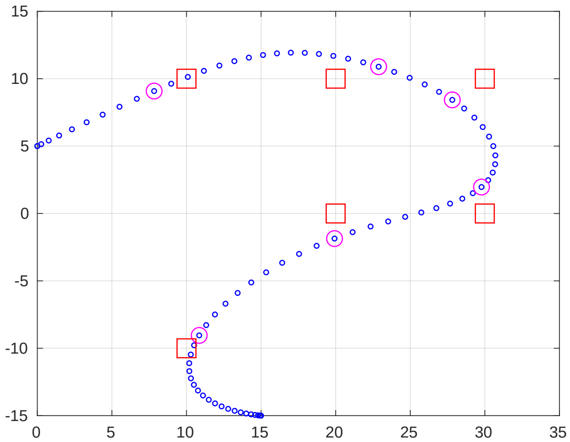
In Figure 1 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times $\tau_k$, for the different values of $\lambda$ parameter, when the cost function uses $\ell_2^2$ regularizer.



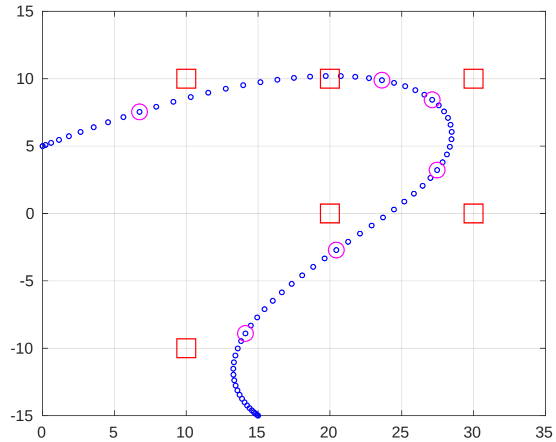

(a) $\lambda = 10^{-3}$



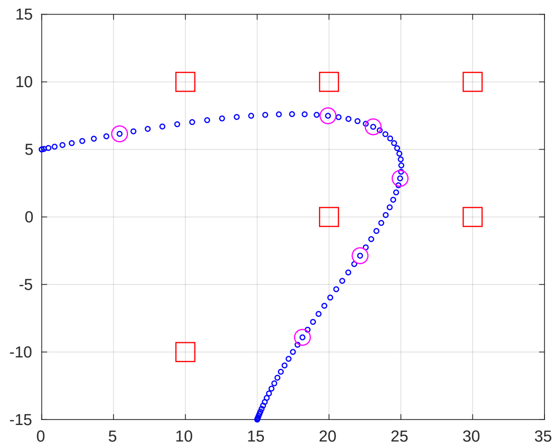

(b) $\lambda = 10^{-2}$



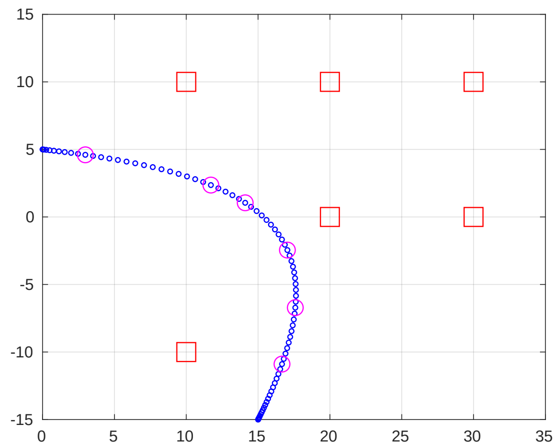

(c) $\lambda = 10^{-1}$



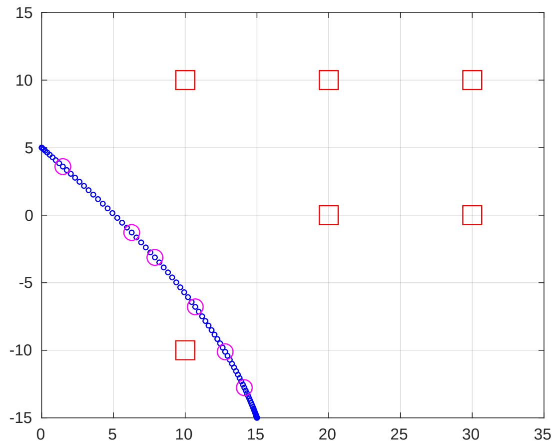

(d) $\lambda = 10^{0}$




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




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

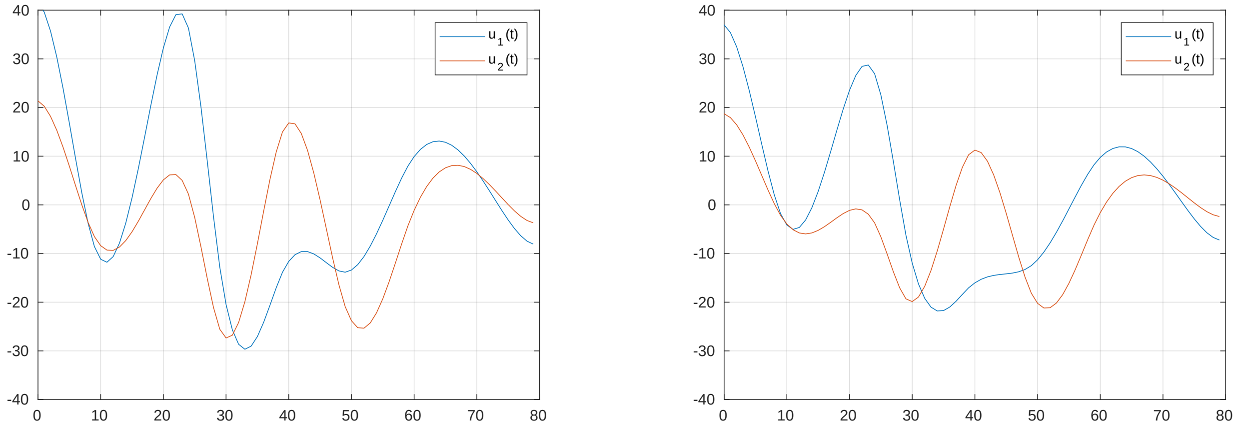



(g) $\lambda = 10^3$

**Figure 1:** Positions of the robot from $t = 0$ to $t = T$ for the different values of $\lambda$ with the $\ell_2^2$ regularizer.

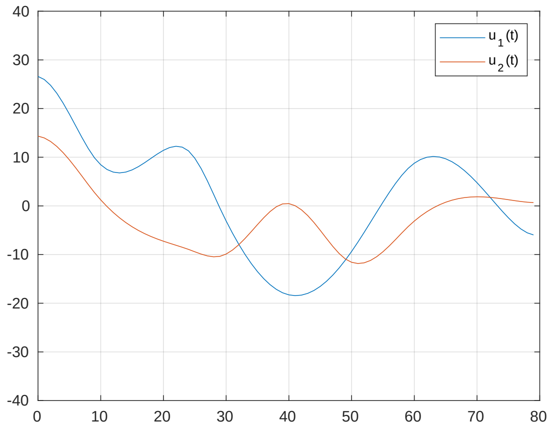
## b)

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 $\lambda$.

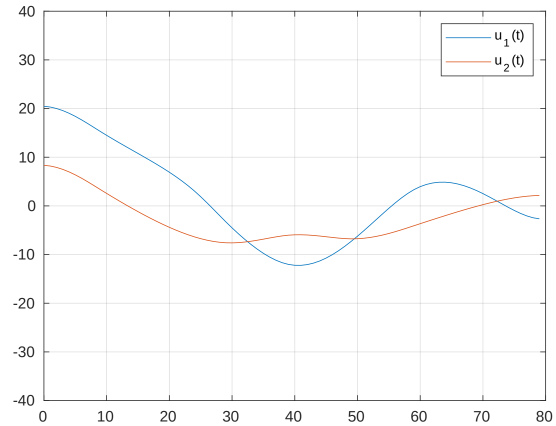



(a) $\lambda = 10^{-3}$

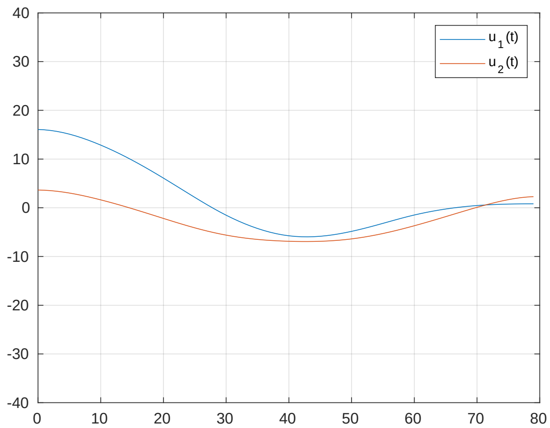
(b) $\lambda = 10^{-2}$



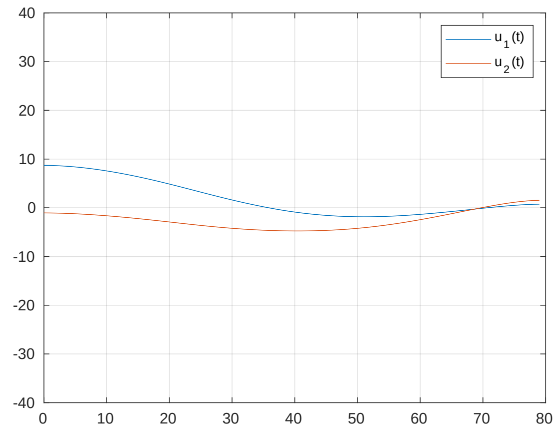

(c) $\lambda = 10^{-1}$



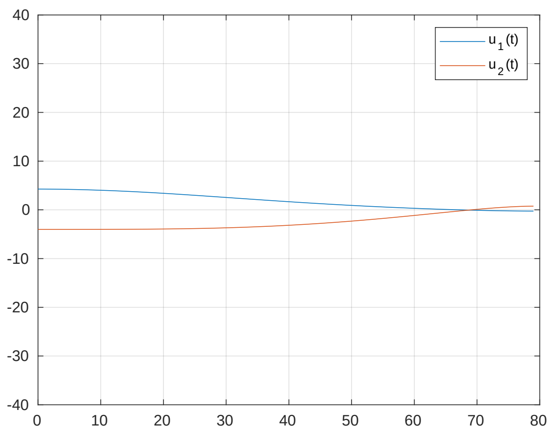

(d) $\lambda = 10^{0}$




(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 $\lambda$, with the $\ell_2^2$ 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 $\lambda$.

| $\lambda$ | *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 $\lambda$, with the $\ell_2^2$ 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 $\lambda$.

| $\lambda$ | *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 $\lambda$, with the $\ell_2^2$ regularizer.

```
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;

```

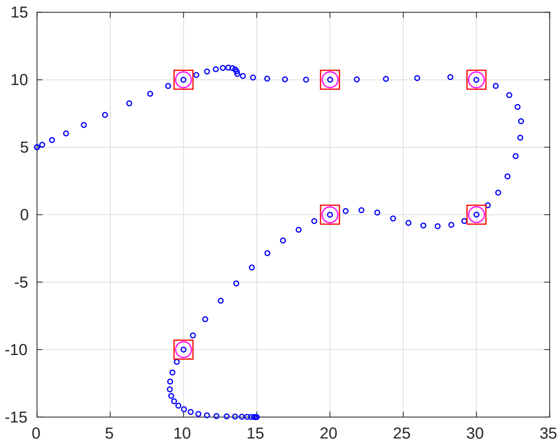
```
        for t = 1:T
            norm( u(:,t)) <= Umax;
        end

        for t = 1:T
            x(:, t+1) == A * x(:, t) + B * u(:, t);
        end
    cvx_end;

    % plot the results
    plot_graphs(x, u, tau+1, w);

    % save plots
    str = num2str(lambda(L));
    save_str = strrep(str,'.',',');
    saveas(figure(1), strcat('Figures/task1/lambda_',save_str , '_position.pdf'));
    saveas(figure(2), strcat('Figures/task1/lambda_', save_str, '_control.pdf'));

    % changes in control signal
    counter = 0;
    for t=2:1:T
        if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
            counter = counter + 1;
        end
    end

    %calculation of mean deviation
    m=0;
    for i=1:1:k
        m = m + norm(E * x(:, tau(i)+1) - w(:, i), 2);
    end
    mean_deviation = m/k;

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

    close all;
end
```

**Listing 1:** Script Task 1

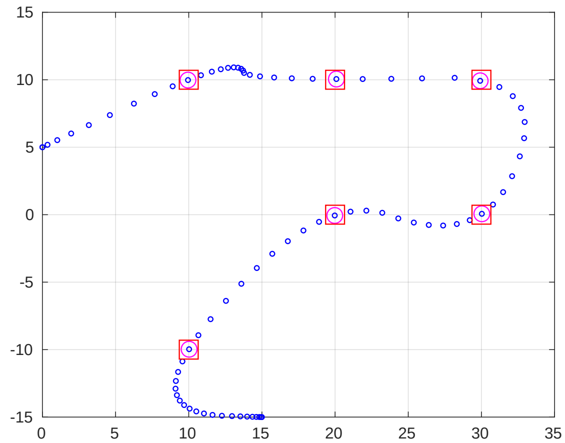
# Task 2

## a)

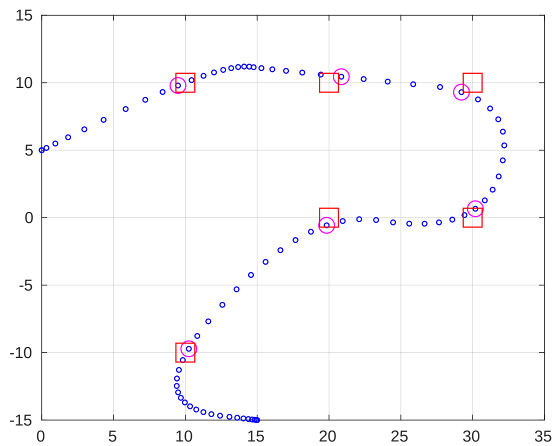
In Figure 3 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times $\tau_k$, for the different values of $\lambda$ parameter, when the cost function uses $\ell_2$ regularizer.



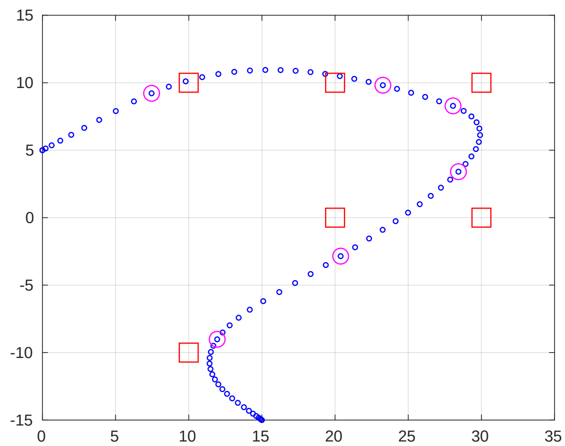

(a) $\lambda = 10^{-3}$



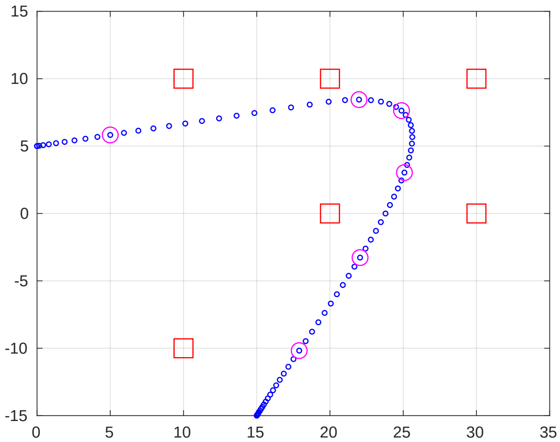

(b) $\lambda = 10^{-2}$



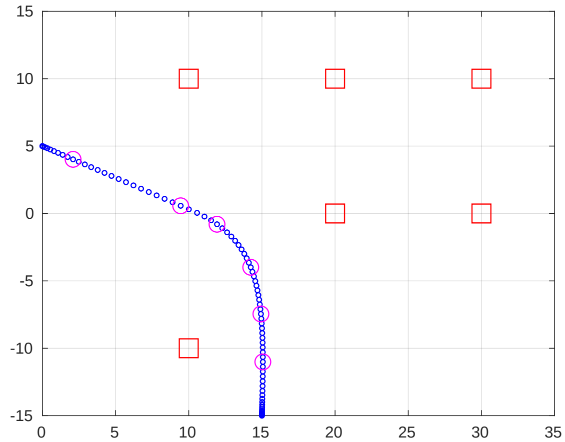

(c) $\lambda = 10^{-1}$



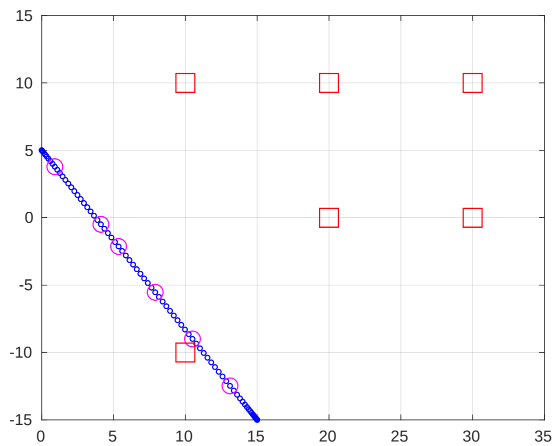

(d) $\lambda = 10^{0}$




(e) $\lambda = 10^1$




(f) $\lambda = 10^2$

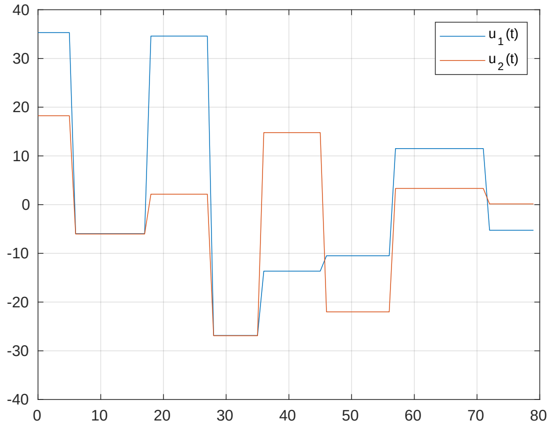



(g) $\lambda = 10^3$

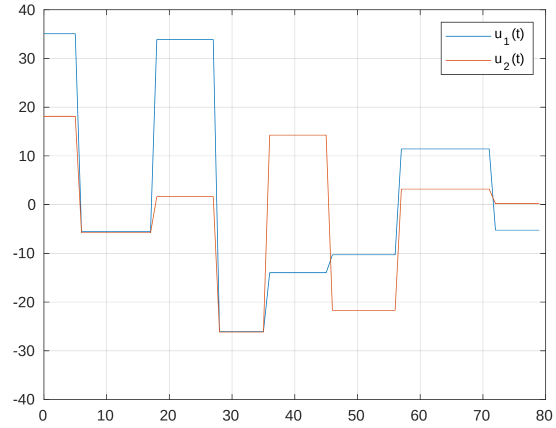
**Figure 3:** Positions of the robot from $t = 0$ to $t = T$ for the different values of $\lambda$, with the $\ell_2$ regularizer.

## b)

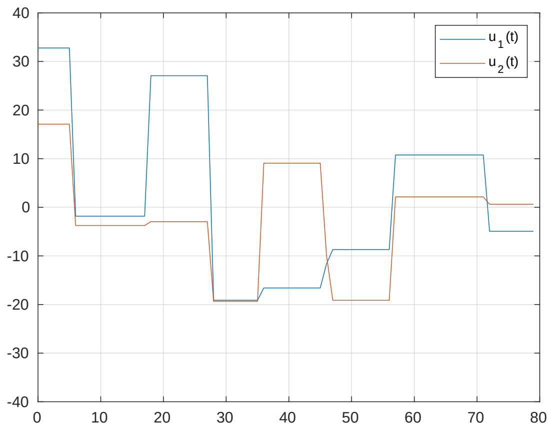
In Figure 4 are represented the plots of the optimal control signal $u(t)$, from $t = 0$ to $t = T - 1$, for the different values of $\lambda$.



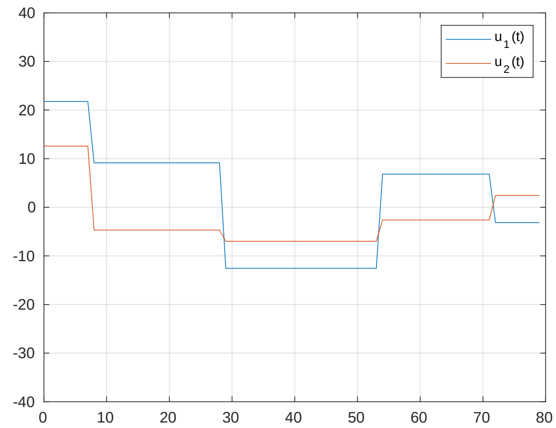

(a) $\lambda = 10^{-3}$



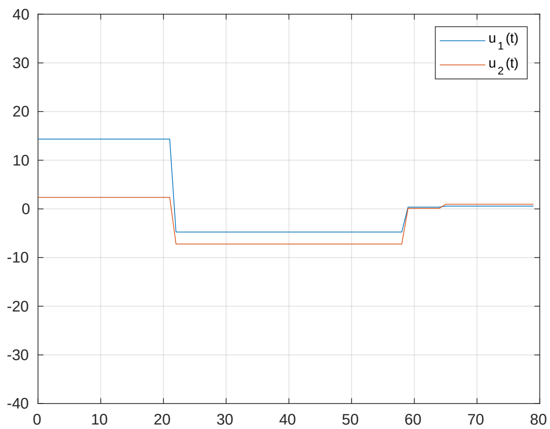

(b) $\lambda = 10^{-2}$



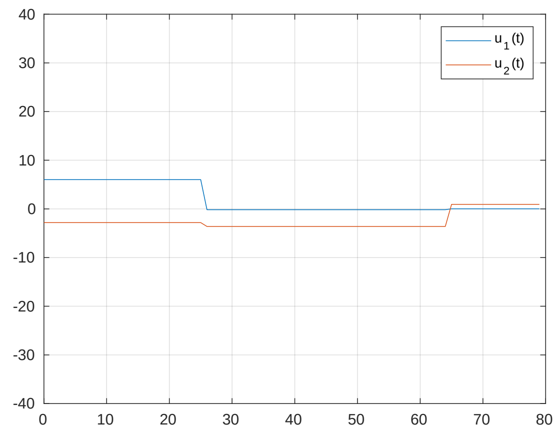

(c) $\lambda = 10^{-1}$



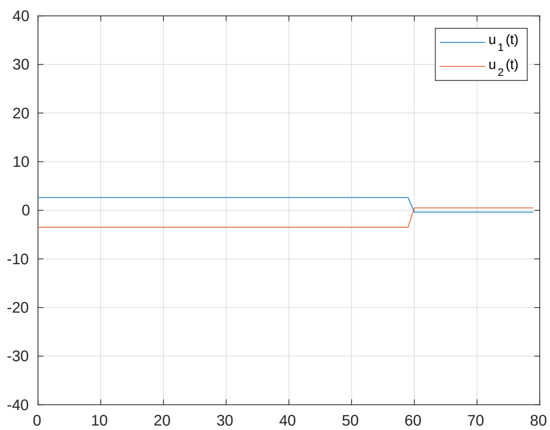

(d) $\lambda = 10^{0}$




(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 $\lambda$, with the $\ell_2$ 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 $\lambda$.

| $\lambda$ | *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 $\lambda$, with the $\ell_2$ regularizer.

## d)

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

| $\lambda$ | *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 $\lambda$, with the $\ell_2$ regularizer.

```
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;
```

```
    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 + 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;

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

    for t = 1:T
        x(:,t+1) == A*x(:,t) + B*u(:,t);
    end

cvx_end;

%plot the results
plot_graphs(x, u, tau+1, w);

% save plots
str = num2str(lambda(L));
save_str = strrep(str,'.',',');
saveas(figure(1), strcat('Figures/task2/lambda_', save_str , '_position.pdf'));
saveas(figure(2), strcat('Figures/task2/lambda_', save_str , '_control.pdf'));

%changes in control signal
counter = 0;
for t=2:1:T
   if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
       counter = counter + 1;
   end
end

%calculation of mean deviation
m=0;
for i=1:1:k
    m = m + norm(E * x(:, tau(i)+1) - w(:, i), 2);
end
```

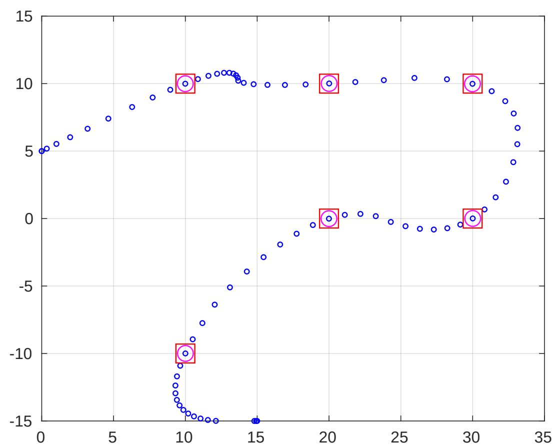
```
    mean_deviation = m/k;

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

    close all
end
```

**Listing 2:** Script Task 2

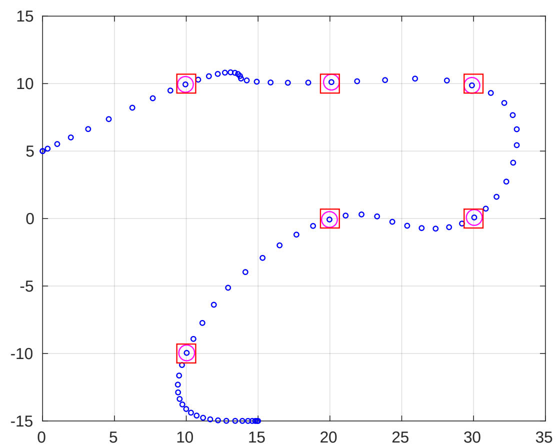
# Task 3

## a)

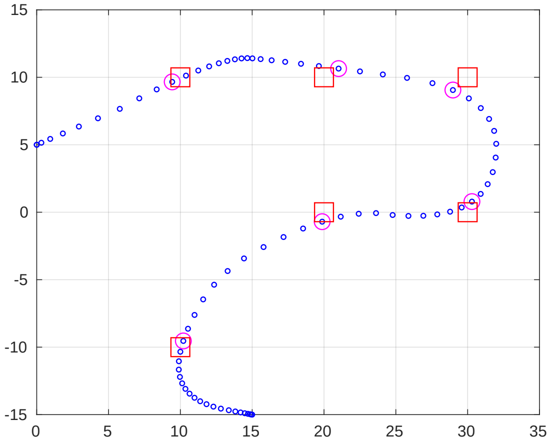
In Figure 5 are represented the plots of the optimal positions of the robot from $t = 0$ to $t = T$, the target positions and the robot positions at the appointed times $\tau_k$, for the different values of $\lambda$ parameter, when the cost function uses $\ell_1$ regularizer.



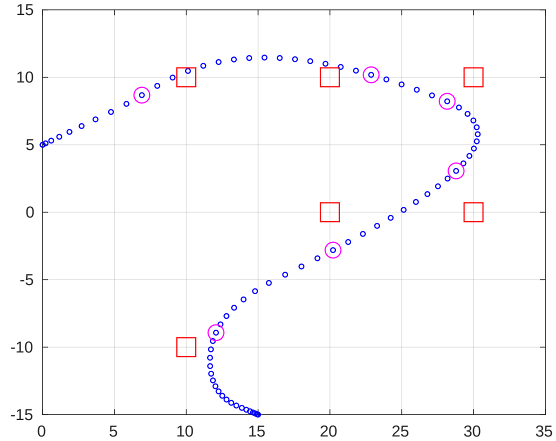

(a) $\lambda = 10^{-3}$



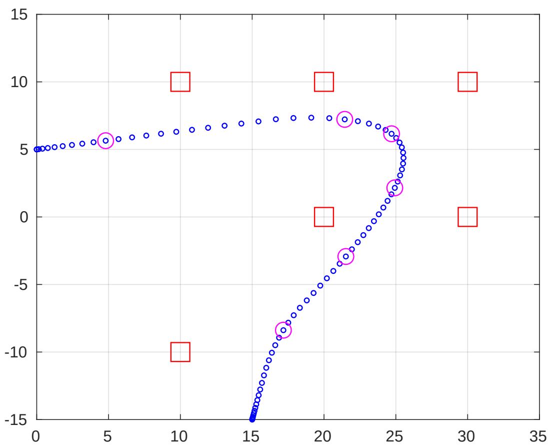

(b) $\lambda = 10^{-2}$



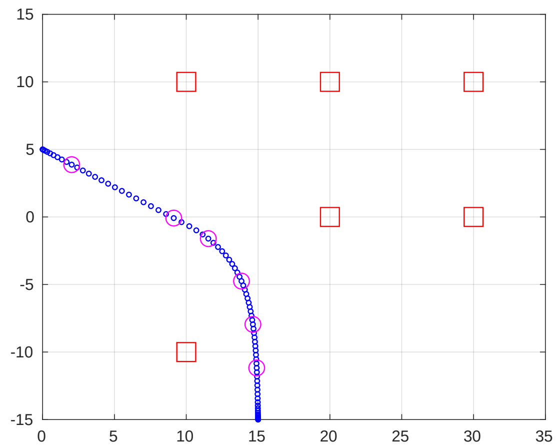

(c) $\lambda = 10^{-1}$



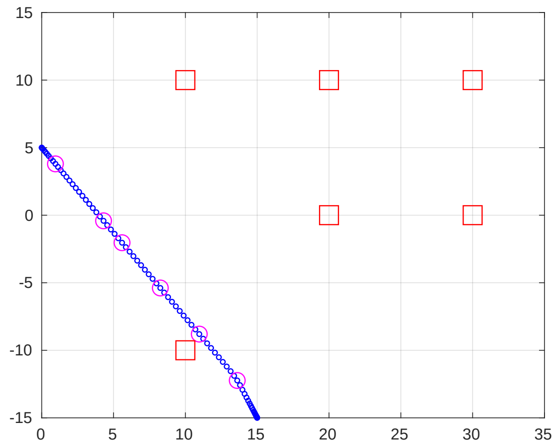

(d) $\lambda = 10^{0}$




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




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

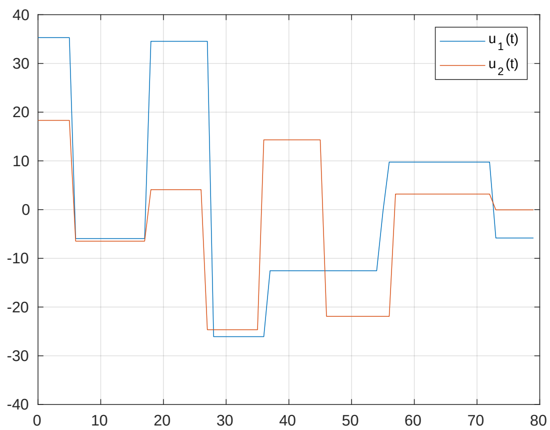



(g) $\lambda = 10^3$

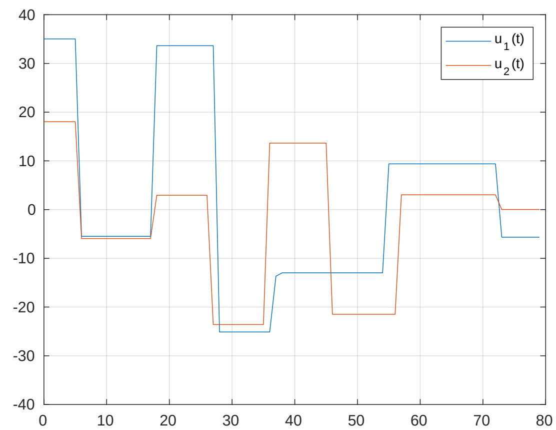
**Figure 5:** Positions of the robot from $t = 0$ to $t = T$ for the different values of $\lambda$, with the $\ell_1$ regularizer.

## b)

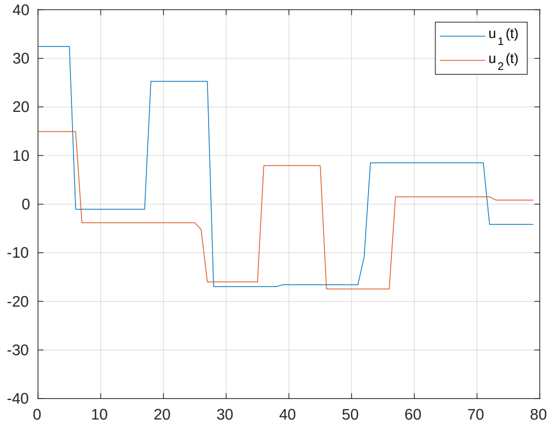
In Figure 6 are represented the plots of the optimal control signal $u(t)$, from $t = 0$ to $t = T - 1$, for the different values of $\lambda$.



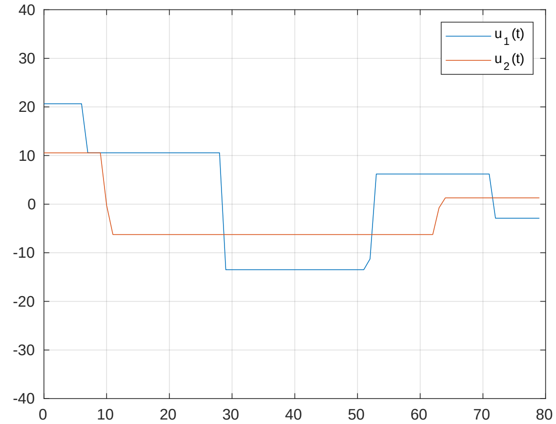

(a) $\lambda = 10^{-3}$



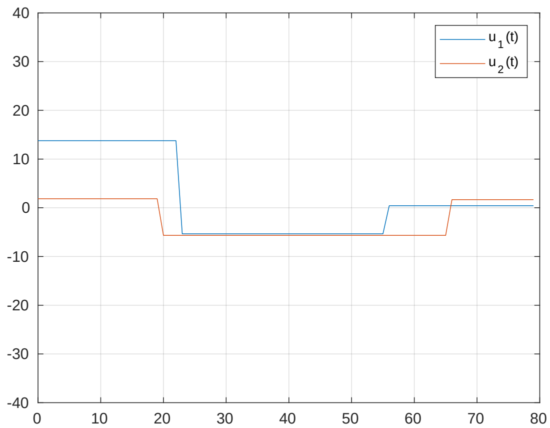

(b) $\lambda = 10^{-2}$



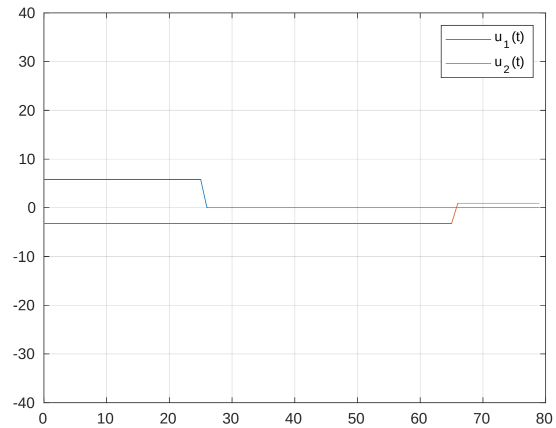

(c) $\lambda = 10^{-1}$



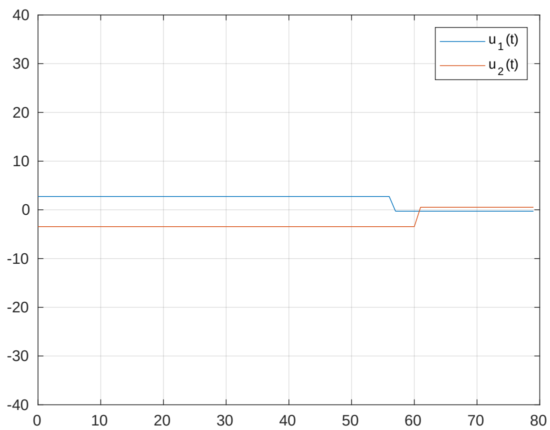

(d) $\lambda = 10^{0}$




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




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




(g) $\lambda = 10^3$

**Figure 6:** Optimal control signal $u(t)$ from $t = 0$ to $t = T - 1$ for the different values of $\lambda$, with the $\ell_1$ 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 $\lambda$.

| $\lambda$ | *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 $\lambda$, with the $\ell_1$ regularizer.

## d)

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

| $\lambda$ | *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 $\lambda$, with the $\ell_1$ regularizer.

```
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;
        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 + norm(u(:, i)-u(:, i-1), 1);
        end

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

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

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

```

```
        for t = 1:T
            x(:,t+1) == A*x(:,t) + B*u(:,t);
        end

    cvx_end;

    %plot the results
    plot_graphs(x, u, tau+1, w);

    % save plots
    str = num2str(lambda(L));
    save_str = strrep(str,'.',',');
    saveas(figure(1), strcat('Figures/task3/lambda_', save_str , '_position.pdf'));
    saveas(figure(2), strcat('Figures/task3/lambda_', save_str , '_control.pdf'));

    %changes in control signal
    counter = 0;
    for t=2:1:T
      if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
          counter = counter + 1;
      end
    end

    %calculation of mean deviation
    m=0;
    for i=1:1:k
        m = m + norm(E * x(:, tau(i)+1) - w(:, i), 2);
    end
    mean_deviation = m/k;

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

    close all
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 $\lambda$ 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 $\lambda$ 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 $\lambda$ 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 $\lambda$, 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 $\ell_2^2$ 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 $\ell_2$ and $\ell_1$ 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 $\ell_2^2$ regularizer, the algorithm will never decrease below the given threshold ($10^{-6}$) and never reach 0. However, using the $\ell_2^2$ 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 $\ell_2$ regularizer, the algorithm can minimize the second term until zero (the same happens when using the $\ell_1$ 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 $\ell_1$ 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 $\ell_1$ 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 $\lambda$: the lower it is, the better it will be satisfied. The wish of simpler control (wish 4) is only really satisfied for norms $\ell_2$ and $\ell_1$, and the higher the $\lambda$ 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.

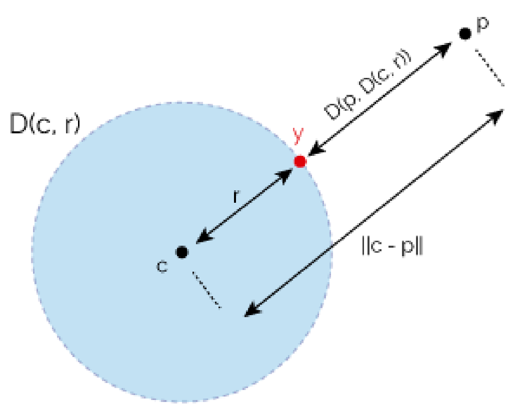



**Figure 7:** Distance of point to the disk, if it is outside it. The point $y$ found by the minimization is highlighted.

# Task 6

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

**a)**

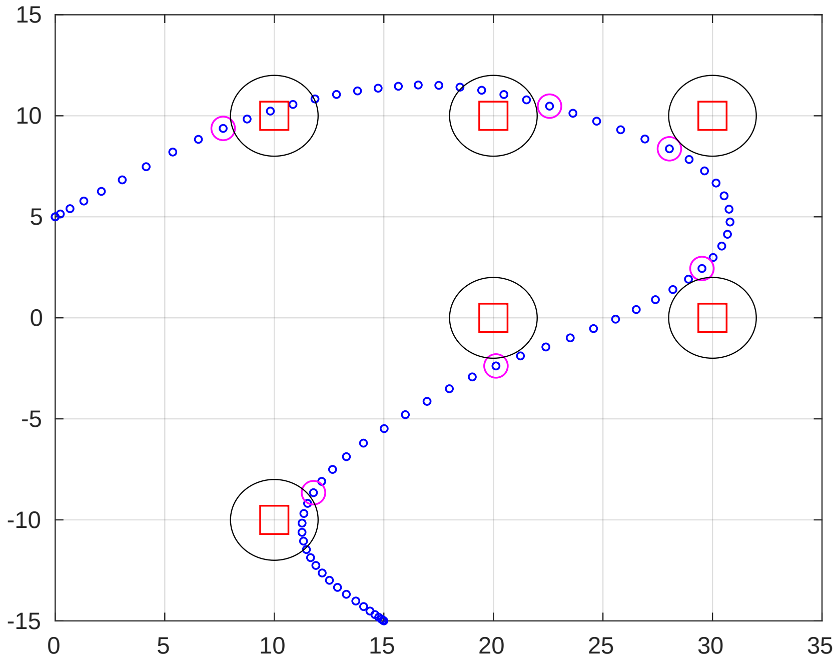



**Figure 8:** Positions of the robot from $t = 0$ to $t = T$, positions at the appointed times $\tau_k$ for $1 \leq k \leq K$ are marked in blue. The waypoints are signaled as red squares and the disk is drawn in black.

**b)**

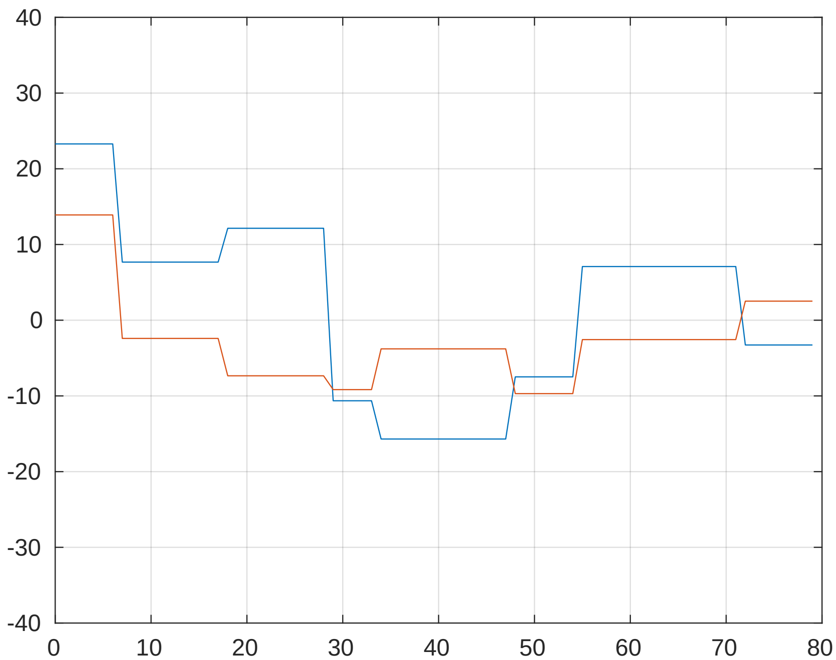



**Figure 9:** Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$.

## 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 ($\tau_k$) 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.

```
clear all;
close all;

radius = 2;

%load the workspace
load('dataB.mat');
```

```

%chosse the value of lambda
lambda1 = 0.1;

% 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

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

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

    f = f_waypoints + lambda1 * f_regularizer;
    minimize( f );

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

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

    for t = 1:T
        x(:, t+1) == A * x(:, t) + B * u(:, t);
    end

cvx_end;

%plot the results
plot_graphs_disks(x, u, tau+1, c, radius);


%changes in control signal
counter = 0;
for t=2:1:T
   if norm(u(:,t)-u(:,t-1), 2) > power(10,-6)
       counter = counter + 1;
   end
end

%calculation of mean deviation
```

```
m = 0;
m_ = 0;
for i=1:1:k
    m = m + norm((E * x(:, tau(i))) - c(:, i));
    m_ = m_ + pos(norm((E * x(:, tau(i))) - c(:, i) ) - radius);
end
mean_deviation = m/k;
mean_desviation_disk = m_/k;

disp(counter);
disp(mean_deviation);
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.

```
clear all;
close all;

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

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

    f = 0;
    minimize( f );

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

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

    for i=1:1:k
        E * x(:, tau(i) + 1) == w(:, i);
    end
```

```

    for t = 1:T
        x(:, t+1) == A * x(:, t) + B * u(:, t);
    end

cvx_end;


captured=0;
for i=1:1:k
    dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);

    %disp(dw);
    if dw <= 10^-6
        captured = captured + 1;
    end
end

disp(captured);


plot_graphs(x, u, tau+1, w);
```

**Listing 5:** Script Task 7

# Task 8

Function $\phi$ 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.

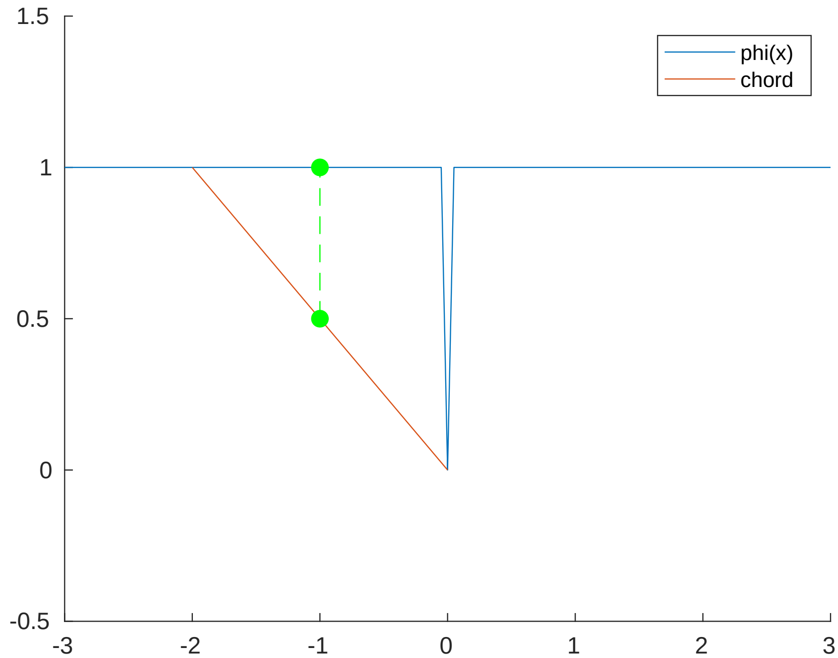



**Figure 10:** Graphical proof hat $\phi$ is nonconvex

By definition, function $\phi$ 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 $\alpha = 0.5$:

$$\phi(-1, 0) \nleq 0.5\phi(-2, 0) + 0.5\phi(0, 0)$$
$$1 \nleq 0.5$$

So we can conclude the function $\phi$ is nonconvex.

```
clear all;
close all;

x = -3:0.05:3;
phi = x;

for i=1:1:length(x)
   if phi(i) ≠ 0
       phi(i) = 1;
   end
end

x_hat = -2:0.05:0;
```

```
chord = −0.5 * x_hat;

figure(1);
hold on;

f = plot(x, phi);
g = plot(x_hat, chord);
line([−1 −1], [0.5 1], 'Color', 'green', 'LineStyle', '−−');
scatter([−1 −1], [0.5 1], 'MarkerEdgeColor', 'green', 'MarkerFaceColor', '
    ↪ green', 'LineWidth', 1.5);

axis([−3 3 −0.5 1.5]);
xlabel('x');
legend([f g], 'phi(x)', 'chord');

saveas(figure(1), 'Figures/task8/nonconvex.pdf');
```

**Listing 6:** Script Task 8

# Task 9

## a)

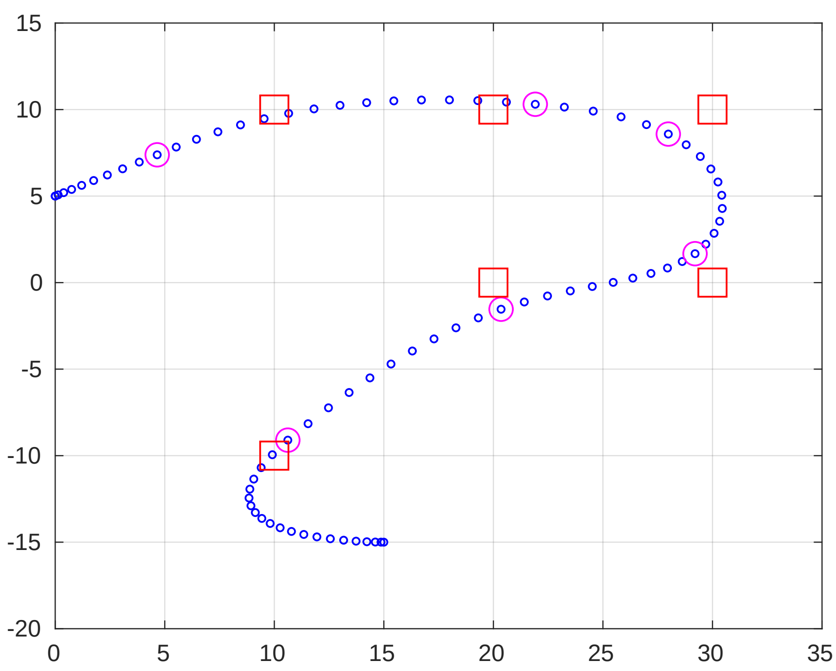



**Figure 11:** Positions of the robot from $t = 0$ to $t = T$ (blue dots) for the new formulation with the $\ell_2^2$ formulation. Waypoints $\omega_k$ are marked as red square and positions at the appointed times $\tau_k$ are marked with red circles.

## b)

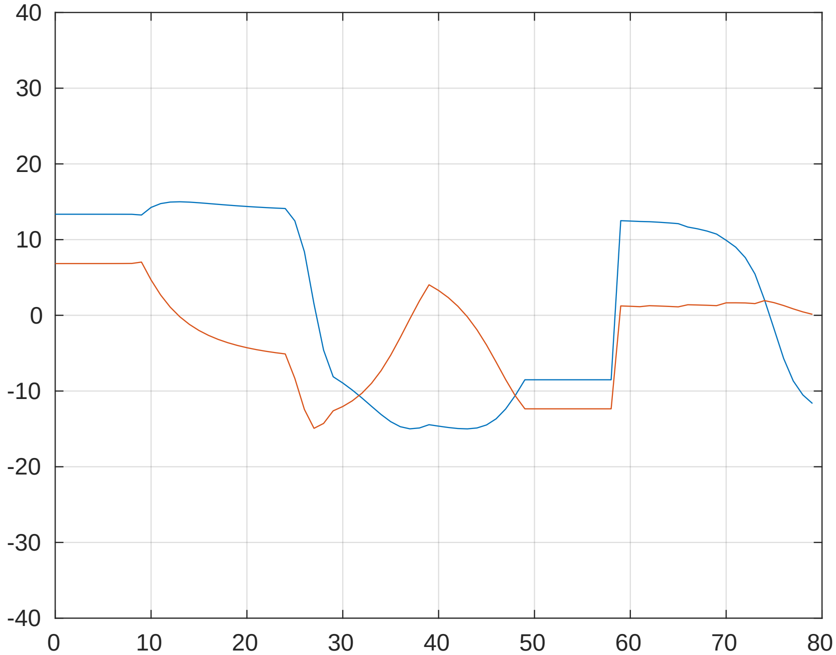



**Figure 12:** Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$, for the formulation with the $\ell_2^2$ norm.

## c)

The robot did not capture any waypoint.

```
clear all;
close all;

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

% 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

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

    minimize( f );
```

```

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

    for t = 1:T
        norm( u(:,t)) ≤ Umax;
    end

    for t = 1:T
        x(:, t+1) == A ∗ x(:, t) + B ∗ u(:, t);
    end

cvx_end;

% plot postions and control signals
plot_graphs(x, u, tau+1, w);

% save plots
saveas(figure(1), 'Figures/task9/position.pdf');
saveas(figure(2), 'Figures/task9/control.pdf');

captured=0;
for i=1:1:k
    dw = norm(E ∗ x(:, tau(i)+1) − w(:, i), 2);

    %disp(dw);
    if dw ≤ 10^−6
        captured = captured + 1;
    end
end

str1 = ['Number of waypoints capture by the robot = ', num2str(captured)];
disp(str1);

close all;
```

**Listing 7:** Script Task 9

# Task 10

## a)

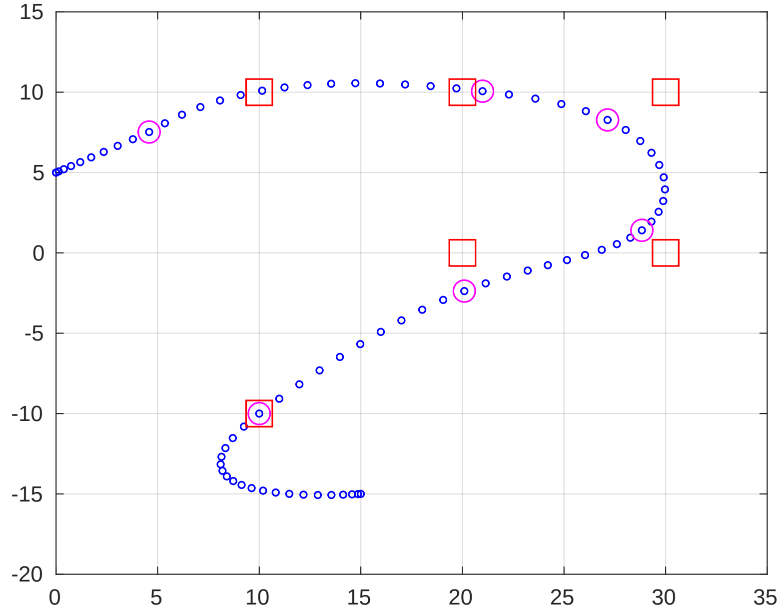



**Figure 13:** Positions of the robot from $t = 0$ to $t = T$ (blue dots) for the new formulation with the $\ell_2$ formulation. Waypoints $\omega_k$ are marked as red square and positions at the appointed times $\tau_k$ are marked with red circles.

## b)

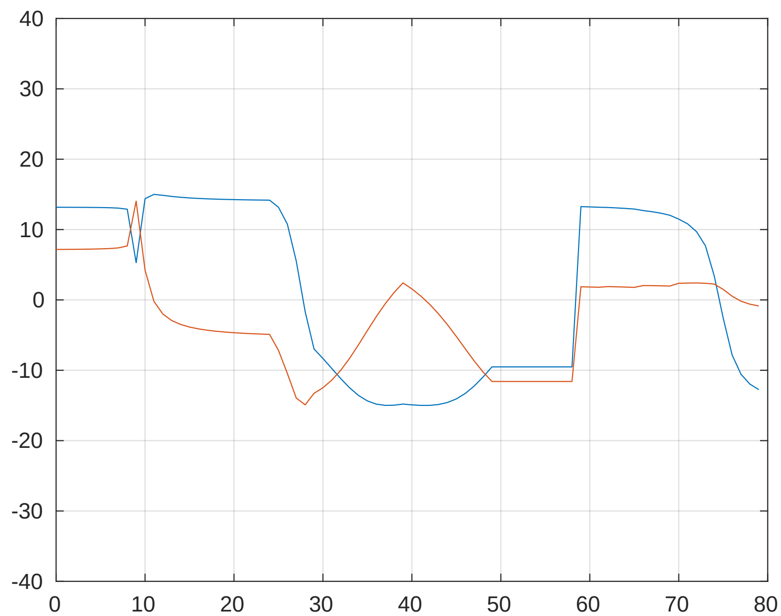



**Figure 14:** Optimal control signal $u(t)$, where $u(t) = (u_1(t), u_2(t))$, from $t = 0$ to $t = T - 1$, for the formulation with the $\ell_2$ norm.

## c)

The robot captured 1 waypoint.

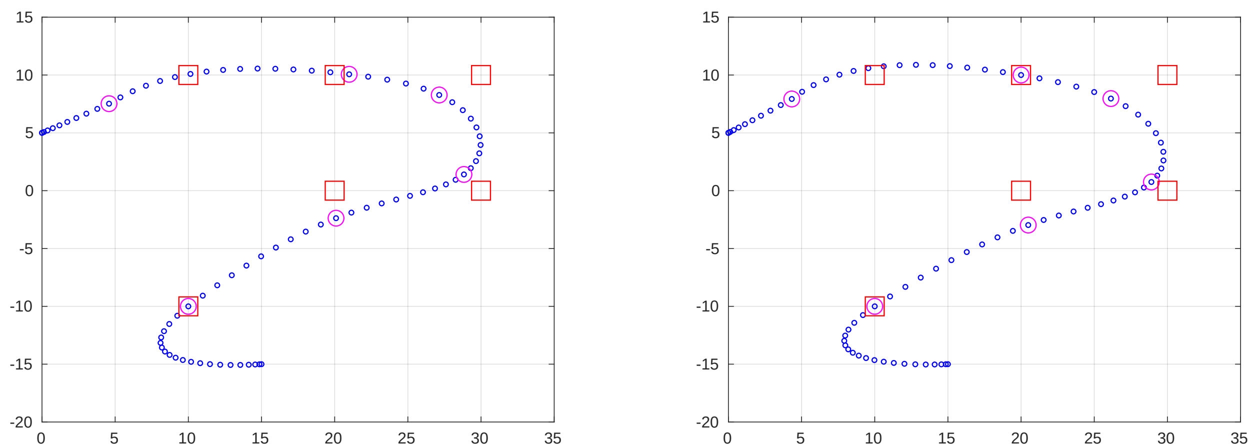
```
clear all;
close all;

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

% 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

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

    minimize( f );

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

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

    for t = 1:T
        x(:, t+1) == A * x(:, t) + B * u(:, t);
    end

cvx_end;

% plot postions and control signals
plot_graphs(x, u, tau+1, w);

% save plots
saveas(figure(1), 'Figures/task10/position.pdf');
saveas(figure(2), 'Figures/task10/control.pdf');

captured=0;
for i=1:1:k
    dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);

    %disp(dw);
    if dw <= 10^-6
```

```
        captured = captured + 1;
    end
end

str1 = ['Number of waypoints capture by the robot = ', num2str(captured)];
disp(str1);

close all;
```

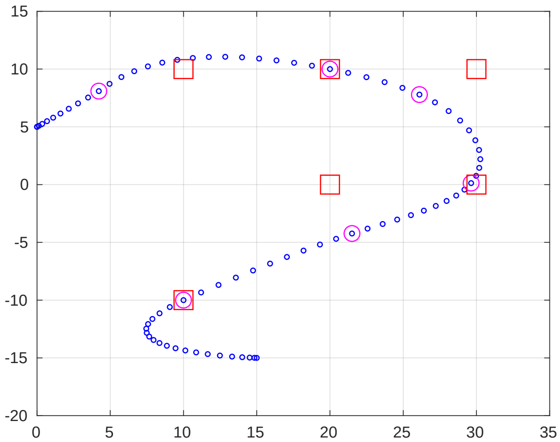
**Listing 8:** Script Task 10

# Task 11

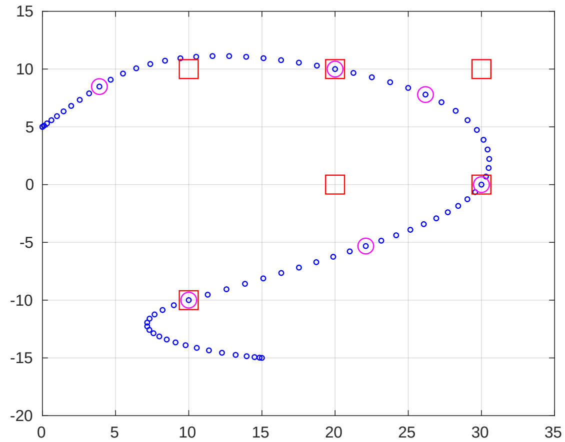
## a)



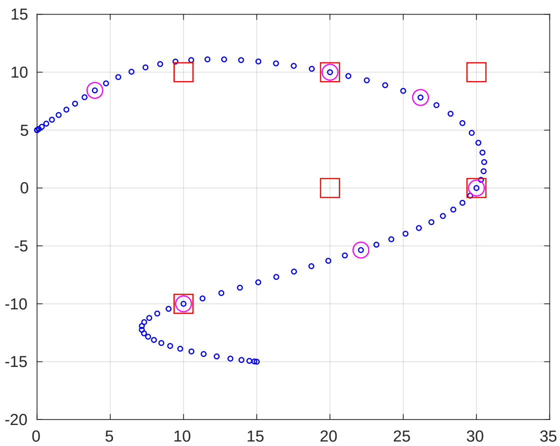

(a) $m = 0$ (b) $m = 1$



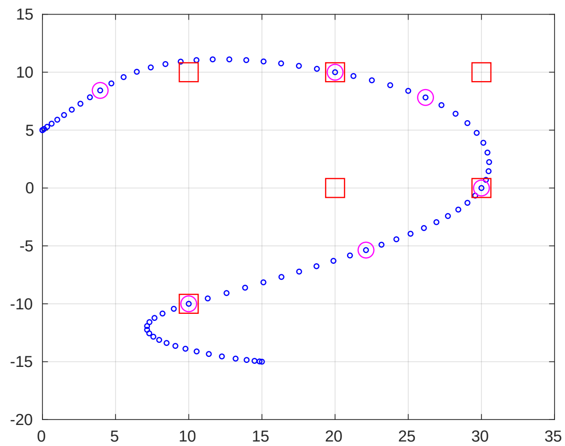

(c) $m = 2$



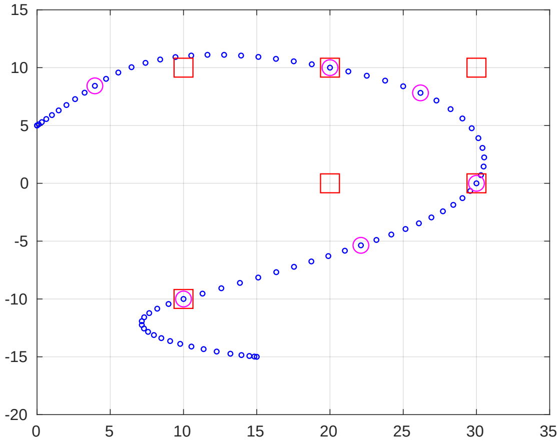

(d) $m = 3$



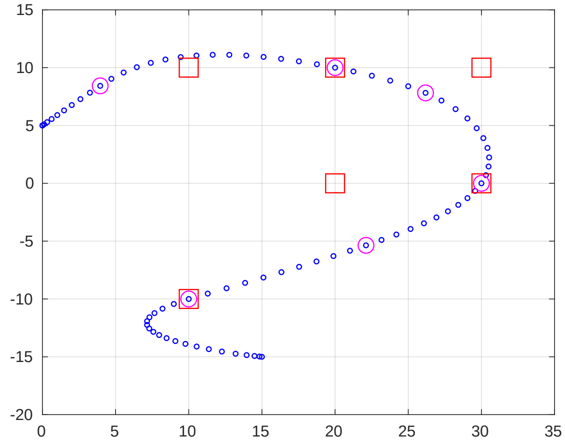

(e) $m = 4$



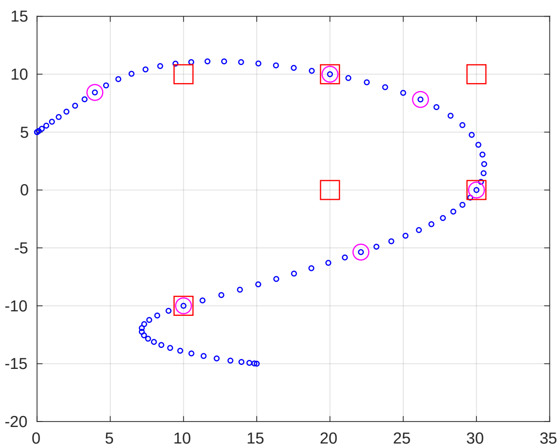

(f) $m = 5$



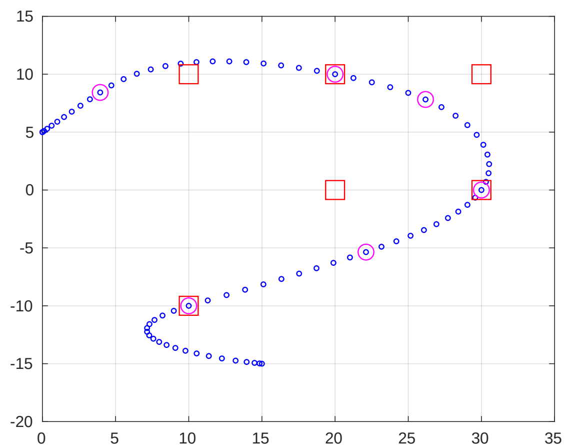

(g) $m = 6$




(h) $m = 7$



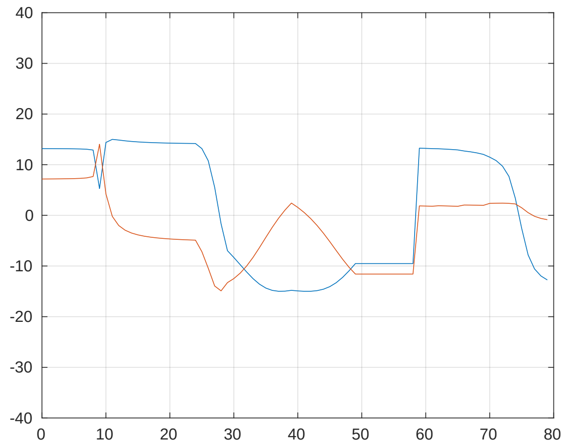

(i) $m = 8$



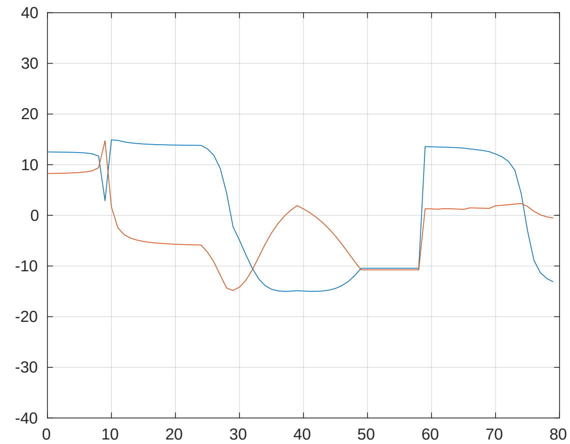

(j) $m = 9$

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

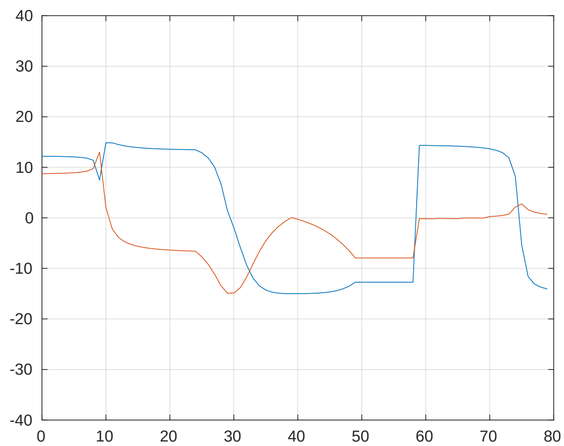
**b)**



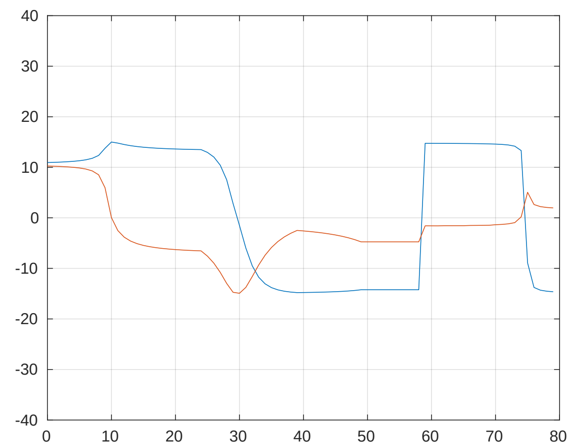
(a) $m = 0$



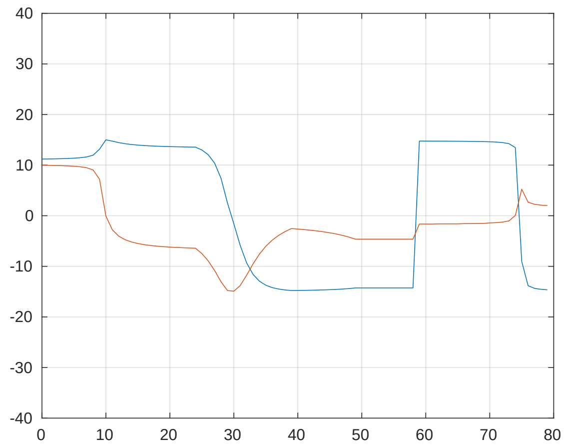
(b) $m = 1$



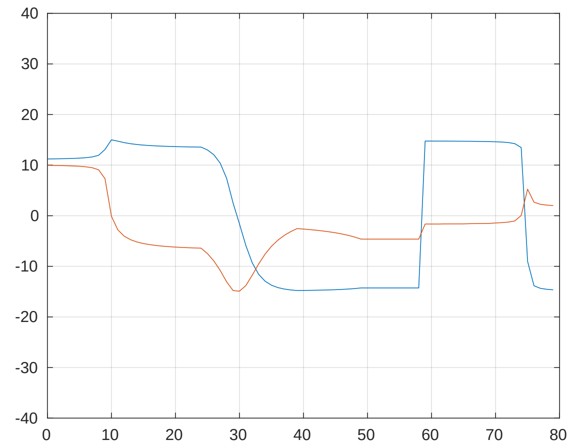
(c) $m = 2$



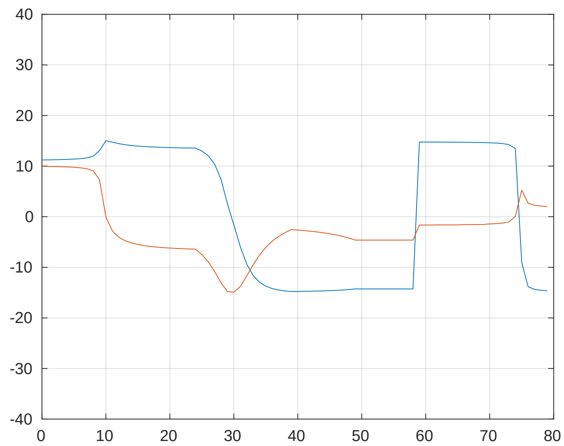
(d) $m = 3$



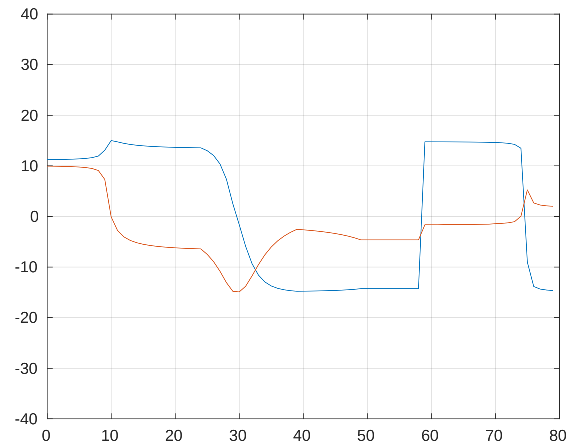

(e) $m = 4$



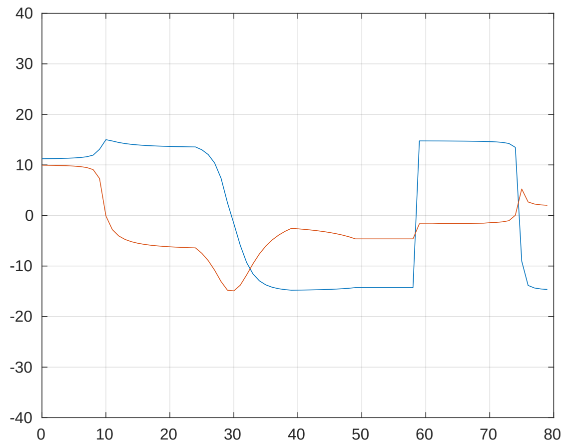

(f) $m = 5$



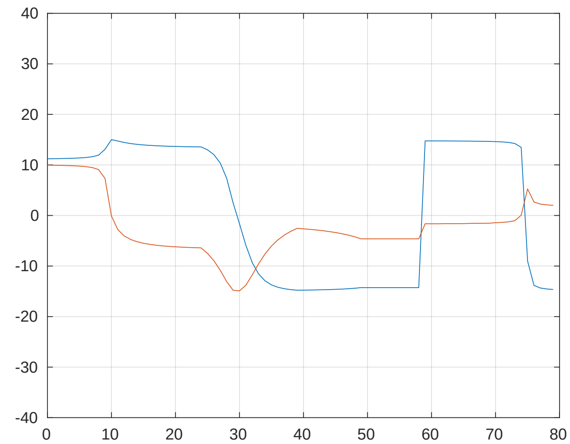

(g) $m = 6$




(h) $m = 7$




(i) $m = 8$




(j) $m = 9$

**Figure 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.

```
clear all;
close all;

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

```

```
weights = ones(k, 1);
epson = 10^-6;
iter = 10;

for m=1:1:iter
    % solve iterative 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

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

        minimize( f );

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

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

        for t = 1:T
            x(:, t+1) == A * x(:, t) + B * u(:, t);
        end
    cvx_end;

    % plot postions and control signals
    plot_graphs(x, u, tau+1, w);

    % save plots
    saveas(figure(1), strcat('Figures/task11/iter_', num2str(m - 1), '_position.pdf'));
    saveas(figure(2), strcat('Figures/task11/iter_', num2str(m - 1), '_control.pdf'));

    % calc new weights
    for i=1:1:k
       weights(i) = 1 / (norm(E * x(:, tau(i)+1) - w(:, i), 2) + epson);
    end

    %disp(weights);

    captured=0;
    for i=1:1:k
        dw = norm(E * x(:, tau(i)+1) - w(:, i), 2);

```

```
        if dw ≤ 10^−6
            captured = captured + 1;
        end
    end


    str1 = ['Number of waypoints capture on iteration ', num2str(m − 1), '
       ↪ = ', num2str(captured)];
    disp(str1);

    close all;
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:

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

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 $\ell_2$ norm formulation one waypoint is captured. The optimizer is minimizing the sum of distances to the waypoints, and replacing the $\ell_2^2$ with the $\ell_2$ 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 $\ell_2$ 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.

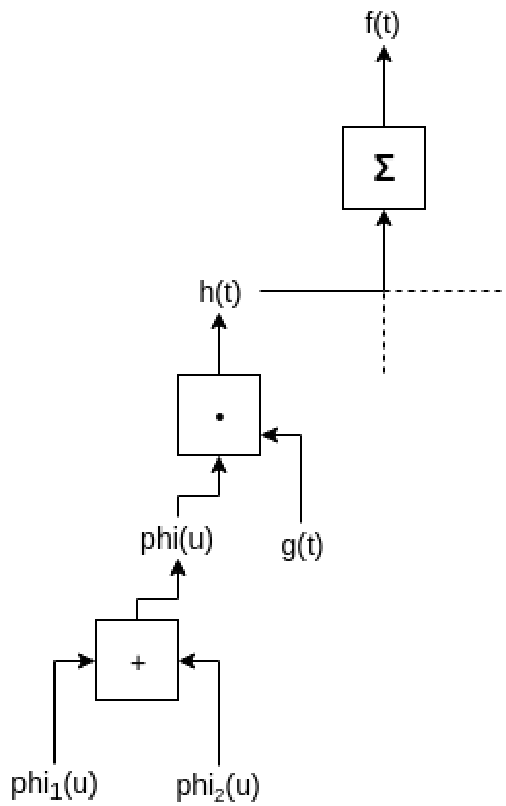



**Figure 17:** Recursion tree proving function's $f(t)$ convexity.

## Task 2

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

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

**Listing 10:** Script to calculate function value

```
function [ result ] = gradient_f_hat( t, X_hat, Y, k )

result = 1/k * X_hat * (exp(X_hat'*t)./(1 + exp(X_hat'*t)) - Y');

end
```

**Listing 11:** Script to calculate gradient of function

```
function [ result ] = hessian_f_hat( t, X_hat, Y, k )

result = 1/k * X_hat * diag(exp(X_hat'*t)./((1 + exp(X_hat'*t)).^2)) * X_hat';

end
```

**Listing 12:** Script to calculate hessian of function

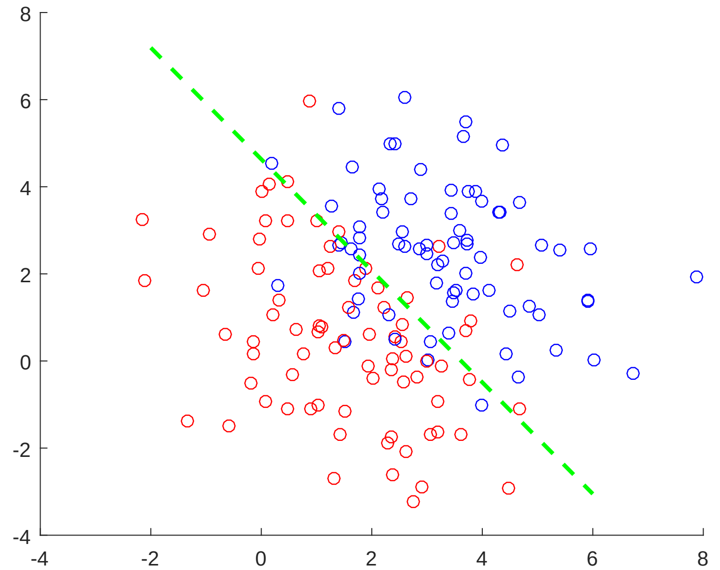



**Figure 18:** Gradient method result for data1.m.

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

The total number of iterations was: 1125.

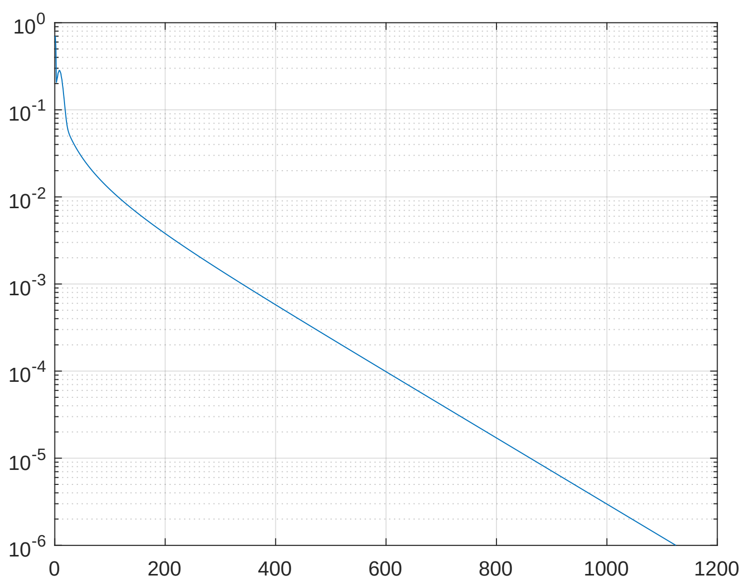



**Figure 19:** Norm of the gradient along iterations of the gradient method for data1.m.

# Task 3

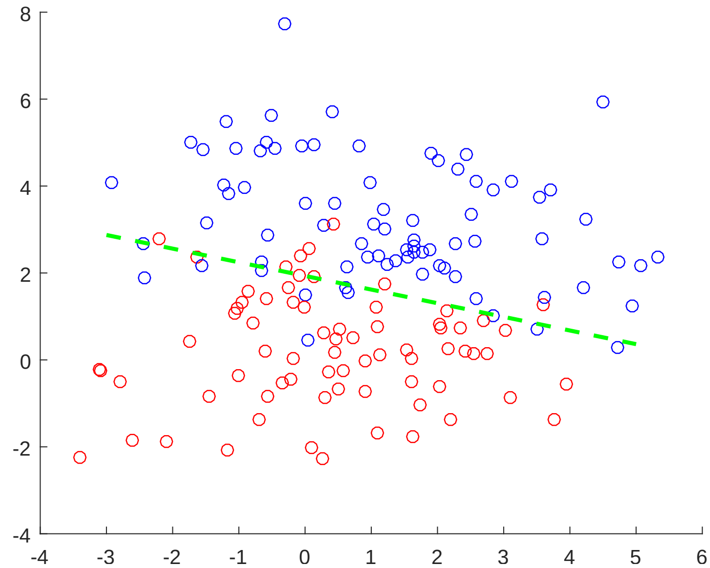



**Figure 20:** Gradient method result for data2.m.

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

The total number of iterations was: 1362.

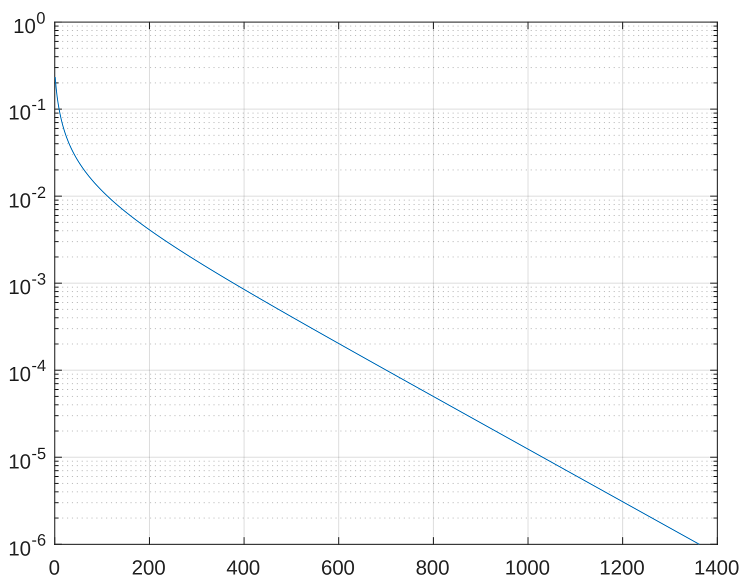



**Figure 21:** Norm of the gradient along iterations of the gradient method for data2.m.

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

```
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
semilogy(gradients);
grid on;

iter = length(gradients)

s0 = t(1)
s1 = t(2)
r = t(3)


figure;
%Plot data points
for i = 1:150
    if Y(i) == 0
        scatter(X(1, i), X(2, i), [], 'red');
        hold on
    else
        scatter(X(1, i), X(2, i), [], 'blue');
        hold on
    end
end

%Plot resulting line
xResult = -3:0.001:5;
yResult = (r/s1) - (s0/s1)*xResult;
plot(xResult, yResult,['--', 'g'], 'LineWidth', 2)
```

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

# Task 4

## data3.m

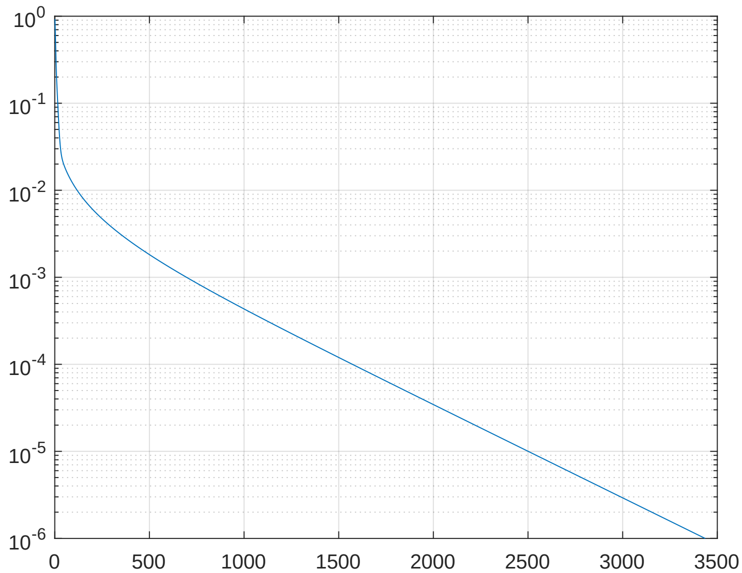



**Figure 22:** Norm of the gradient for the gradient method along iterations for data3.m.

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

$r = -4.7984$

The total number of iterations was: 3436.

## dataset4.m

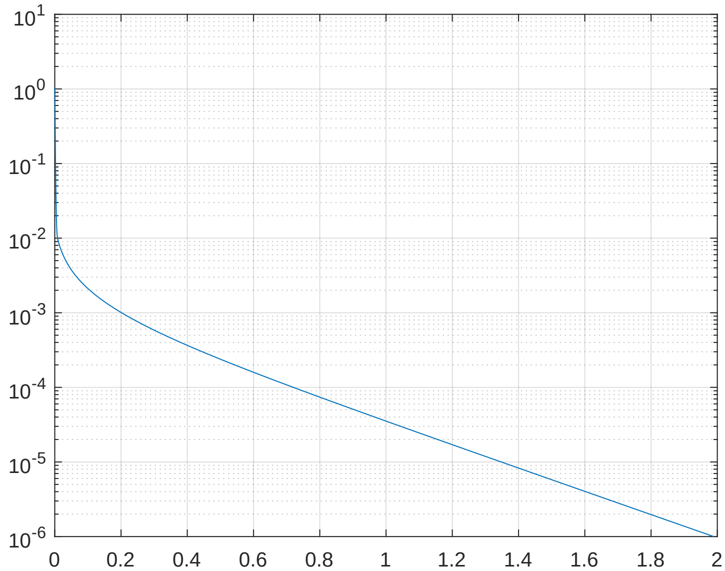



**Figure 23:** Norm of the gradient for the gradient method along iterations for dataset4.m. The horizontal axis is multiplied by $10^4$.

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

$$r = 7.6701\,,$$

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);
```

```

%Initial point for gradient descent
t0 = [s0 r0]'

%Backtracking parameters
alpha0 = 1;
y = 10^(-4);
beta = 0.5;

%Transformation of X
X_hat = [X; -ones(length(X), 1)'];

%Algorithm - Gradient Descent
t = t0;
alpha = alpha0;
gradients = [];
while norm(gradient_f_hat(t, X_hat, Y, k)) ≥ epslon
    d = -gradient_f_hat(t, X_hat, Y, k);
    alpha = alpha0;
    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))
        alpha = beta .* alpha;
    end
    t = t + (alpha .* d);
    %f_hat(t, X_hat, Y, k)
    gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
end

%plot figure with logarithmic y-axis
figure;
semilogy(gradients);
grid on;

iter = length(gradients)

s = t(1:n)
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)

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

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

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

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

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

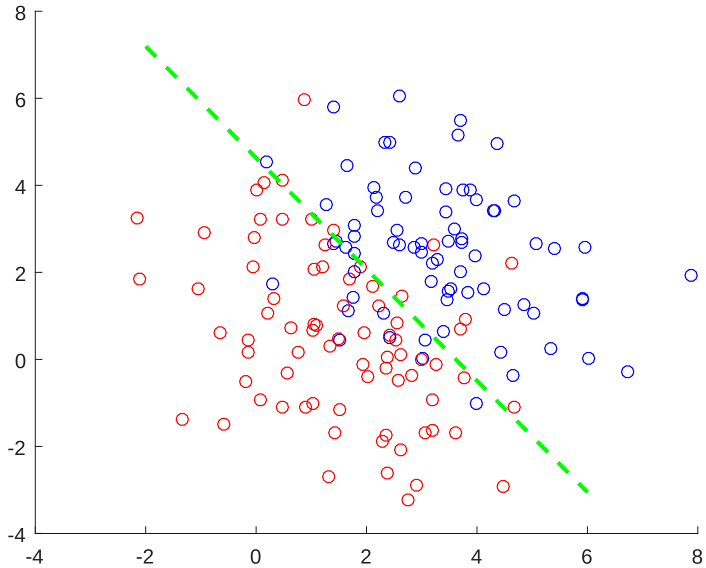



**Figure 24:** Newton method result for data1.m

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

Total number of iterations: 8.

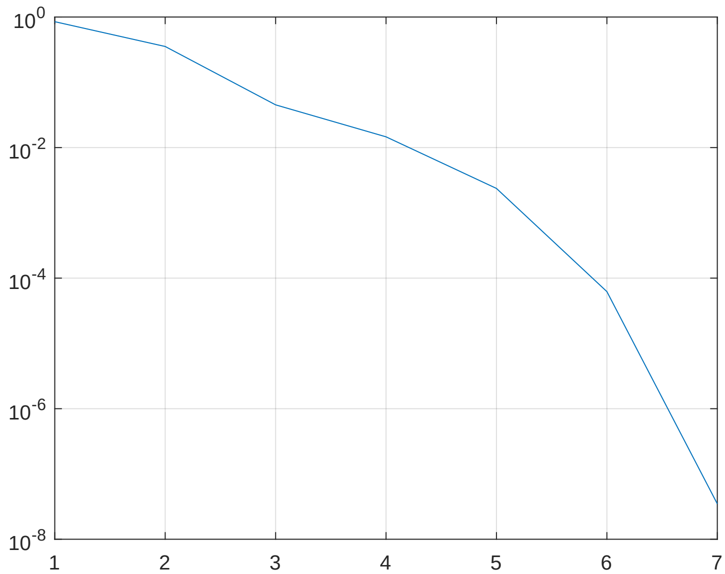



**Figure 25:** Norm of the gradient along iterations for data1.m

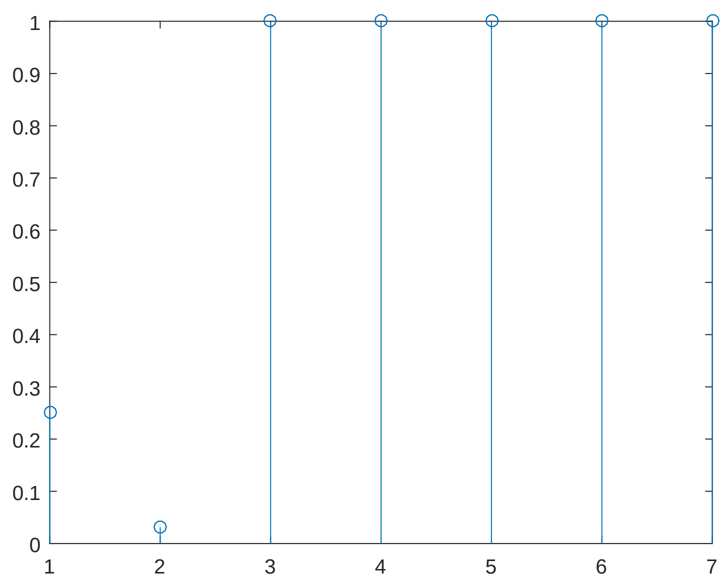



**Figure 26:** Value of alpha along iterations for data1.m

## data2.m

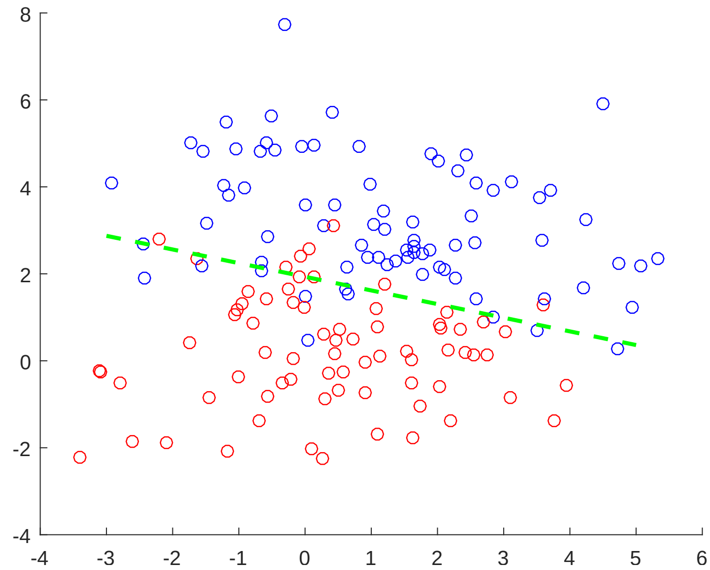



**Figure 27:** Newton method result for data2.m

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

Total number of iterations: 8.

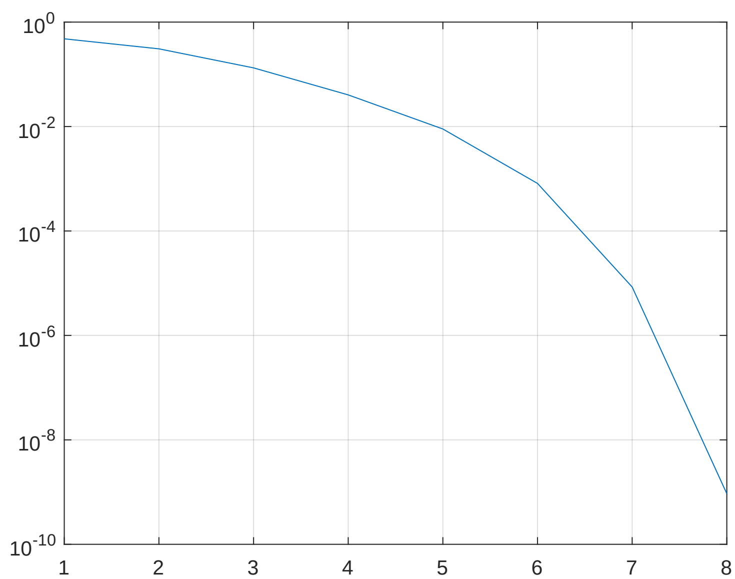



**Figure 28:** Norm of the gradient along iterations for data2.m.

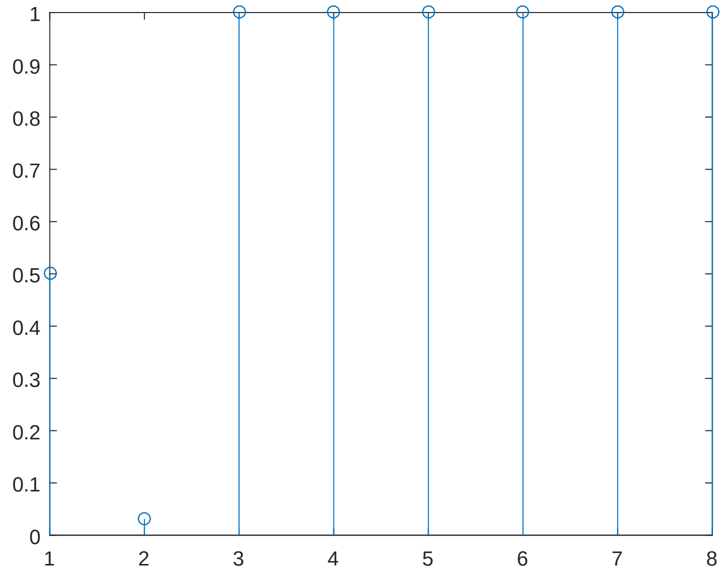



**Figure 29:** Value of alpha along iterations for data2.m.

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):

```
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;
```

```

%Transformation of X
X_hat = [X; -ones(length(X), 1).'];

%Algorithm - Newton Method
t = t0;
alpha = alpha0;
alphas = [];
gradients = [];
while norm(gradient_f_hat(t, X_hat, Y, k)) ≥ epslon
    g = gradient_f_hat(t, X_hat, Y, k);
    d = -(hessian_f_hat(t, X_hat, Y, k)\g);
    alpha = alpha0;
    while f_hat(t + alpha.*d, X_hat, Y, k) ≥ f_hat(t, X_hat, Y, k) + (y.*g'*(alpha.*d))
        alpha = beta .* alpha;
    end
    t = t + (alpha .* d)
    %f_hat(t, X_hat, Y, k)
    gradients = [gradients norm(gradient_f_hat(t, X_hat, Y, k))];
    alphas = [alphas alpha];
end

%plot alpha for each iteration
figure;
stem(alphas);

%plot figure with logarithmic y-axis
figure;
semilogy(gradients);
grid on;

s0 = t(1)
s1 = t(2)
r = -t(3)

figure;
%Plot data points
for i = 1:150
    if Y(i) == 0
        scatter(X(1, i), X(2, i), [], 'red');
        hold on
    else
        scatter(X(1, i), X(2, i), [], 'blue');
        hold on
    end
end

%Plot resulting line
xResult = -3:0.001:5;
yResult = (r/s1) - (s0/s1)*xResult;
```

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

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

## data3.m

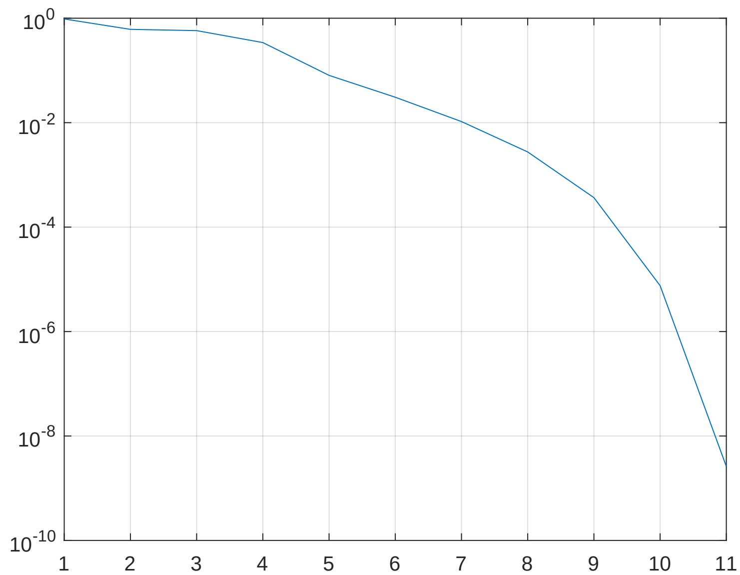



**Figure 30:** Norm of the gradient along iterations for data3.m.

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

$r = 4.7987$

The total number of iterations was: 11.

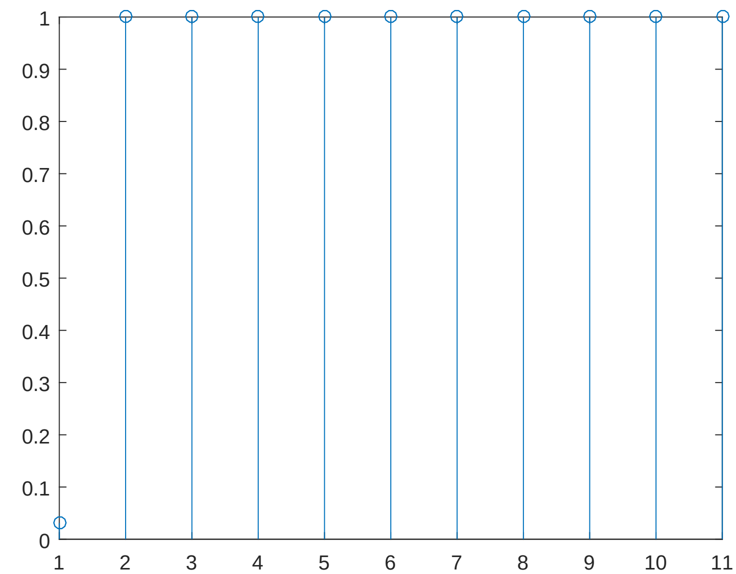



**Figure 31:** Value of alpha along iterations for data3.m

## dataset4.m

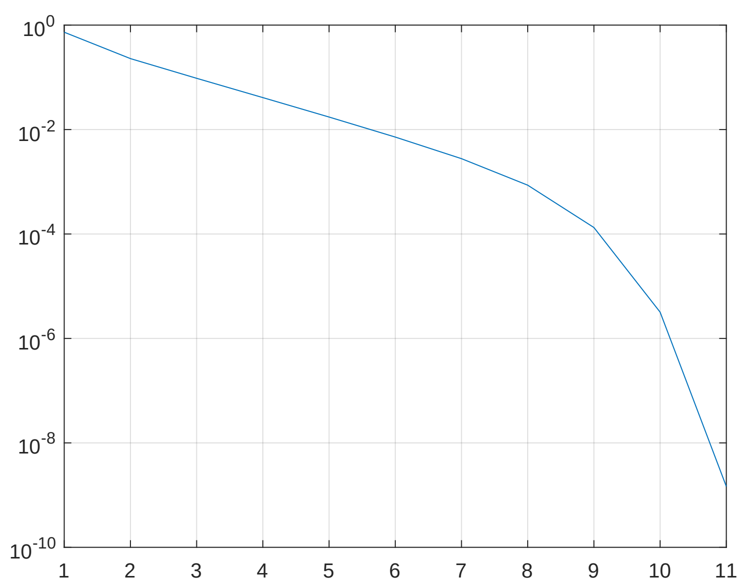



**Figure 32:** Norm of the gradient along iterations for dataset4.m.

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

$r = 7.6718$

The total number of iterations was: 11.

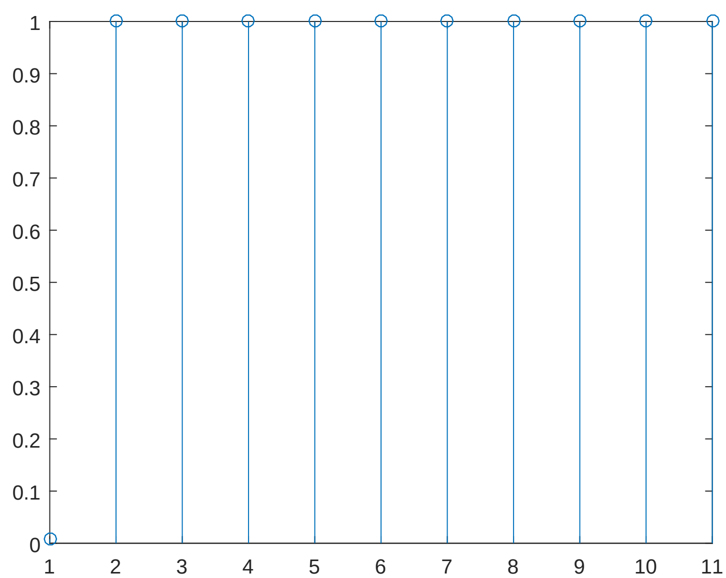



**Figure 33:** Value of alpha along iterations for dataset4.m

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):

```
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 
```

```
%plot alpha for each iteration
figure;
stem(alphas);

%plot figure with logarithmic y—axis
figure;
semilogy(gradients);
grid on;

s = t(1:n)
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

$$
\begin{aligned}
&\underset{x}{\text{minimize}} \quad \underbrace{\sum_{(m,p)\in\mathcal{A}} \left(\|a_m - s_p\| - y_{mp}\right)^2 + \sum_{(p,q)\in\mathcal{S}} \left(\|s_p - s_q\| - z_{pq}\right)^2}_{f(x)} \\
&= \underset{x}{\text{minimize}} \quad \underbrace{\sum_{(m,p)\in\mathcal{A}} \left(g_{mp}(x)\right)^2 + \sum_{(p,q)\in\mathcal{S}} \left(h_{pq}(x)\right)^2}_{f(x)}.
\end{aligned} \tag{5}
$$

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

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

This equation can be rearrange as the following least square problem

$$\underset{x}{\text{minimize}}\, \|Ax - b\|^2$$

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 ($\lambda$) 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

$$s_1 = \begin{bmatrix} -2.0205 \\ 6.3478 \end{bmatrix}, \quad s_2 = \begin{bmatrix} -2.5704 \\ 9.1908 \end{bmatrix}, \quad s_3 = \begin{bmatrix} 3.7052 \\ -1.7748 \end{bmatrix}, \quad s_4 = \begin{bmatrix} -4.4309 \\ -2.3808 \end{bmatrix},$$
$$s_5 = \begin{bmatrix} 1.6285 \\ -11.8744 \end{bmatrix}, \quad s_6 = \begin{bmatrix} 3.3785 \\ 1.1180 \end{bmatrix}, \quad s_7 = \begin{bmatrix} 4.2787 \\ 9.5108 \end{bmatrix}, \quad s_8 = \begin{bmatrix} -1.9070 \\ -5.2535 \end{bmatrix}$$

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

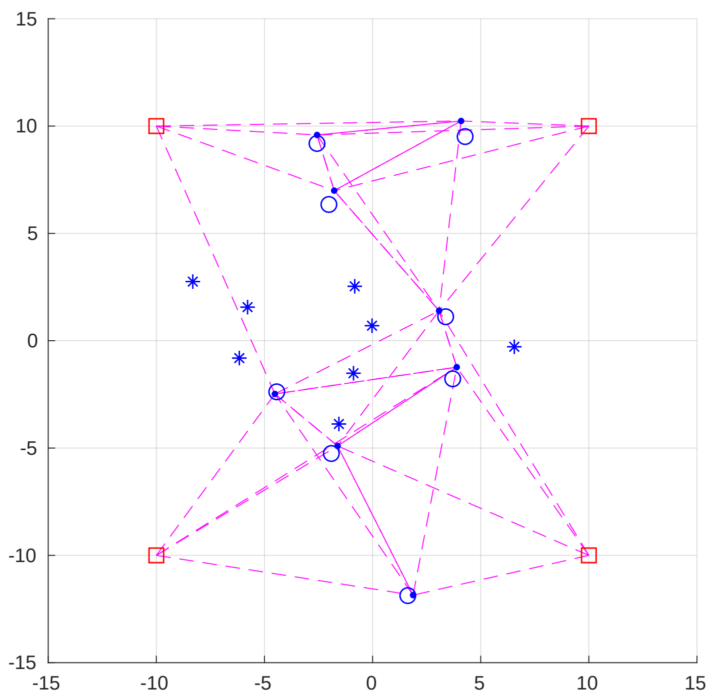



**Figure 34:** Network Localization

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.

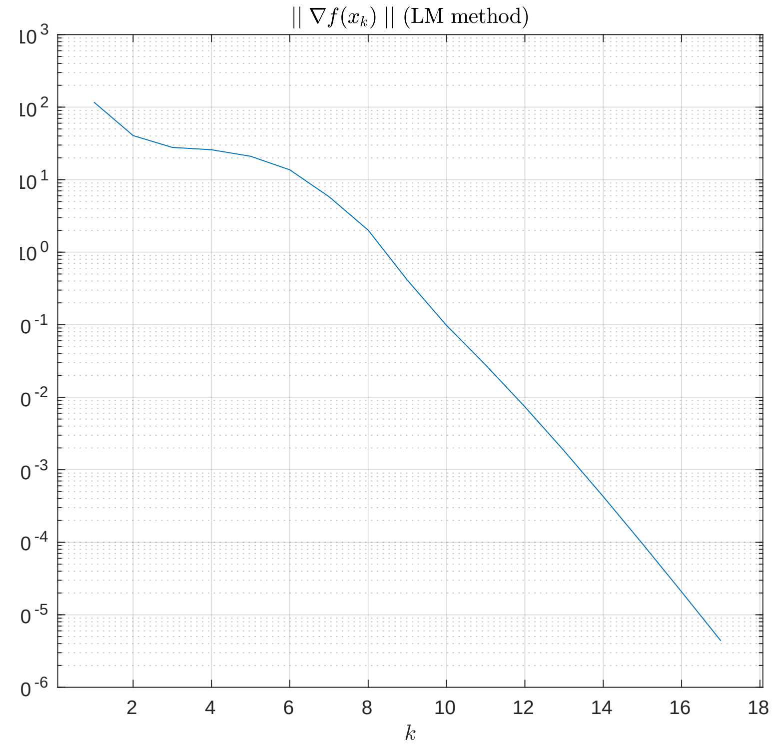



**Figure 35:** Norm of the gradient along the iterations of the LM method

```
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
```

```
for j=1:1:size(iS)
    E(:,:,j) = B(:,:,iS(j,1))-B(:,:,iS(j,2));
end

xk = xinit;
n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );


while n_grad > tolerance

    gradi_fp = gradient_fp(iA, A, iS, B, E, xk);
    func_fp = fp(A, iA, iS, B, y, z, xk, E);

    b_aux = gradi_fp'*xk  - func_fp;
    v_lambda = sqrt(lambda).*xk;
    b = [b_aux; v_lambda];

    A_aux = sqrt(lambda) .* I_a;
    A_ = [gradi_fp'; A_aux];

    belief = A_\b;

    f_belief = f(A, iA, iS, B, y, z, belief, E);
    f_xk = f(A, iA, iS, B, y, z, xk, E);

    if f_belief < f_xk
        xk = belief;
        lambda = 0.7* lambda;
    else
        lambda = 2* lambda;
    end

    iter = iter+1;
    plot_n_grad(iter) = n_grad;
    %data for the next iteration
    n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );


end

sensor = zeros(2,8);
for i=1:1:8
    sensor(:,i) = xk(2*i-1:2*i);
end
plotgraph(A, iA, iS, sensor, xinit, S, plot_n_grad, iter);
```

**Listing 17:** Script Task 8

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

end
```

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

```
function [f_] = fp(A, iA, iS, B, y, z, x, E)
    %returns a (px1) vector (p=size(iA)+size(iS))
    sA =  size(iA, 1);
    sS =  size(iS, 1);
    f_ = zeros(sA+sS, 1);

    for j=1:1:sA
        f_(j) = norm(A(:,iA(j,1)) - B(:,:,iA(j,2))*x) - y(j);
    end

    for i=1:1:sS
        f_(i+sA) = norm(E(:,:,i)*x) - z(i);
    end

end
```

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

```
function [gradientf] = gradient_f(A, iA, iS, B, y, z, x, E)
%return a column of 16x1 with all the parcial derivatives of f
    sA =  size(iA, 1);
    sS =  size(iS, 1);
    grad_f = zeros(16, sS+sA);

    for i=1:1:sA
        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));
    end
    for i=1:1:sS
        grad_f(:,i+sA) = 2*(norm(E(:,:,i)*x)-z(i))*(E(:,:,i)'*(E(:,:,i)*x))
            ↪ ./(norm(E(:,:,i)*x));
    end
   %sum of the rows
   gradientf = sum(grad_f, 2);
end
```

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

```
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 $h_{pq}(x)$

```
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);
```

```
semilogy(v_i, n_grad);
xlabel('$$k$$','Interpreter','latex');
title('$$\mid\mid \nabla f(x_k) \mid\mid$$ (LM method)','Interpreter','
    ↪ latex');
grid on
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.

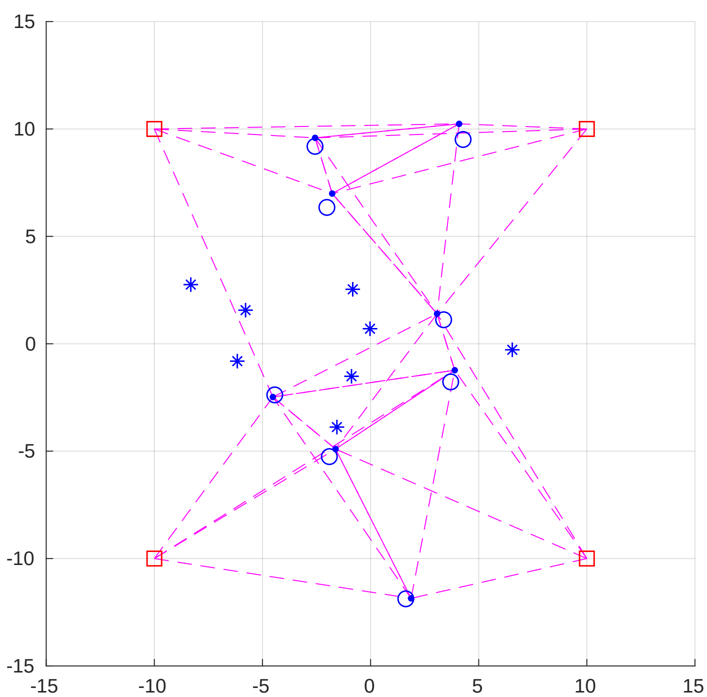



**Figure 36:** Network Localization

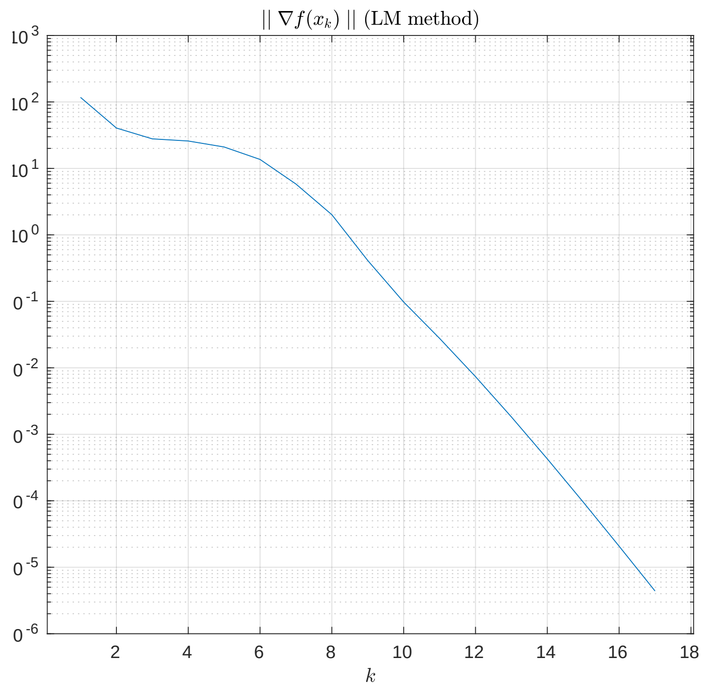



**Figure 37:** Norm of the gradient along the iterations of the LM method

```
clear all;
close all;
cost_function = 1000000;


for it=1:1:5000000

    clearvars -except cost_function x_best iter_best plot_n_grad_best it
        ↪ x_bestinit

    %load the workspace
    load('lmdata2.mat');

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

    iter = 0;
    plot_n_grad = [];

    %generate random vector of 16x1 in the interval of [-11 11]
    xinit = -11 + (11+11).*rand(16,1);

    %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
    for j=1:1:size(iS)
        E(:,:,j) = B(:,:,iS(j,1))-B(:,:,iS(j,2));
    end

    xk = xinit;
    n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );


    while n_grad > tolerance

        gradi_fp = gradient_fp(iA, A, iS, B, E, xk);
        func_fp = fp(A, iA, iS, B, y, z, xk, E);

        b_aux = gradi_fp'*xk  - func_fp;
        v_lambda = sqrt(lambda).*xk;
        b = [b_aux; v_lambda];

        A_aux = sqrt(lambda) .* I_a;
        A_ = [gradi_fp'; A_aux];

        belief = A_\b;

        f_belief = f(A, iA, iS, B, y, z, belief, E);
        f_xk = f(A, iA, iS, B, y, z, xk, E);

        if f_belief < f_xk
            xk = belief;
            lambda = 0.7* lambda;
        else
            lambda = 2* lambda;
        end

        iter = iter+1;
        plot_n_grad(iter) = n_grad;
        %data for the next iteration
        n_grad = norm( gradient_f(A, iA, iS, B, y, z, xk, E) );

        if iter > 1000
            %diverge
            break
        end
    end

```

```

    f_final = f(A, iA, iS, B, y, z, xk, E);

    if f_final < cost_function
        cost_function = f_final;
        x_bestinit = xinit;
        x_best = xk;
        iter_best = iter;
        plot_n_grad_best = plot_n_grad;
    end
end


sensor = zeros(2,8);
for i=1:1:8
    sensor(:,i) = x_best(2*i-1:2*i);
end
plotgraph_task9(A, iA, iS, sensor,x_bestinit, plot_n_grad_best,iter_best);
```

**Listing 23:** Script of task 9

```
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);
semilogy(v_i, n_grad);
xlabel('$$k$$','Interpreter','latex');
title('$$\mid\mid \nabla f(x_k) \mid\mid$$ (LM method)','Interpreter','latex');
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 $g(y)$.

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

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

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

The KKT system is

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

Using the first equation, we can determine $y$ in function of $\mu$.

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

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

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

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

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

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

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

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

However, using the inequality $\mu^* \geqslant 0$, we conclude that if $\left\|p-c\right\|_2 \geqslant r$, that is, if the point $p$ is outside of the circle with center in $c$ and radius $r$, $\mu^*$ 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., $\left\|p-c\right\|_2 < r$, $\mu^* = 0$

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

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

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

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 $\ell_2^2$ norm as matrix multiplication, the optimization problem with the new variable becomes:

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

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

$$\begin{aligned} &\sum_{k=1}^{K} (EX_{\tau_k} z - \omega_k)^T \mathbf{I} (EX_{\tau_k} z - \omega_k) + \sum_{t=1}^{T-1} (Y_t z - Y_{t-1} z)^T \lambda \mathbf{I} (Y_t z - Y_{t-1} z) \\ =&( \begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \end{bmatrix} )^T \mathbf{I} ( \begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \end{bmatrix} ) + ( \begin{bmatrix} Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z)^T \lambda \mathbf{I} ( \begin{bmatrix} Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z) \\ =&( \begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \\ Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \\ 0 \\ \ldots \\ 0 \end{bmatrix} )^T \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \lambda \mathbf{I} \end{bmatrix} ( \begin{bmatrix} EX_{\tau_1} \\ \ldots \\ EX_{\tau_K} \\ Y_1 - Y_0 \\ \ldots \\ Y_{T-1} - Y_{T-2} \end{bmatrix} z - \begin{bmatrix} \omega_1 \\ \ldots \\ \omega_K \\ 0 \\ \ldots \\ 0 \end{bmatrix} ) \\ =&(Rz - v)^T D (Rz - v) \end{aligned}$$

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

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

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, $\lambda^*$ 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.