Project of Optimization and Algorithms

João Xavier
Instituto Superior Técnico
September 2025

Contents

1 Tracking a moving target

We want to control a vehicle so that it tracks a given moving target over a finite discrete-time horizon $\{1, 2, 3, \ldots, T\}$.

1.1 Deterministic target

Target. The trajectory of the target is assumed known from the beginning of the horizon. The trajectory is denoted by $q(t) \in \mathbf{R}^2$ for $1 \leq t \leq T$ and is shown in Figure 1.

The state of our vehicle. The position of our vehicle at time t is denoted by $p(t) \in \mathbf{R}^2$; its velocity, by $v(t) \in \mathbf{R}^2$. The *state* $x(t)$ of our vehicle at time t is

$$x(t) = \begin{bmatrix} p(t) \\ v(t) \end{bmatrix}. \tag{1}$$

Note that $x(t)$ is a four-dimensional vector: $x(t) \in \mathbf{R}^4$, for $1 \leq t \leq T$.

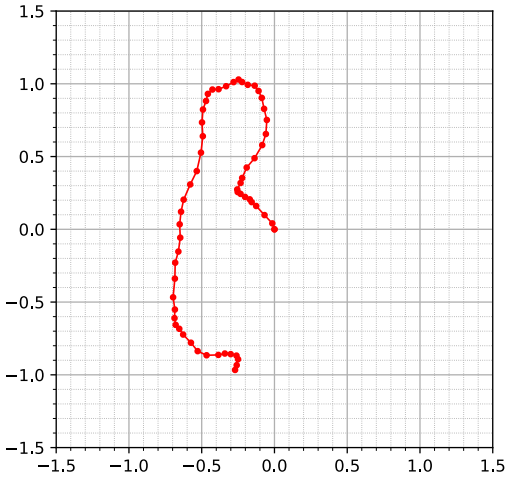

Figure 1: Trajectory of the target, which starts at the origin, for an horizon of $T = 60$ time steps.

The control signal. We act on our vehicle by applying a force at each time t in $\{1, \ldots, T\}$. (These forces are applied in fact by the motor in the vehicle) The force that we apply at time t, denoted by $u(t)$, is two-dimensional, that is, $u(t) \in \mathbf{R}^2$ for $1 \leq t \leq T-1$. The sequence of forces applied throughout time, $\{u(t) \colon t = 1, \ldots, T-1\}$, is called the *control signal*. This is the signal we want to design.

The control signal changes the state of our vehicle. Each time we apply a force to the vehicle, we push it a bit, thereby changing its state (position and velocity). More precisely, if $x(t)$ is the current state and the force $u(t)$ is applied, then the state changes to

$$x(t+1) = Ax(t) + Bu(t), \tag{2}$$

where $A \in \mathbf{R}^{4\times 4}$ and $B \in \mathbf{R}^{4\times 2}$ are known matrices, depending on physical constants such as the mass of the vehicle and the drag coefficient of the environment. For this project, take

$$A = \begin{bmatrix} 1 & 0 & 0.1 & 0 \\ 0 & 1 & 0 & 0.1 \\ 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0.8 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

The initial state (initial position and initial velocity) at $t = 1$ of our vehicle is given and is denoted by x_{init}. Consider

$$x_{\text{init}} = \begin{bmatrix} 0.5 \\ 0 \\ 1 \\ -1 \end{bmatrix}.$$

To illustrate, the trajectory of our vehicle is shown in Figure 2 for a control signal $\{u(t)\colon t = 1, \ldots, T-1\}$ generated at random.

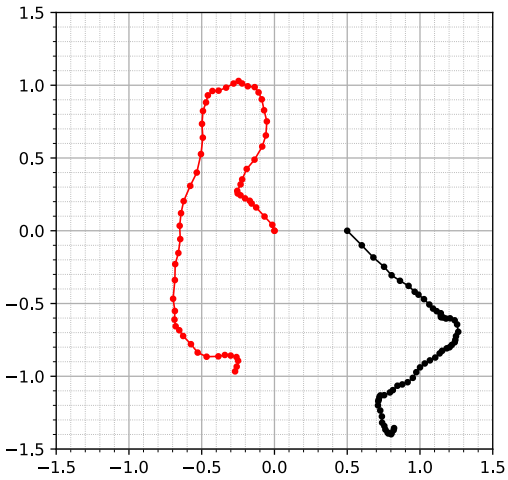

Figure 2: Trajectory of the target (red) and trajectory of our vehicle (black) corresponding to a control signal generated at random.

Our wishes for the control signal. As mentioned earlier, we wish a small tracking error. Thus, we want to design a control signal that makes our vehicle track the target as close as possible, which means that we want $p(t)$ (the position of our vehicle at time t) to be as close as possible to $q(t)$ (the position of the target at time t), for $1 \leq t \leq T$. In this project, we measure the tracking error (TE) as

$$\text{TE} = \sum_{t=1}^{T} \|Ex(t) - q(t)\|_2 \,,$$

where $\|v\|_2 = \sqrt{v^T v}$ is the Euclidean norm and

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Note that, given the definition of the state $x(t)$ in (1), the term $Ex(t)$ is just $p(t)$, the position of our vehicle at time t.

We now add another wish: a small control signal. This is because acting on the vehicle consumes resources (e.g., fuel). In this project, we measure the control effort (CE) as

$$\text{CE} = \sum_{t=1}^{T-1} \|u(t)\|_2^2 \,.$$

The optimization problem. In sum, we have two wishes: a small tracking error TE and a small control effort CE. We now merge these two wishes in a single cost function, arriving at the optimization problem

$$\begin{aligned} \underset{x,u}{\text{minimize}} \quad & \underbrace{\sum_{t=1}^{T} \|Ex(t) - q(t)\|_2}_{\text{TE}} + \rho \underbrace{\sum_{t=1}^{T-1} \|u(t)\|_2^2}_{\text{CE}} \\ \text{subject to} \quad & x(1) = x_{\text{initial}} \\ & x(t+1) = Ax(t) + Bu(t), \qquad \text{for } 1 \leq t \leq T-1. \end{aligned} \tag{3}$$

Some comments about this optimization problem:

- the variables to optimize are x and u, where x stands for $\{x(1), x(2), \ldots, x(T)\}$, which is the sequence of states of our vehicle, and u stands for $\{u(1), u(2), \ldots, u(T-1)\}$, which is the control signal we apply to the vehicle. Thus, we are composing the state trajectory and the control signal, jointly. Of course, these variables are linked by the physics of the problem, which is expressed by the last constraint in (3);
- The two wishes, TE and CE, are merged additively in the cost function of (3) by a weight $\rho > 0$. Increasing ρ increases the importance of CE relative to TE, and vice-versa.

Task 1. [Numerical task] Use the software `CVXPY` from `https://www.cvxpy.org` to solve problem (3) for various values of ρ. Specifically, solve 12 instances of problem (3), defined as follows:

instance i	ρ_i
1	10
2	5
3	2
4	1
5	0.5
6	0.2
7	0.1
8	0.05
9	0.02
10	0.01
11	0.005
12	0.002

After you solve instance i, plot the trajectory of the target against the trajectory of the vehicle. Also, register the tracking error and control effort, which we denote by $\text{TE}^\star_{\rho_i}$ and $\text{CE}^\star_{\rho_i}$. After you solve all instances, plot $\text{TE}^\star_{\rho_i}$ versus $\text{CE}^\star_{\rho_i}$, for $1 \leq i \leq 12$. Comment all your results, that is, explain why they make sense.

The trajectory of the target can be found in the `numpy` file `target_1.npy`.

Example. So that you can have some examples to check your code, we plot the trajectory of the target against the trajectory of the vehicle for instance $i = 5$ that is, for $\rho = 0.5$) in Figure 3.

We also give the pairs $(\text{TE}^\star_{\rho_i}, \text{CE}^\star_{\rho_i})$ for the first and last instance: $(\text{TE}^\star_{\rho_1}, \text{CE}^\star_{\rho_1}) = (88.84, 0.28)$ and $(\text{TE}^\star_{\rho_{12}}, \text{CE}^\star_{\rho_{12}}) = (2.60, 860.26)$, which are plotted in Figure 4.

Task 2. [Theoretical task] Let $\text{TE}^\star_\rho$ and $\text{CE}^\star_\rho$ denote the tracking error and control effort obtained after optimizing (3) for a given $\rho > 0$. Consider now two values of ρ, say, ρ_a and ρ_b, and suppose that $\text{TE}^\star_{\rho_a} \leq \text{TE}^\star_{\rho_b}$. Prove that $\text{CE}^\star_{\rho_a} \geq \text{CE}^\star_{\rho_b}$.

⇒ if a is optimal $TE^*_{\rho_a} + \rho_a CE^*_{\rho_a} \leq TE^*_{\rho_a} + \rho_b CE_a^*$; $TE_{\rho_a}^* \leq TE_{\rho_b}^*$

Task 3. [Theoretical task] Show that the optimization problem (3) has a unique solution. Hint: rewrite (3) as an unconstrained optimization problem that depends only on the variable u.

↳ Strongly Convex function.

Prove Strong Convexity

⇒ pick the constraint & rewrite function as the constraint

⇒ we get a big eqn. we get $f(u) + \beta \|u\|_2^2$ Check from the slides, which we get arguments from we can prove Strong Convex.

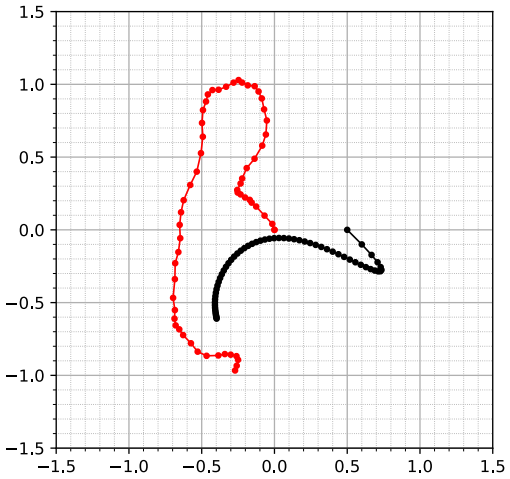

Figure 3: Trajectory of the target (red) and trajectory of our vehicle (black) corresponding to a control signal optimized as in (3) for $\rho = 0.5$.

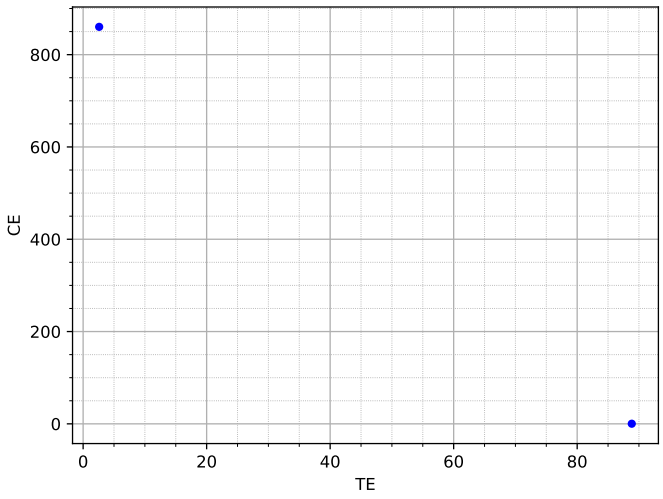

Figure 4: Pairs $(\mathrm{TE}^\star_{\rho_i}, \mathrm{CE}^\star_{\rho_i})$ for $\rho_1 = 10$ and $\rho_{12} = 0.002$. Note that Task 1 asks you to complete this figure by adding the remaining pairs for $i \in \{2, 3, \ldots, 11\}$.

1.2 Random target with no information

Up to now, the trajectory of the target was assumed deterministic. We consider now a more challenging scenario in which the trajectory is random. Specifically, we know only that the target can execute two possible trajectories, but we ignore which one the target is going to execute in the time horizon $\{1, \ldots, T\}$. We know, however, the prior probabilities of the target executing each trajectory, that is, how likely the target is of executing each of the two possible trajectories.

We denote by p_1 the probability that the target executes trajectory 1 and by p_2 the probability that the target executes trajectory 2. Of course, being probabilities, the given constants p_1 and p_2 are non-negative and sum to one: $p_1 \geq 0$, $p_2 \geq 0$, and $p_1 + p_2 = 1$. As a curiosity, note that the special case $p_1 = 1$ and $p_2 = 0$ (respectively, the case $p_1 = 0$ and $p_2 = 1$) corresponds in fact to a deterministic target that executes always trajectory 1 (respectively, trajectory 2).

The two possible trajectories are denoted by $q_1(t) \in \mathbf{R}^2$ (trajectory 1) and by $q_2(t) \in \mathbf{R}^2$ (trajectory 2), for $1 \leq t \leq T$. These two trajectories, along with a trajectory of the vehicle for a randomly generated control signal, are shown in Figure 5.

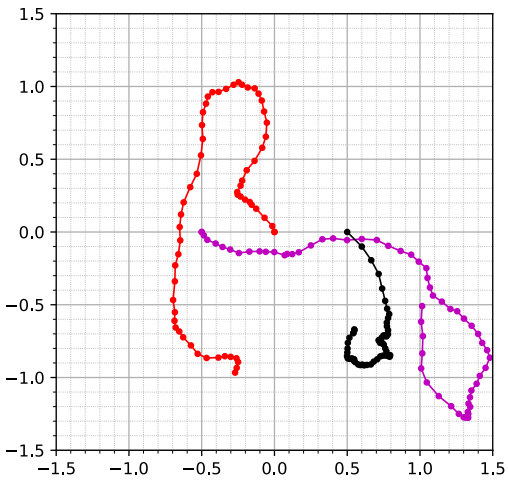

Figure 5: The two known possible trajectories that the target can execute, trajectory 1 (red) and trajectory 2 (magenta), and a trajectory of our vehicle (black) corresponding to a control signal generated at random.

The optimization problem. As mentioned, the probabilities p_1 and p_2 and the possible trajectories $q_1(t)$ and $q_2(t)$ are known to us. What we ignore is which one of the two possible trajectories the target is actually going to execute from $t = 1$ to $t = T$. To cope with this uncertainty, we choose to design a control signal that minimizes the average tracking error

now we have 2 trajectory, with probability p_1 & p_2

plus the control effort:

$$\begin{array}{ll} \underset{x,u}{\text{minimize}} & \underbrace{p_1 \underbrace{\sum_{t=1}^{T} \|Ex(t) - q_1(t)\|_2}_{\text{TE}_1} + p_2 \underbrace{\sum_{t=1}^{T} \|Ex(t) - q_2(t)\|_2}_{\text{TE}_2}}_{\text{average TE}} + \rho \underbrace{\sum_{t=1}^{T-1} \|u(t)\|_2^2}_{\text{CE}} \\ \text{subject to} & x(1) = x_{\text{initial}} \\ & x(t+1) = Ax(t) + Bu(t). \quad \text{for } 1 \leq t \leq T-1. \end{array} \tag{4}$$

In (4), the terms TE_1 and TE_2 represent the mismatches between the trajectory of the vehicle and each of the possible trajectories of the target. Thus, the term $p_1\text{TE}_1 + p_2\text{TE}_2$ in the cost function represents the average tracking error.

Task 4. [Numerical task] Use the software `CVXPY` from `https://www.cvxpy.org` to solve problem (4) for various values of p_1 and p_2, always taking $\rho = 0.1$. Specifically, solve 11 instances of problem (3), defined as follows:

instance i	$(p_1)_i$	$(p_2)_i$
1	0	1
2	0.1	0.9
3	0.2	0.8
4	0.3	0.7
5	0.4	0.6
6	0.5	0.5
7	0.6	0.4
8	0.7	0.3
9	0.8	0.2
10	0.9	0.1
11	1	0

After you solve instance i, plot the two possible trajectories of the target against the trajectory of the vehicle. Comment your results.
Trajectory 1 of the target can be found in the file `target_1.npy` and trajectory 2 can be found in the file `target_2.npy`.

Example. So you can check your code we plot the result of the instance $p_1 = 0.6$ and $p_2 = 0.4$ in Figure 6.

1.3 Random target with midway information

We keep the scenario described in the previous section 1.2 and add a twist: we assume that the trajectory—either 1 or 2—that the target executes from time $t = 1$ is revealed to us at

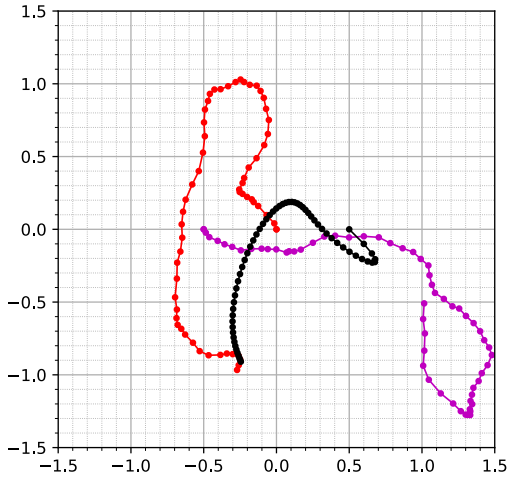

Figure 6: The two known possible trajectories that the target can execute, trajectory 1 (red) and trajectory 2 (magenta), and a trajectory of our vehicle (black) corresponding to a control signal optimized as in (4) for $p_1 = 0.6$ and $p_2 = 0.4$.

$t = 30$. That is, from $t = 1$ to $t = 29$ we ignore which trajectory the target is executing; we know only their prior probabilities. But, about midway in the time horizon, at $t = 30$, we are told which trajectory the target has actually chosen to execute since $t = 1$ (and will keep executing until the end of the time horizon).

The optimization problem. As a starting point to map this scenario into an optimization problem, consider the following incomplete formulation: now we have 2 vechiles

$$\begin{array}{ll} \underset{x_1,u_1,x_2,u_2}{\text{minimize}} & \sum_{k=1}^{K} p_k \left(\sum_{t=1}^{T} \|Ex_k(t) - q_k(t)\|_2 + \rho \sum_{t=1}^{T-1} \|u_k(t)\|_2^2 \right) \\ \text{subject to} & x_1(1) = x_{\text{initial}} \\ & x_1(t+1) = Ax_1(t) + Bu_1(t) \quad \text{for } 1 \le t \le T-1 \\ & x_2(1) = x_{\text{initial}} \\ & x_2(t+1) = Ax_2(t) + Bu_2(t) \quad \text{for } 1 \le t \le T-1 \\ & \boxed{\text{more constraints needed here}}. \end{array} \tag{5}$$

In problem (5) we have two kinds of variables: x_1, u_1 and x_2, u_2. The variable x_k stands for $\{x_k(1), x_k(2), \ldots, x_k(T)\}$, which is the sequence of states of the vehicle resulting from the

application of the control sequence u_k, which stands for $\{u_k(1), u_k(2), \ldots, u_k(T-1)\}$.

Task 5. [Theoretical task] Explain why more constraints are indeed needed in problem (5). Or, put another way, what's wrong with the formulation in (5) if no extra constraints are inserted?

Task 6. [Theoretical task] State which extra constraints need to be added in the indicated slot in formulation (5). (You cannot add more variables; you cannot also change the cost function.) Note that $u_k(t)$, from $t = 30$ onwards, is the control signal we apply after being told that the target is executing trajectory k.

Task 7. [Numerical task] Use the software `CVXPY` from `https://www.cvxpy.org` to solve the optimization problem resulting from you answer to Task 6 for the case $\rho = 0.1$, $p_1 = 0.6$ and $p_2 = 0.4$. After the problem is numerically solved, plot the two possible trajectories of the target against the two possible trajectories of the vehicle (which correspond to x_1 and x_2).

Examples. We give the solution for the case $\rho = 0.1$, $p_1 = 0.4$ and $p_2 = 0.6$ in Figure 7. Additionally, the case $\rho = 0.1$, $p_1 = 0.9$ and $p_2 = 0.1$ is shown in Figure 8.

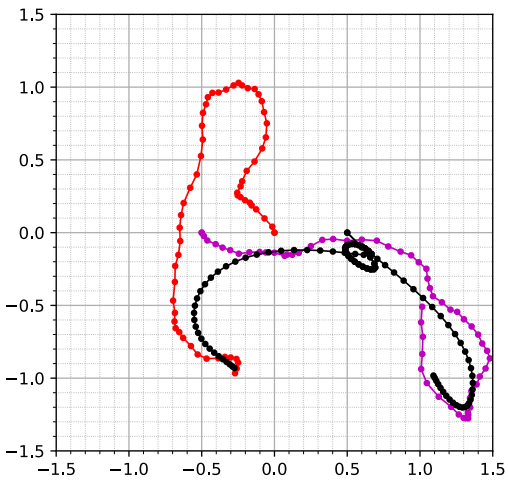

Figure 7: The two known possible trajectories that the target can execute, trajectory 1 (red) and trajectory 2 (magenta), and the two trajectories of our vehicle (black) corresponding to the control signals optimized as in (5) for $\rho = 0.1$, $p_1 = 0.4$ and $p_2 = 0.6$. The trajectories x_1 and x_2 split at $t = 30$.

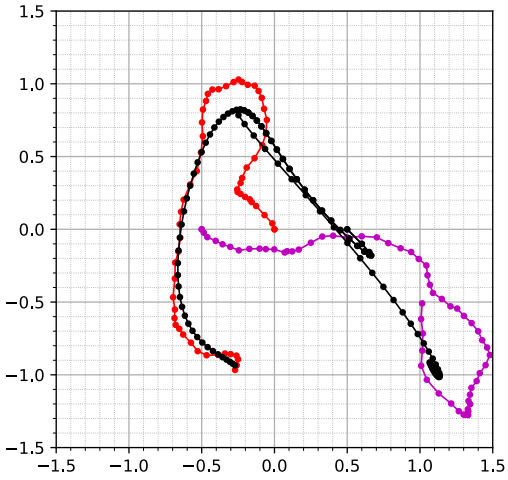

Figure 8: The two known possible trajectories that the target can execute, trajectory 1 (red) and trajectory 2 (magenta), and the two trajectories of our vehicle (black) corresponding to the control signals optimized as in (5) for $\rho = 0.1$, $p_1 = 0.9$ and $p_2 = 0.1$. The trajectories x_1 and x_2 split at $t = 30$.

2 Fitting a circle to a noisy dataset: centralized setting

We want to fit a circle to a given noisy dataset of points in the plane. The dataset consists of points $x_n \in \mathbf{R}^2$, for $1 \leq n \leq N$. An example dataset is in Figure 9.

2.1 The optimization problem

We parameterize the circle by its center $c \in \mathbf{R}^2$ and its radius R. Fitting c and R to the dataset can be cast as the optimization problem

$$\underset{c,R}{\text{minimize}} \underbrace{\sum_{n=1}^{N} \left(\|c - x_n\|_2 - R\right)^2}_{f(c,R)}. \tag{6}$$

Problem (6) has a simple interpretation: It finds the best-fitting circle by minimizing the squared distances from the points to the circle. Our main goal is to solve (6).

You might wonder why does formulation (6) not include a constraint such as $R \geq 0$. The constraint is unnecessary because the optimal R for any given c is automatically nonnegative. That is, if we fix c and minimize $f(c, R)$ over R, we get

$$R^\star(c) = \frac{1}{N} \sum_{n=1}^{N} \|c - x_n\|_2 , \tag{7}$$

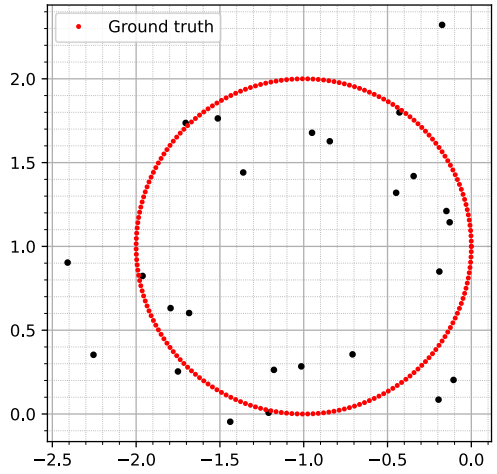

Figure 9: A dataset of $N = 25$ points (black) obtained by adding noise to an underlying ground truth circle (red).

which is always non-negative. Equality (7) has a simple interpretation, too: it computes the average distance from the points to the candidate center c.

Task 8. [Theoretical task] Show that problem (6) is nonconvex by showing that the cost function f is nonconvex.

2.2 A sub-optimal approach via least-squares (LS)

We start by approximating the nonconvex problem (6) by a convex one. To create such approximation, we reason as if the dataset were not noisy. In such an ideal scenario the N points lie exactly on a circle, which means that the N equalities

$$\|x_n - c\|_2^2 = R^2, \quad n = 1, \dots, N$$

must hold for some c and R.

These N equalities can be written as the linear system

$$\underbrace{\begin{bmatrix} 1 & -2x_1^T \\ 1 & -2x_2^T \\ \vdots & \vdots \\ 1 & -2x_N^T \end{bmatrix}}_{A} \begin{bmatrix} y \\ c \end{bmatrix} = \underbrace{\begin{bmatrix} -\|x_1\|_2^2 \\ -\|x_2\|_2^2 \\ \vdots \\ -\|x_N\|_2^2 \end{bmatrix}}_{b} \tag{8}$$

together with the nonlinear equation

$$y = \|c\|_2^2 - R^2. \tag{9}$$

Equations (8) and (9) suggest the following two-step approach:

1. solve the least-squares (LS) problem

$$\underset{y,c}{\text{minimize}} \left\| A \begin{bmatrix} y \\ c \end{bmatrix} - b \right\|_2^2, \tag{10}$$

   (with A and b as in (8)) denoting the solution by $(y^\star, c_{\text{ls}}^\star)$;

2. obtain the associated radius by setting

$$R_{\text{ls}}^\star = \left(\|c_{\text{ls}}^\star\|_2^2 - y^\star \right)^{1/2}. \tag{11}$$

Because it is based on an LS problem, we call this approach the LS approach. For example, applying the LS approach to the dataset in Figure 9 gives the circle in Figure 10. You

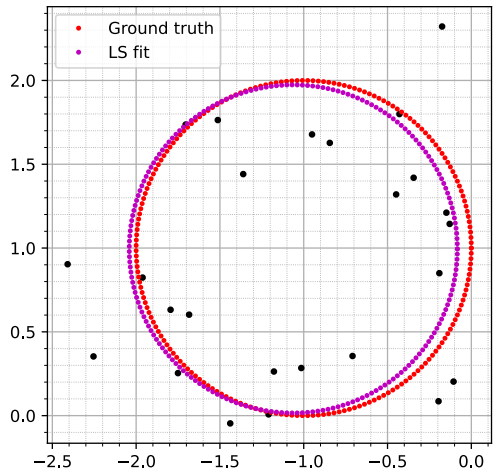

Figure 10: The ground truth circle (red) and the circle fitted by the LS approach (magenta)

are encouraged to replicate Figure 10: load the dataset from the file `circle_data_1.npy`, apply the LS approach, and confirm that you arrive at the center $c_{\text{ls}}^\star = (-1.06, 0.99)$ and radius $R_{\text{ls}}^\star = 0.98$.

Task 9. [Numerical task] Consider now a dataset that is obtained from a non-uniform sampling. Such dataset is illustrated in Figure 11.
Load the dataset in Figure 11 from the file `circle_data_2.npy`, apply the LS approach, and report the center $c_{\text{ls}}^\star$ and radius $R_{\text{ls}}^\star$ that you found. Also, draw the circle with center $c_{\text{ls}}^\star$ and radius $R_{\text{ls}}^\star$ superimposed to the dataset (as illustrated in Figure 10). To solve the LS problem in (8) in `numpy`, use the function `np.linalg.lstsq`.

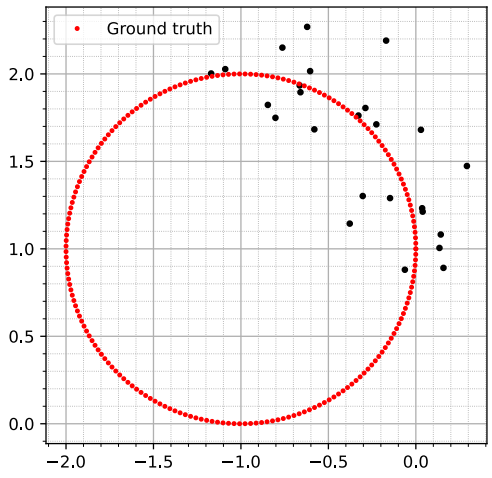

Figure 11: A dataset of $N = 25$ points (black) obtained by adding noise to an underlying ground truth circle (red). Only the first quadrant has been sampled.

2.3 An optimal approach via the Levenberg-Marquardt (LM) algorithm

Although the previous LS approach can work well for some datasets, it is not guaranteed to deliver a local minimizer (let alone a global minimizer) for our problem of interest (6).

Now, problem (6) is a nonlinear least-squares problem. So, it can be addressed directly by the Levenberg-Marquardt (LM) algorithm, which you must implement as described in slide 74 of module 2. From now on, as parameters of the LM algorithm, use $\lambda_0 = 1$ and $\epsilon = 1e-6$. Also, to initialize the LM algorithm, use the center $c^\star_{\text{ls}}$ and radius $R^\star_{\text{ls}}$ found from the LS approach. For the least-squares problem that must be solved at each iteration of the LM algorithm use the `numpy` function `np.linalg.lstsq`.

For example, applying the LM algorithm to the dataset in Figure 9 gives the circle in Figure 12.

You are encouraged to replicate Figure 12: load the dataset from `circle_data_1.npy`, apply your implementation of the LM algorithm, and confirm that you arrive at the center $c^\star_{\text{lm}} = (-1.06, 0.95)$ and radius $R^\star_{\text{lm}} = 0.94$. Also, as you run the LM algorithm keep track of the norm of the gradient across the iterations and plot them at the end: you should obtain Figure 13.

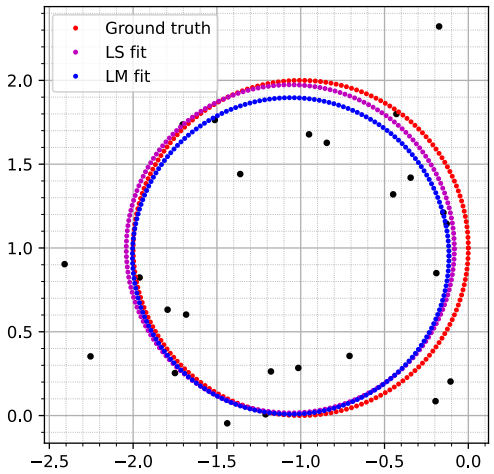

Figure 12: The ground truth circle (red), the circle fitted by the LS approach (magenta), and the circle fitted by the LM algorithm (blue).

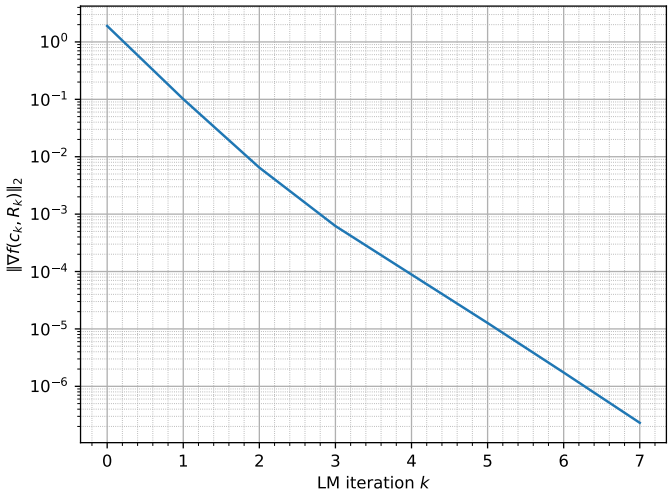

Figure 13: Norm of the gradient of the cost function f in (6) across iterations of the LM algorithm.

Task 10. [Numerical task] Apply your implementation of the LM algorithm to the dataset in Figure 11, which is available in the file `circle_data_2.npy`, and report the center $c^\star_{\text{lm}}$ and radius $R^\star_{\text{lm}}$ that you found. Also, draw the circle with center $c^\star_{\text{lm}}$ and radius $R^\star_{\text{lm}}$ superimposed to the dataset and the circle fitted by the LS approach (as illustrated in Figure 12). Finally, plot the norm of the gradient of f across iterations of the LM algorithm (as illustrated in Figure 13).

3 Fitting a circle to a noisy dataset: distributed setting

In Section 2, we assumed a centralized setting, where the dataset is fully available at some single agent. In this section, we consider a distributed setting, in which the data is scattered across multiple agents, each agent having access only to a small portion of the full dataset.

More precisely, we consider a federated learning setting, with $P = 4$ agents linked to a server—see Figure 14.

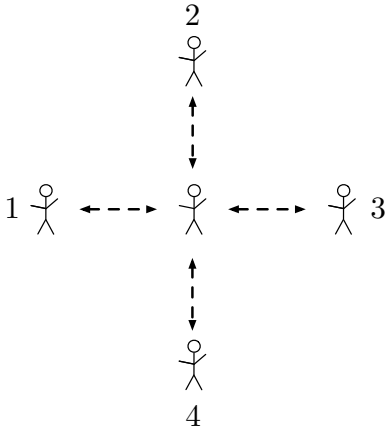

Figure 14: A federated learning setting with a server (in the middle) linked to $P = 4$ agents. Each link is a communication channel that be used used in the direction server-to-agent (downlink) and agent-to-server (uplink).

The dataset is scattered across the four agents. The local data at the agents is to be kept private, that is, the local data is not to be shared with either the remaining agents or the server. Hence, the server has no data. The role of the server is to coordinate the computations such that the collective of agents solve a given optimization problem.

Local datasets. We assume each agent has a local dataset with $N = 10$ points, namely, agent p has the data points $x_{p;n} \in \mathbf{R}^2$ for $1 \leq n \leq N$.

The dataset of agent 1 is available in the file `dataset1.npy`, the dataset of agent 2 in the file `dataset2.npy`, and so on. Figure 15 shows the local dataset of each agent and the circle

fitted by each agent based on its local data alone, each circle fitted by the LS approach of Section 2.2.

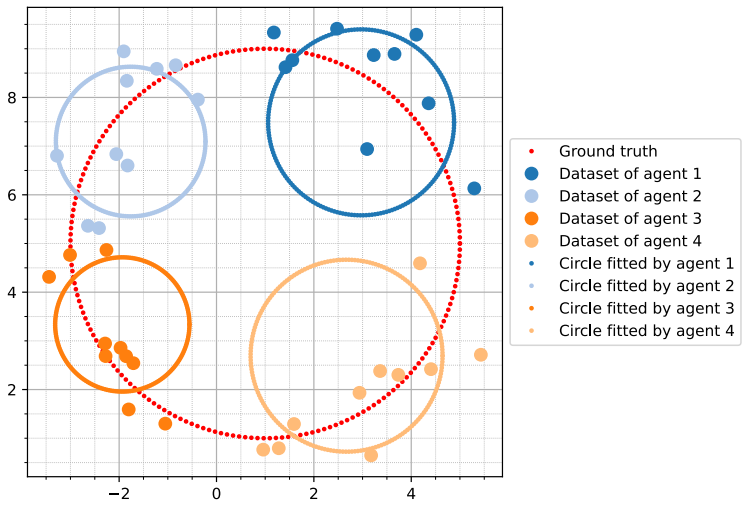

Figure 15: The four local datasets and the corresponding four local circles fitted by the four agents on their own.

As Figure 15 shows, the local circles obtained in this way differ substantially from the ground truth circle. This is because each local circle is fitted by an agent on its own and, as such, does not profit from the datasets of the remaining agents. Clearly, we need the agents to collaborate. However, the agents do not want to share their local data. How to coordinate the agents so that, although no data is shared, the agents arrive at a circle that accounts for the full dataset?

Augmented Lagrangian Method (ALM). A possible approach is via the augmented Lagrangian method (ALM).

First, note that the collective of agents want to solve the optimization problem (10), which, in our distributed setting, corresponds to

$$\underset{x}{\text{minimize}} \sum_{p=1}^{P} \|A_p x - b_p\|_2^2 \tag{12}$$

where $x = (y, c) \in \mathbf{R}^3$ and

$$A_p = \begin{bmatrix} 1 & -2x_{p;1}^T \\ 1 & -2x_{p;2}^T \\ \vdots & \vdots \\ 1 & -2x_{p;N}^T \end{bmatrix}, \quad b_p = \begin{bmatrix} -\|x_{p;1}\|_2^2 \\ -\|x_{p;2}\|_2^2 \\ \vdots \\ -\|x_{p;N}\|_2^2 \end{bmatrix}. \tag{13}$$

The objective function in (13) accounts for the full dataset. This function, however, is not known by any agent, since agent p knows only *its* A_p and b_p (and not *all* the A_p's and b_p's).

Next, reformulate (12) in the equivalent problem

$$\begin{array}{ll} \underset{x_0,x_1,\ldots,x_P}{\text{minimize}} & \sum_{p=1}^{P} \|A_p x_p - b_p\|_2^2 \ . \\ \text{subject to} & x_0 - x_1 = 0 \\ & \vdots \\ & x_0 - x_P = 0. \end{array} \tag{14}$$

In problem (14) the optimization variables are $x_0 \in \mathbf{R}^3$ and $x_p \in \mathbf{R}^3$ for $1 \leq p \leq P$. As will be seen, the variable x_0 will be stored and managed at the server, whereas the variable x_p will be stored and managed by agent p, for $1 \leq p \leq P$.

Finally, apply the ALM to problem (14), which gives the following algorithm:

Augmented Lagrangian Method (ALM) applied to (14)

---

initialize $\lambda^0 = (\lambda_1^0, \ldots, \lambda_P^0)$ and $c > 0$
set $k = 1$
loop
    compute

$$\left(x_0^k, x_1^k, \ldots, x_P^k\right) = \underset{(x_0,x_1,\ldots,x_P)}{\operatorname{argmin}} L_{\lambda^{k-1},c}(x_0, x_1, \ldots, x_P) \tag{15}$$

    update $\lambda_p^k = \lambda_p^{k-1} + c\left(x_0^k - x_p^k\right)$ for $1 \leq p \leq P$
    if $\left\|x_0^k - x_p^k\right\|_2 \leq \epsilon_{\text{ALM}}$ for all $1 \leq p \leq P$, then exit the loop
    $k \leftarrow k + 1$
end loop

Here, for a generic $\lambda = (\lambda_1, \ldots, \lambda_P)$ (with $\lambda_p \in \mathbf{R}^3$ for $1 \leq p \leq P$) and $c > 0$, the symbol $L_{\lambda,c}$ denotes the augmented Lagrangian associated with problem (14), that is, the function

$$L_{\lambda,c} : \ \left(\mathbf{R}^3\right)^{P+1} \to \mathbf{R},$$

where

$$L_{\lambda,c}\left(x_0, x_1, \ldots, x_P\right) = \sum_{p=1}^{P} \left(\|A_p x_p - b_p\|_2^2 + \lambda_p^T \left(x_0 - x_p\right) + \frac{c}{2} \|x_0 - x_p\|_2^2 \right).$$

Distributed implementation of ALM. The only ingredient unclear now is how to solve optimization problem (15), a problem that must be solved at each iteration of ALM to pass from $\left(x_0^{k-1}, x_1^{k-1}, \ldots, x_P^{k-1}\right)$ to $\left(x_0^k, x_1^k, \ldots, x_P^k\right)$.

For this, we consider a block-coordinate descent (BCD) algorithm that minimizes the function $L_{\lambda^{k-1},x}$ over $x_1, \ldots, x_P$ (with x_0 fixed), then over x_0 (with $x_1, \ldots, x_P$ fixed), and so on until no significant change is detected in x_0. More precisely, the BCD algorithm is as follows.

Block-Coordinate Descent (BCD) algorithm for addressing (15)

initialize $u_0^0 = x_0^{k-1}$
set $m = 1$
loop
    compute

$$(u_1^m, \ldots, u_P^m) = \underset{(u_1,\ldots,u_P)}{\operatorname{argmin}} L_{\lambda^{k-1},c}(u_0^{m-1}, u_1, \ldots, u_P) \tag{16}$$

    compute

$$u_0^m = \underset{u_0}{\operatorname{argmin}} L_{\lambda^{k-1},c}(u_0, u_1^m, \ldots, u_P^m) \tag{17}$$

    if $\left\|u_0^m - u_0^{m-1}\right\|_2 \leq \epsilon_{\text{BCD}}$ exit the loop
    $m \leftarrow m + 1$
end loop
set $x_0^k = u_0^m$ and $x_p^k = u_p^m$ for $1 \leq p \leq P$

This BCD algorithm can be implemented in the federated setting of Figure 14, if we let the server store and update x_0 and each agent p store and update x_p:

(a) the server, which holds x_0^{k-1}, starts by setting $u_0^0 = x_0^{k-1}$ and $m = 1$;

(b) the server sends u_0^{m-1} to all the agents (downlink communication). Thus, at this point, each agent p knows u_0^{m-1};

(c) problem (16) can then be solved in parallel by the P agents, because, for fixed u_0^{m-1}, the objective function in (16) separates additively in P elementary functions, the pth elementary function involving only the dataset and variables u_p and u_0^{m-1}, all of which agent p knows. Thus, each agent p can generate locally u_p^m on its own;

(d) the P agents send their u_p^m to the server (uplink communication);

(e) the server, upon receiving $(u_1^m, \ldots, u_P^m)$ from the agents, solves problem (17), thus generating u_0^m;

(f) the server then checks the exit condition: if the condition does not apply, the server increments the counter m and we go back to step (b) for one more round of the BCD algorithm.

Task 11. [Numerical task] Solve problem (14) via the described ALM and BCD algorithms. For ALM, initialize $\lambda_p^0 = 0$ for $1 \leq p \leq P$, and use $c = 100$ and $\epsilon_{\text{ALM}} = 0.01$; for the BCD algorithm, use $x_0^0 = 0$ and $\epsilon_{\text{BCD}} = 0.01$. After ALM terminates, recover the center and radius of the fitted circle as follows: write $x_0 \in \mathbf{R}^3$ as $x_0 = (y, c)$, where $y \in \mathbf{R}$ and $c_{\text{FL}} \in \mathbf{R}^2$; take the center as c_{FL} and the radius as $R_{\text{FL}} = \left(\|c_{\text{FL}}\|_2^2 - y\right)^{1/2}$. Report the center and radius thus obtained and plot this circle fitted by Federated Learning against the local datasets and local circles of Figure 15. Also, report the number of ALM iterations, and for each ALM iteration, the number of BCD iterations. Plot also the maximum constraint violation $\max\left\{\left\|x_0^k - x_p^k\right\|_2 \,:\, 1 \leq p \leq P\right\}$ across the iterations k of ALM.

So that you can check your code, we give the results for the case $c = 50$. Figure 16 shows the circle fitted by Federated Learning: the center is $(1.23, 5.04)$ and the radius is 4.08.

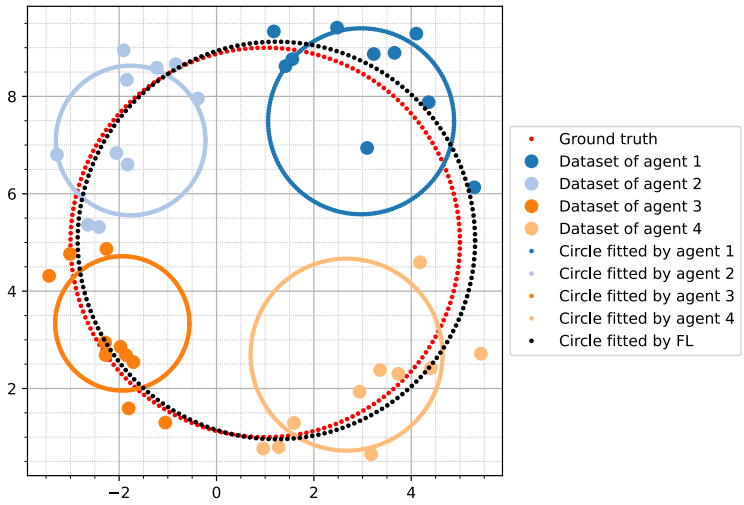

Figure 16: The four local datasets and the corresponding four local circles fitted by the four agents on their own, plus the circle fitted by Federated Learning.

This example took 6 ALM iterations. The maximum constraint violation across iterations is plotted in Figure 17, and the number of BCD iterations for each ALM iteration is given in Table 1.

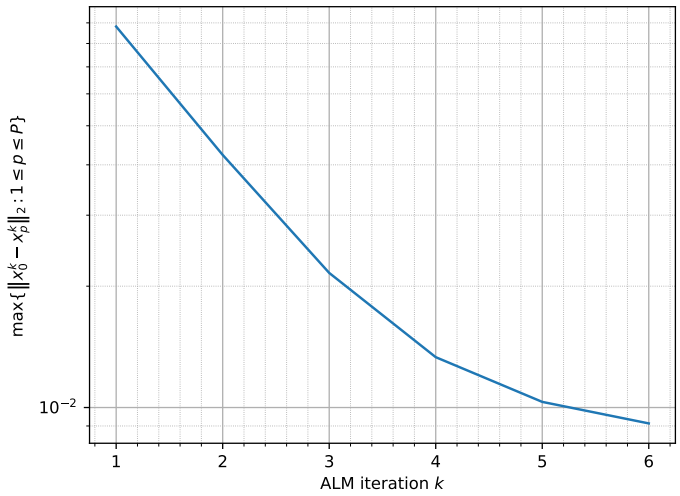

Figure 17: Maximum constraint violation across ALM iterations, for the case $c = 50$.

ALM iteration k	Number of BCD iterations
1	115
2	26
3	25
4	11
5	3
6	2

Table 1: Number of BCD iterations per ALM iteration, for the case $c = 50$.