Optimization and Algorithms
Part 2 of the Project

João Xavier
Instituto Superior Técnico
November 2018

Contents

1 Logistic regression

Automatic prediction. Consider the dataset $D = \{(x_k, y_k) \in \mathbf{R}^2 \times \{0,1\} \colon k = 1, \ldots, K\}$ pictured in Figure 1. In this dataset, each element is a pair of the form (x_k, y_k), where $x_k \in \mathbf{R}^2$ is a vector with two entries and $y_k \in \{0,1\}$ is a binary label, encoded in Figure 1 with the color red for $y_k = 0$ and blue for $y_k = 1$.

This kind of dataset arises in many real-life setups. For example, it could be generated by a doctor as follows. The doctor starts by taking two measurements (say, weight and blood pressure) from an individual 1, thereby filling the two entries of the vector $x_1 \in \mathbf{R}^2$; then, based on those measurements, the doctor decides whether individual 1 suffers from a certain illness, thereby filling the binary label $y_1 \in \{0,1\}$ (for example, setting $y_1 = 1$ if the individual has the illness, and $y_1 = 0$, otherwise). As the doctor repeats this procedure for individuals $2, 3, \ldots, K$, the dataset D is generated.

In applications like these, an interesting problem is automatic prediction, that is, prediction without the intervention of the doctor: given a *new* individual with measurements $x \in \mathbf{R}^2$, what should be his binary label $y \in \{0,1\}$?

Probabilistic model. One way to address the prediction problem is to model how the vector of measurements x impacts the behaviour of the binary label y. A popular such model is logistic regression.

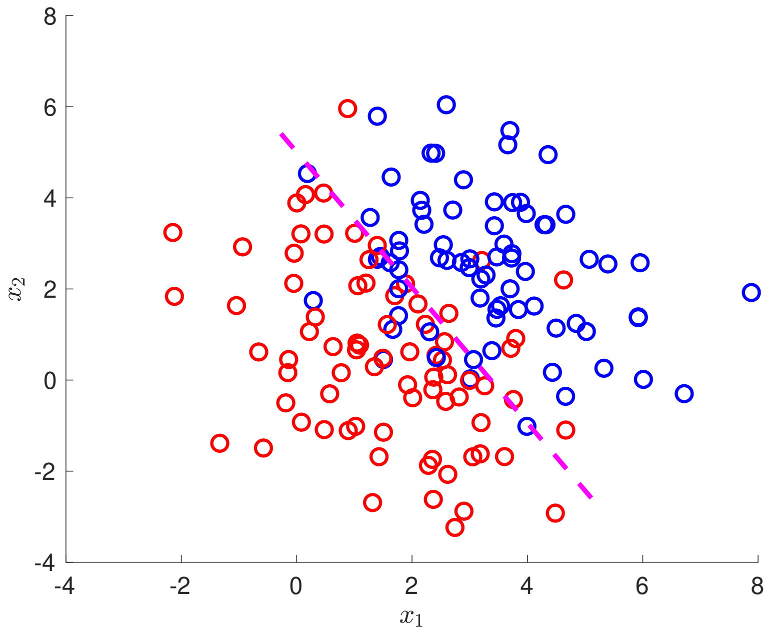

Figure 1: A dataset $D = \{(x_k, y_k) \in \mathbf{R}^2 \times \{0,1\} \colon k = 1, \ldots, 150\}$ where each element $x_k \in \mathbf{R}^2$ is colored blue if the corresponding binary label y_k is 1, and colored red if y_k is 0. As we move to the right, orthogonally to the magenta line, the blue label becomes more probable; conversely, as we move to the left, orthogonally to the magenta line, the red label becomes more probable. On the magenta line, both labels are equally probable.

In logistic regression, a given x creates a probability distribution for the label y, thus making the label y either more likely to be 0 or 1. Specifically, for a given x, logistic regression assigns the probability of y being 1 to be equal to $\exp(s^T x - r)/(1 + \exp(s^T x - r))$ and the probability of y being 0 to be equal to $1/(1 + \exp(s^T x - r))$. Here, $s \in \mathbf{R}^n$ and $r \in \mathbf{R}$ are the model parameters.

Roughly speaking, logistic regression says that when a vector of measurements x verifies $s^T x - r > 0$, its label y is more likely to be 1; in fact, the more positive is $s^T x - r$, the more likely the label is to be 1 (because the probability $\exp(s^T x - r)/(1 + \exp(s^T x - r))$ converges to 1 as $s^T x - r$ grows). Conversely, if $s^T x < r$, its label y is more likely to be 0. Finally, for vectors x satisfying $s^T x = r$, the label y is equally likely to be 0 or 1. In two dimensions, the equation $s^T x = r$ defines a line; in three dimensions, it defines a plane, and so on. Figure 1 also shows the line $s^T x = r$ in magenta.

To conclude, once we know a value for (s, r) matched to our dataset, we can "solve" the prediction problem easily: we simply output, for given new x, the probabilities of its corresponding y being 1 or 0 as $\exp(s^T x - r)/(1 + \exp(s^T x - r))$ and $1/(1 + \exp(s^T x - r))$, respectively.

But how do we find an (s, r) matched to our dataset?

The optimization problem. An (s, r) matched to our dataset can be found by a famous estimation principle in probability—the maximum likelihood (ML) principle. In ML, we search for the (s, r) that maximizes the likelihood of the parameters (s, r) given the dataset D. For the generic logistic regression model in dimension n (that is, in the dataset D, each vector x_k has n measurements), ML leads to the following optimization problem:

$$\underset{(s,r)\in\mathbf{R}^n\times\mathbf{R}}{\text{minimize}} \quad \underbrace{\frac{1}{K}\sum_{k=1}^{K}\left(\log\left(1+\exp(s^T x_k - r)\right) - y_k\left(s^T x_k - r\right)\right)}_{f(s,r)}. \tag{1}$$

By solving problem (1), we find the (s, r) best matched to our dataset.

Task 1. Show that the objective function f in problem (1) is convex.

1.1 Gradient method

The dataset for Figure 1 is in the MATLAB file `data1.mat`, which contains a matrix `X` of size $n \times K$ (where $n = 2$ and $K = 150$) and a vector `Y` of length K: column k of the matrix

X is x_k, and entry k of the vector Y is y_k.

Task 2. Solve problem (1) for the dataset `data1.mat` using the gradient method given in slide 48 of the set of slides "Part 2: unconstrained optimization" (available in the course webpage):

- Regarding slide 48, use the initialization $s_0 = (-1, -1)$ and $r_0 = 0$, and set $\epsilon = 10^{-6}$ for the stopping criterion;
- For the backtracking subroutine, use the values of $\widehat{\alpha}$, γ, and β given in slide 47.

Report the values of s and r that you obtain; plot the line $\{x \in \mathbf{R}^2 \colon s^T x = r\}$ along with the dataset; and plot the norm of the gradient of the cost function along iterations.

To check your code, we now give the answers for task 1 (which you should reproduce). The gradient method finds $s = (1.3495, 1.0540)$ and $r = 4.8815$. Figure 2 plots the line $\{x \in \mathbf{R}^2 \colon s^T x = r\}$ along with the dataset `data1.mat`, and Figure 3 plots the norm of the gradient along iterations.

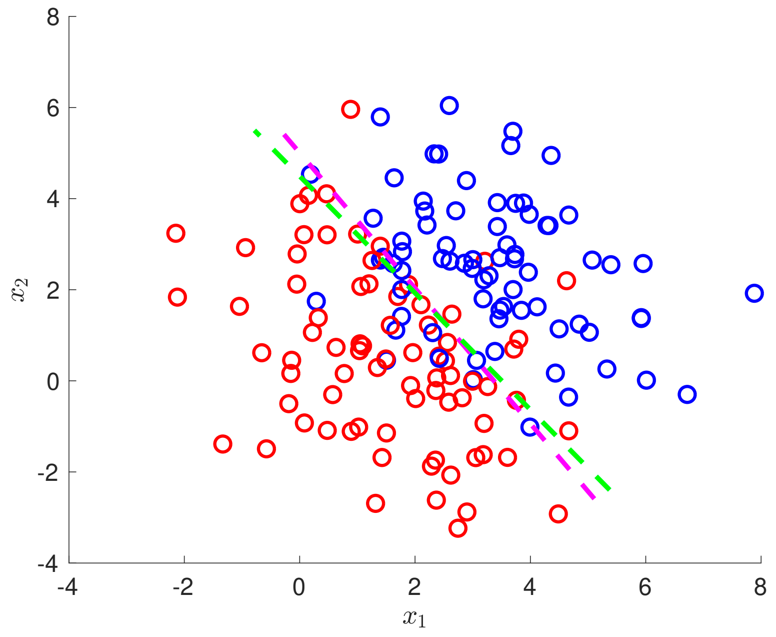

Figure 2: The dataset `data1.mat` with the green line $\{x \in \mathbf{R}^2 \colon s^T x = r\}$ superimposed, where (s, r) is the solution of problem (1) found by the gradient method (Task 2).

Task 3. Redo task 2 for the dataset `data2.mat`.

The dataset `data2.mat` is given in Figure 4.

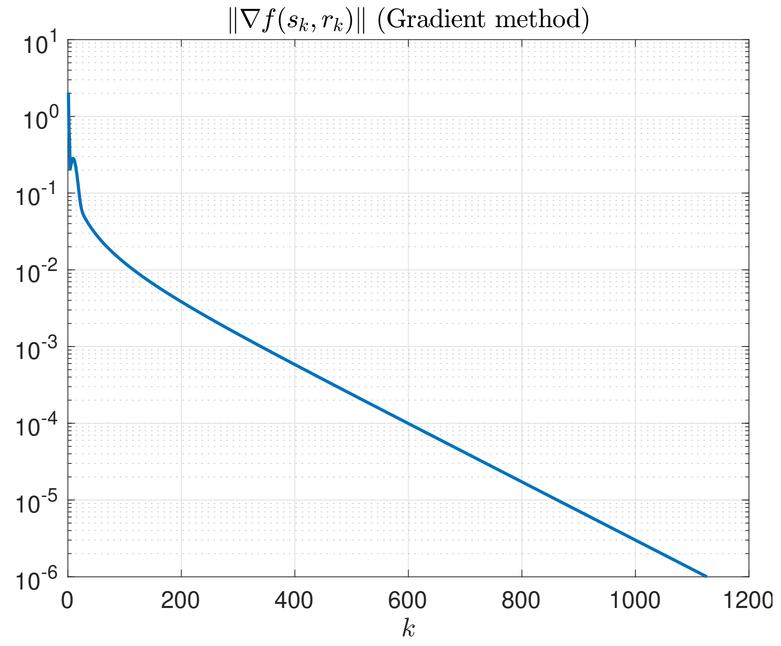

Figure 3: Norm of the gradient along the iterations of the gradient method, when the gradient method is applied to problem (1) with dataset `data1.mat` (Task 2).

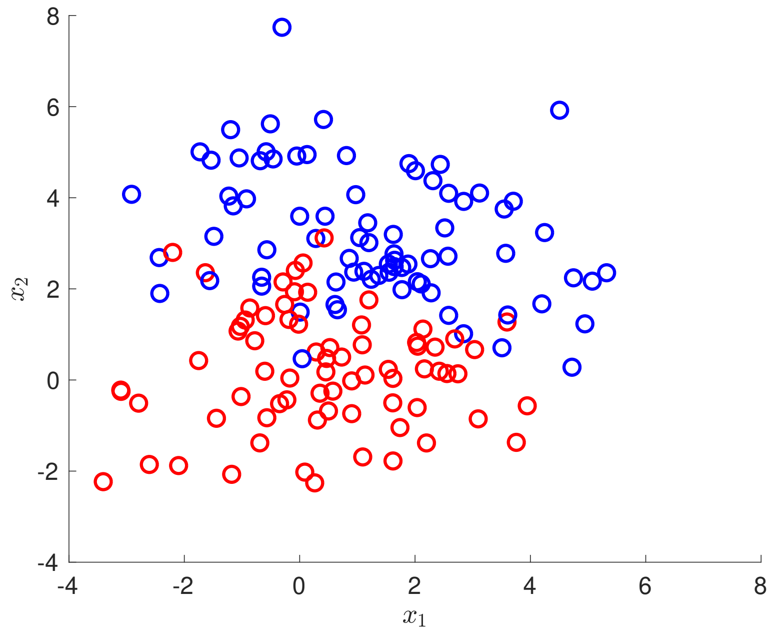

Figure 4: The dataset `data2.mat` for task 3.

Task 4. Use the gradient method to solve problem (1) for the datasets `data3.mat` (for which, $n = 30$ and $K = 500$) and `dataset4.m` (for which, $n = 100$ and $K = 8000$):

- Regarding slide 48, use the initialization $s_0 = (-1, -1, \ldots, -1)$ and $r_0 = 0$, and set $\epsilon = 10^{-6}$ for the stopping criterion;
- For the backtracking subroutine, use the values of $\widehat{\alpha}$, γ, and β given in slide 47.

1.2 Newton method

We now solve problem (1) with the Newton method given in slide 81 of the set of slides "Part 2: unconstrained optimization."

In each iteration of Newton's method, the main computation is solving a linear system of the form $Hd = -g$, where d is the Newton direction, and H and g are the Hessian and the gradient of the cost function at the current iterate. For an efficient implementation of Newton's method in MATLAB, both the gradient and Hessian should be expressed through matrix operations, to avoid time-consuming `for` loops (in fact, this also applies for the gradient method in the previous section: the gradient should be computed preferably without `for` loops). The next task helps finding compact matrix expressions for both the gradient and Hessian.

Task 5. Let $\phi : \mathbf{R} \to \mathbf{R}$ be a twice-differentiable function. Suppose the function $p : \mathbf{R}^3 \to \mathbf{R}$ is given by

$$p(x) = \sum_{k=1}^{K} \phi\left(a_k^T x\right),$$

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

(a) Show that the gradient of p at x is given by $\nabla p(x) = Av$, where $A = \begin{bmatrix} a_1 & a_2 & \cdots & a_K \end{bmatrix}$ and

$$v = \begin{bmatrix} \dot{\phi}\left(a_1^T x\right) \\ \dot{\phi}\left(a_2^T x\right) \\ \vdots \\ \dot{\phi}\left(a_K^T x\right) \end{bmatrix}.$$

(Note that $\dot{\phi}$ is the derivative of ϕ.)

(b) Show that the Hessian of p at x is given by $\nabla^2 p(x) = ADA^T$, where D is the diagonal matrix

$$D = \begin{bmatrix} \ddot{\phi}\left(a_1^T x\right) & & & \\ & \ddot{\phi}\left(a_2^T x\right) & & \\ & & \ddots & \\ & & & \ddot{\phi}\left(a_K^T x\right) \end{bmatrix}.$$

(Note that $\ddot{\phi}$ is the second derivative of ϕ.)

Task 6. Solve problem (1) for the datasets `data1.mat`, `data2.mat`, `data3.mat`, and `data4.mat` using the Newton method:

- Regarding slide 81, use the initialization $s_0 = (-1, -1, \ldots, -1)$ and $r_0 = 0$, and set $\epsilon = 10^{-6}$ for the stopping criterion;
- For the backtracking subroutine, use the values of $\widehat{\alpha}$, γ, and β given in slide 47.

For each dataset, plot the norm of the gradient of the cost function along iterations, and the values of the stepsizes (α_k) determined by the backtracking subroutine.

So you can check your code, we give the answers for the dataset `data1.mat` (which you should reproduce): Figure 5 plots the norm of the gradient along iterations, and Figure 6 plots the stepsizes found by the backtracking subroutine.

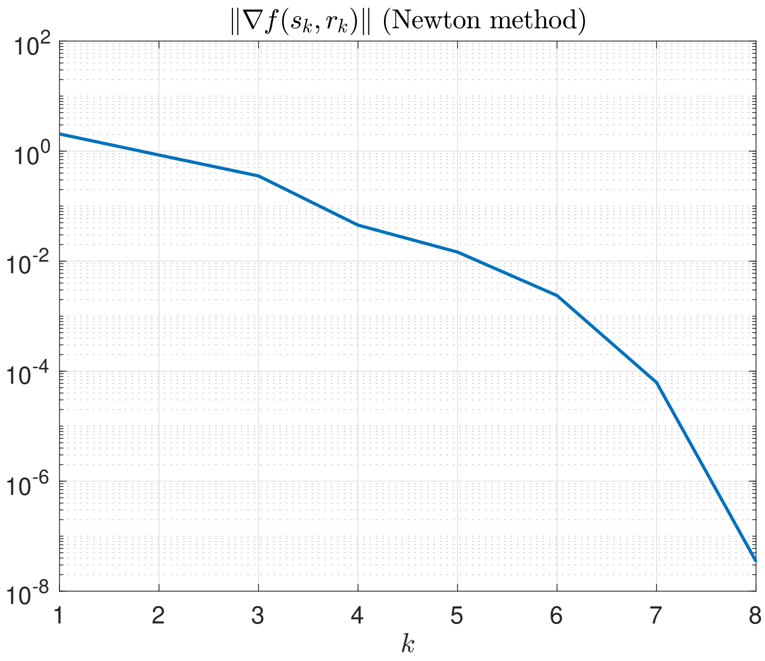

Figure 5: Norm of the gradient along the iterations of the Newton method, when the Newton method is applied to problem (1) with dataset `data1.mat` (Task 6).

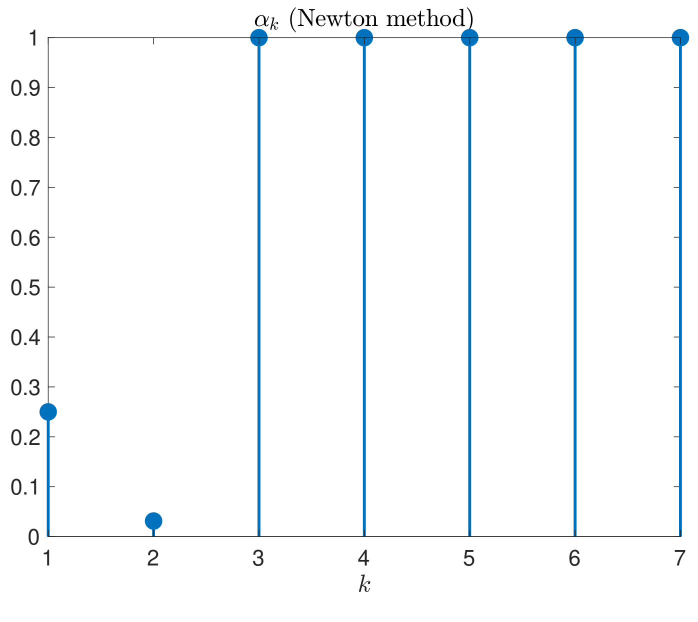

Figure 6: Values of the stepsizes along the iterations of the Newton method, when the Newton method is applied to problem (1) with dataset `data1.mat` (Task 6).

Task 7. Comment on all the results you obtained. In particular, compare the relative behaviour of the gradient and Newton method on problem (1).

2 Network localization

In network localization, we want to find the positions of several *sensors*, given noisy measurements of the distance between some of them and some anchors (sensors with known positions) and between some of the sensors.

Specifically, let $\{a_1, \ldots, a_M\}$ be the set of anchors, with the position of anchor m being $a_m \in \mathbf{R}^2$ (which we know). The set of sensors is denoted by $\{s_1, \ldots, s_P\}$, with the position of sensor p being $s_p \in \mathbf{R}^2$ (which we want to find). We are given two kinds of measurements: noisy measurements of the distance between some anchors and some sensors, of the form $y_{mp} = \|a_m - s_p\| +$"noise"; and noisy measurements between some sensors and other sensors, of the form $z_{pq} = \|s_p - s_q\| +$ "noise". See Figure 7 for an example of such a setup, with $M = 4$ anchors and $P = 8$ sensors.

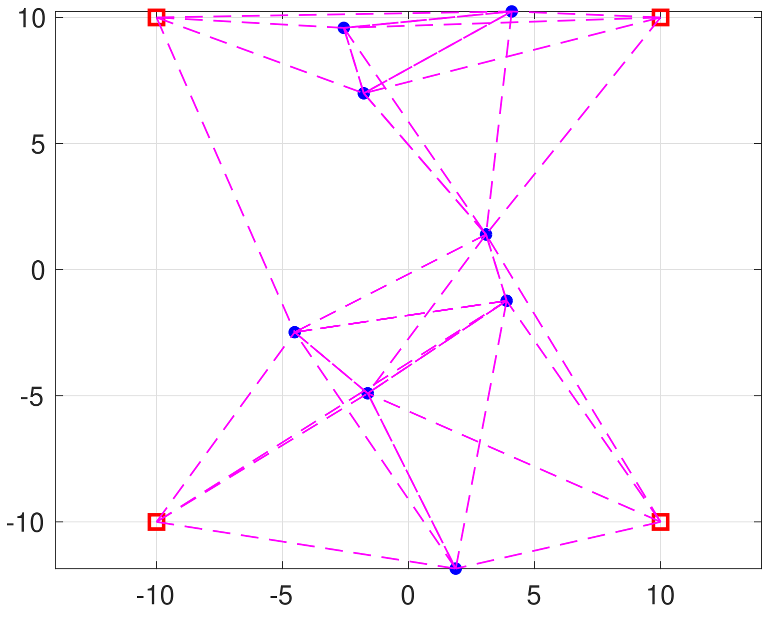

Figure 7: A network localization setup: anchors are the red squares; sensors are the blue dots. In network localization, the positions of the anchors are known; the positions of the sensors are unknown. We want to find the positions of the sensors, given noisy measurements of the distance between some sensors and some anchors, and between some sensors and some sensors. In this figure, each available measurement is represented by a magenta line: we have measurements between each anchor and four sensors, and between each sensor and three other sensors.

The optimization problem. Gather the unknown positions $s_1, \ldots, s_P$ of the sensors in the variable

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

Note that the column vector x is in $\mathbf{R}^{2P}$. Finding the sensors' positions can be posed as the following optimization problem:

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

where $\mathcal{A}$ is the set of pairs (anchor, sensor) for which we have noisy measurements of their distance, and $\mathcal{S}$ is the set of pairs (sensor, sensor) for which we have noisy measurements of their distance. For example, if we have a noisy measurement of the distance between anchor 3 and sensor 7 (which we denote by y_{37}), then the pair $(3, 7)$ is an element of the set $\mathcal{A}$; likewise, if we have a noisy measurement of the distance between sensor 2 and sensor 5 (which we denote by z_{25}), then the pair $(2, 5)$ is an element of the set $\mathcal{S}$.

2.1 Levenberg-Marquardt (LM) method

Task 8. Solve problem (2) using the LM method given in slide 90 of the set of slides "Part 2: unconstrained optimization," with parameters $\lambda_0 = 1$ and $\epsilon = 10^{-6}$.

The data for problem (2), which corresponds to the example in Figure 7, is in the file `lmdata1.mat`. This file contains the following matrices and vectors:

- the matrix `A` of size 2×4 contains the positions of the four anchors: column m of `A` is the position $a_m \in \mathbf{R}^2$ of anchor m;
- the matrix `iA` of size 16×2 represents the set $\mathcal{A}$, with each row of `iA` being a pair of indexes of the form (anchor, sensor); for example, the first row of `iA` is $(1, 4)$, which means that we have a noisy measurement of the distance between anchor 1 and sensor 4—this measurement is denoted y_{14} and is available in the next vector `y`;
- the vector `y` of length 16 contains the noisy measurements that correspond to the pairs (anchor, sensor) listed in the matrix `iA`; for example, the first entry of `y` gives y_{14};
- the matrix `iS` of size 24×2 represents the set $\mathcal{S}$, with each row of `iS` being a pair of indexes of the form (sensor, sensor); for example, the first row of `iS` is $(1, 2)$, which means that we have a noisy measurement of the distance between sensor 1 and sensor 2—this measurement is denoted z_{12} and is available in the next vector `z`;
- the vector `z` of length 24 contains the noisy measurements that correspond to the pairs (sensor, sensor) listed in the matrix `iS`; for example, the first entry of `z` gives z_{12};
- the vector `xinit` of length 16 contains the initialization for the LM method; thus, this vector, which is of the form
$$\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_P \end{bmatrix},$$
contains the initial guesses for the positions of all the sensors;
- the matrix `S` of size 2×8 contains the true positions of the sensors: column p of `S` is $s_p \in \mathbf{R}^2$, the position of sensor p. Of course, your algorithm should *never* access this matrix: the algorithm is trying to guess this matrix! We provide it only for your curiosity.

After running your code, plot the estimates of the sensor positions given by the LM method, and the norm of the gradient of the cost function along iterations. (So you can check your code, we provide these answers in figures 8 and 9, which you should reproduce.)

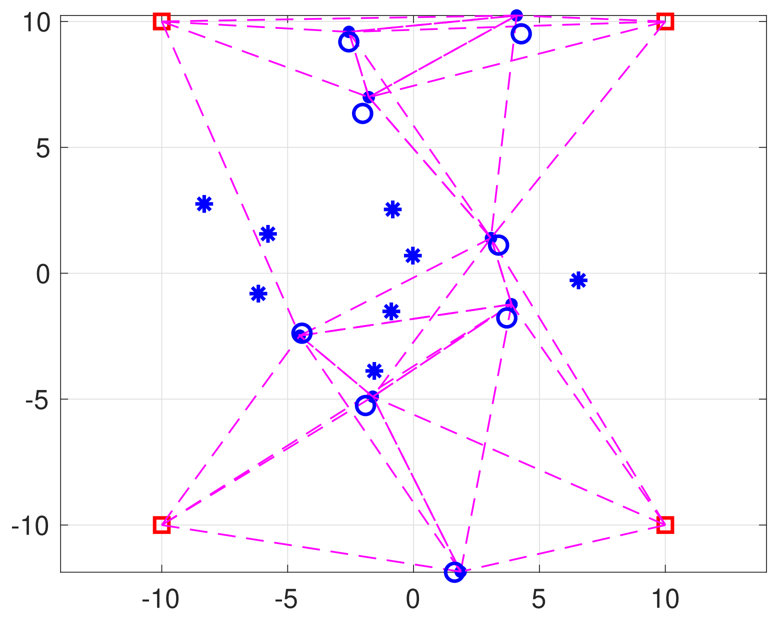

Figure 8: Network localization setup for task 8, described in the file `lmdata1.mat`: anchors are the red squares (matrix `A`); sensors are the blue dots (matrix `S`); the initialization for the LM method are the blue stars (vector `xinit`); from this initialization, the LM method converged to the blue circles.

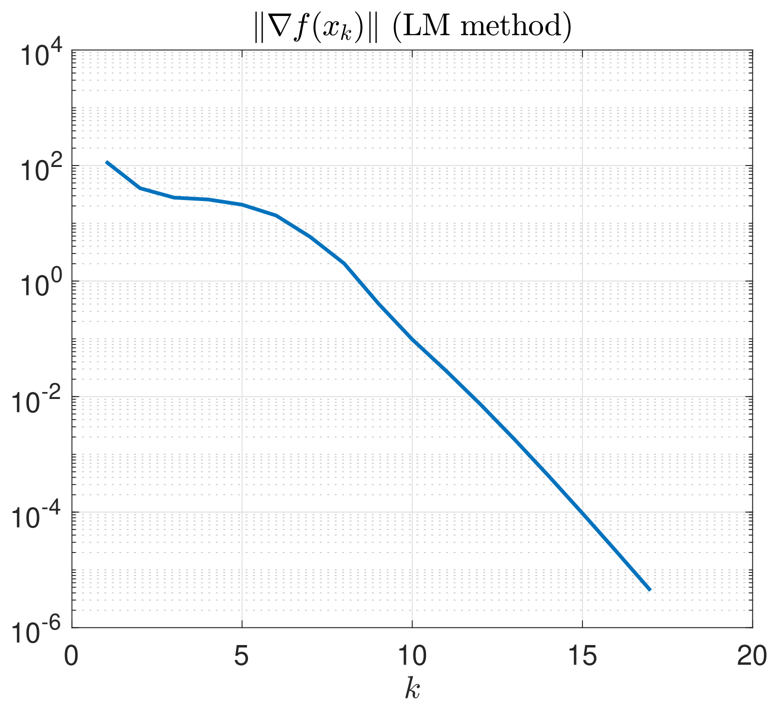

Figure 9: Norm of the gradient along the iterations of the LM method, when the LM method is applied to problem (2) with the data of file `lmdata1.mat` (Task 8).

Task 9. Solve problem (2) using the LM method given in slide 90 of the set of slides "Part 2: unconstrained optimization," with parameters $\lambda_0 = 1$ and $\epsilon = 10^{-6}$. The data for problem (2) is in the file `lmdata2.mat`; in this file, we neither provide an initialization nor the true positions of the sensors. You should run the LM method from several random initializations, thereby obtaining several solutions; report only the best solution—the one offering the smallest value of the cost function.