FTML practical session 5

20 avril 2025

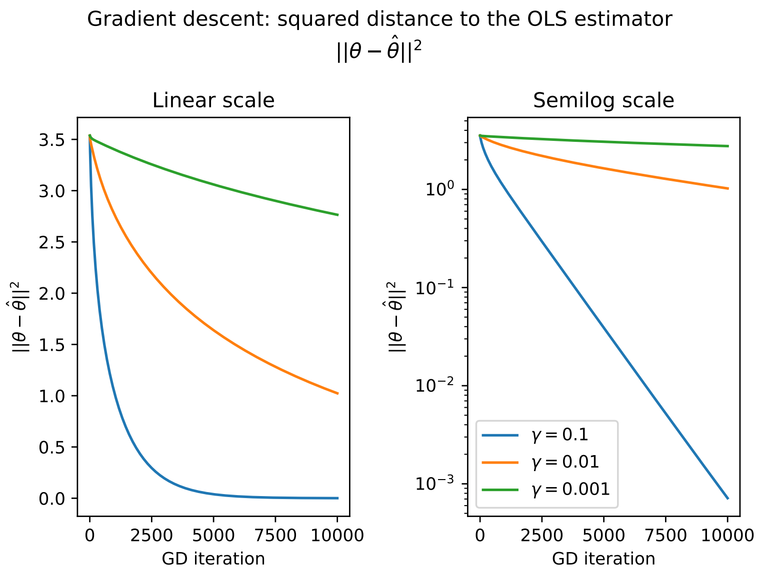

TABLE DES MATIÈRES

1 GRADIENT DESCENT AND LINE SEARCH FOR A LEAST SQUARES PROBLEM

We come back to a least squares problem, that we will use to show an example of gradient descent algorithm, with a non-constant learning rate.

1.1 Setting

1.1.1 *Notations*

We recall the setting and the notations here :

— $\mathcal{X} = \mathbb{R}^d$
— $\mathcal{Y} = \mathbb{R}$
— $\theta = \mathbb{R}^d$ defines the estimator, which is the function $x \to x^T\theta$.
— design matrix : $X \in \mathbb{R}^{n,d}$
— vector of outputs (labels) : $y \in \mathbb{R}^n$.

We want to minimize the function f representing the empirical risk :

$$f(\theta) = \frac{1}{2n}||X\theta - y||^2 \tag{1}$$

The $\frac{1}{2}$ is just convenient here because it disappears in the gradient.

1.1.2 *Gradient descent*

We recall that the gradient of f and its Hessian write :

$$\begin{aligned} \nabla_\theta f(\theta) &= \frac{1}{n}X^T(X\theta - y) \\ &= H\theta - \frac{1}{n}X^T y \end{aligned} \tag{2}$$

$$H = \frac{1}{n}X^T X \tag{3}$$

An **iteration** of the gradient algorithm writes :

$$\theta_{t+1} = \theta_t - \gamma\nabla_\theta f(\theta_t) \tag{4}$$

where $\gamma > 0$ is the learning rate.

1.1.3 *Global minimizers*

We consider the global minimizers of f, noted θ^*. They all necessary verify

$$\nabla_\theta f(\theta^*) = 0 \tag{5}$$

Which means that

$$H\theta* = \frac{1}{n}X^T y \tag{6}$$

As $\theta \to f(\theta)$ is convex, we know that this condition is also sufficient : any vector that cancels the gradient is a global minimum.
If H is not invertible, there might be several minimizers. However, if f is strongly convex, then H is invertible and θ^* is unique, we note it θ^*. We will consider such a case here. In general, H is a positive semi-definite matrix, and here we hence consider a case where it is definite positive.

1.2 Vanilla gradient descent

Implement a gradient descent (GD) on the data contained in **exercice_1_line_search/data/**, and experiment the learning rate in order to observe convergence or divergence.

Template files in the folder :

— **main.py**

— **algorithms.py** (to edit)

You should observe something like figure 1 in the cases of convergence.

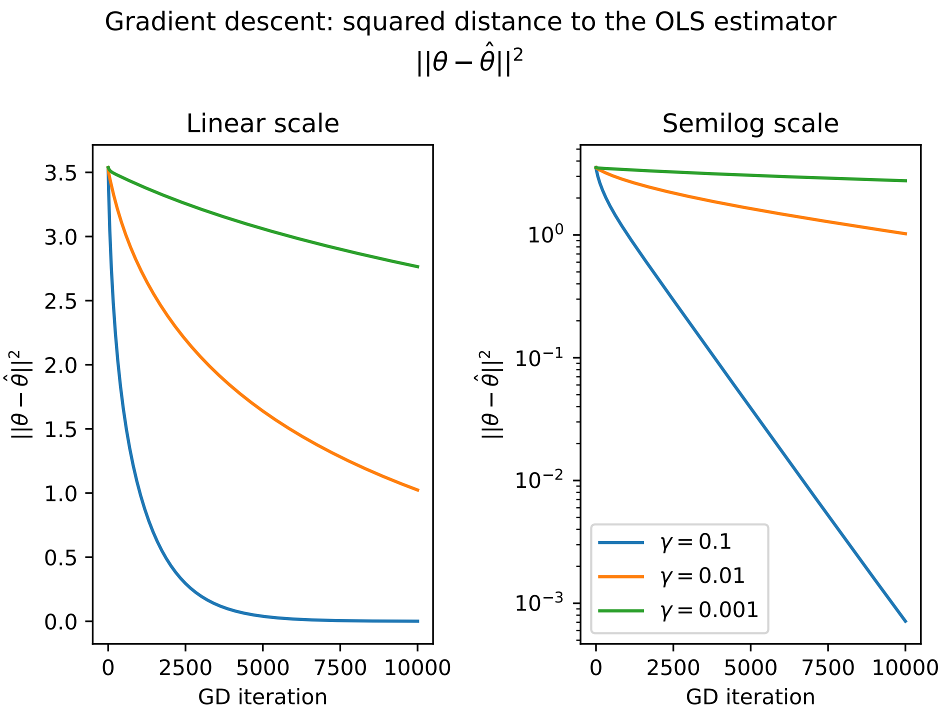

FIGURE 1 – Constant step size gradient descent and for several learning rates.

1.3 Line search : a nested optimization problem

Considering a fixed iteration step θ_t, we note

$$\alpha(\gamma) = \theta_t - \gamma \nabla_\theta f(\theta_t) \in \mathbb{R}^d \tag{7}$$

The **exact line seach** method attempts to find the optimal step γ^*, at each iteration. This means, given the position θ_t, the parameter γ that minimizes the function defined by

$$\begin{aligned} g(\gamma) &= f(\theta_t - \gamma \nabla_\theta f(\theta_t)) \\ &= f(\alpha(\gamma)) \end{aligned} \tag{8}$$

Is $g : \mathbb{R} \to \mathbb{R}$ a convex function ?

Find the value γ^* that minimizes $\gamma \to g(\gamma)$ for a given θ_t, and perform gradient descent where γ is set optimally thanks to this method (exact line search). Compare the convergence speeds by measuring the distance between the iterate and the OLS estimator $\hat{\theta}$ at each iteration.

You can uncomment the lines that call the line_search() function in the main file. You should observe something like figure 2

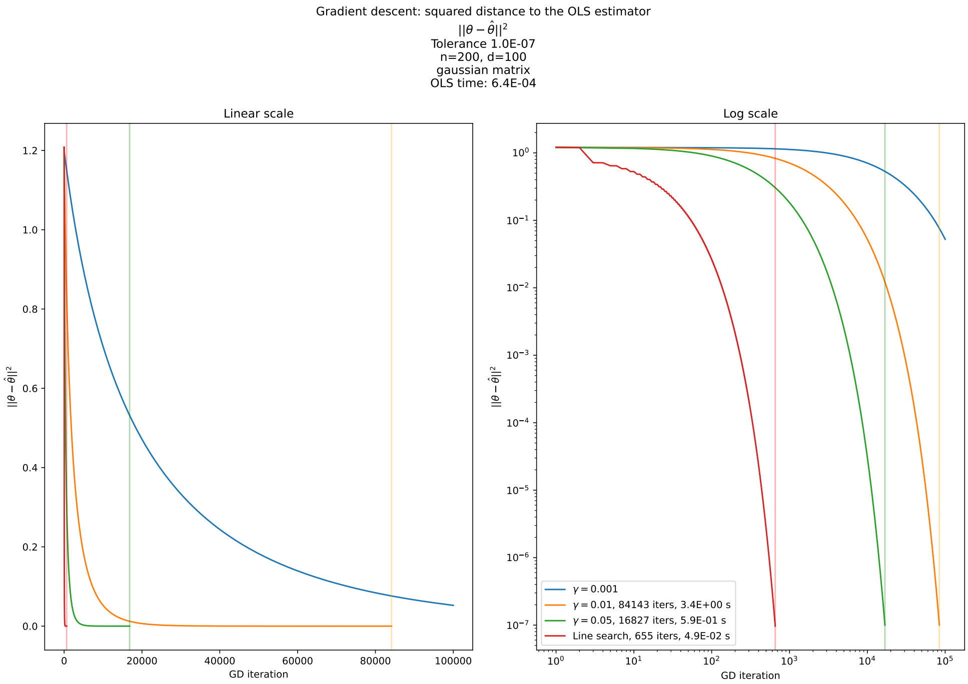

Figure 2 – Comparison between constant step size gradient descent and line search. In this image, we compare the number of iterations and running time to reach a value for $||\theta - \hat{\theta}||$ where $\hat{\theta}$ is the OLS estimator and θ the iterate. The right plot is also in full-log scale, as opposed to the previous plots. We note that for these dimensions, OLS solved analytically is way faster than GD and line search. For way larger dimensions (typically $d > 10^5$), the balance might be in favor of GD-type methods.

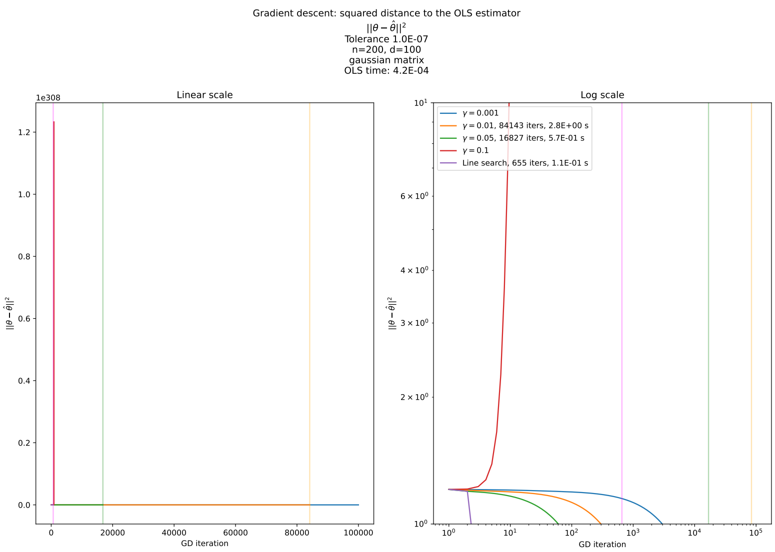

Figure 3 – If the learning rate is too large, we might have a divergence !

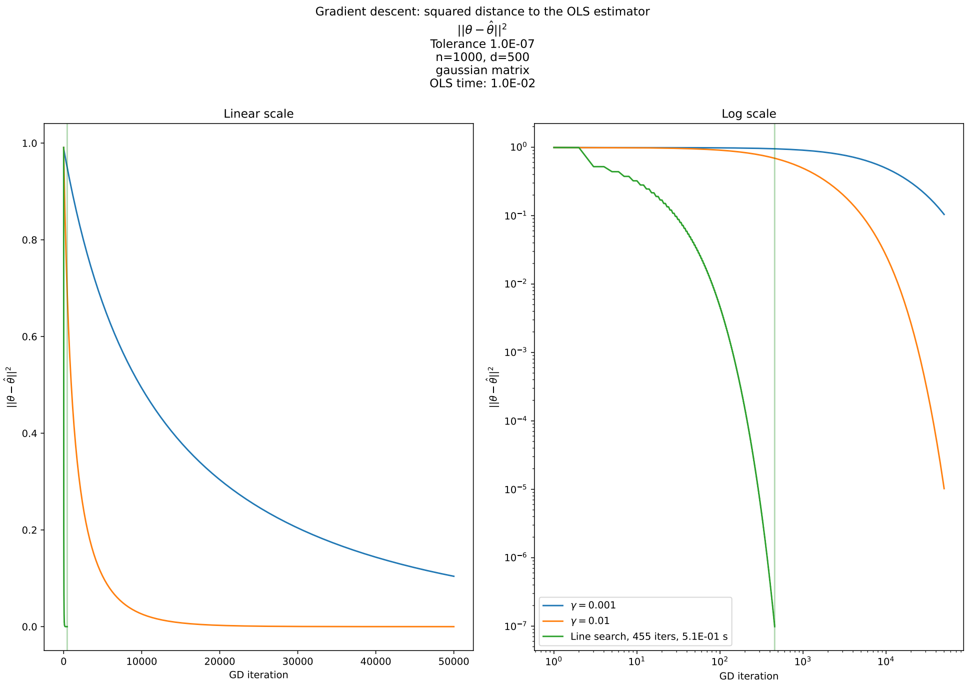

FIGURE 4 – Same image for larger dimensions of the problem.