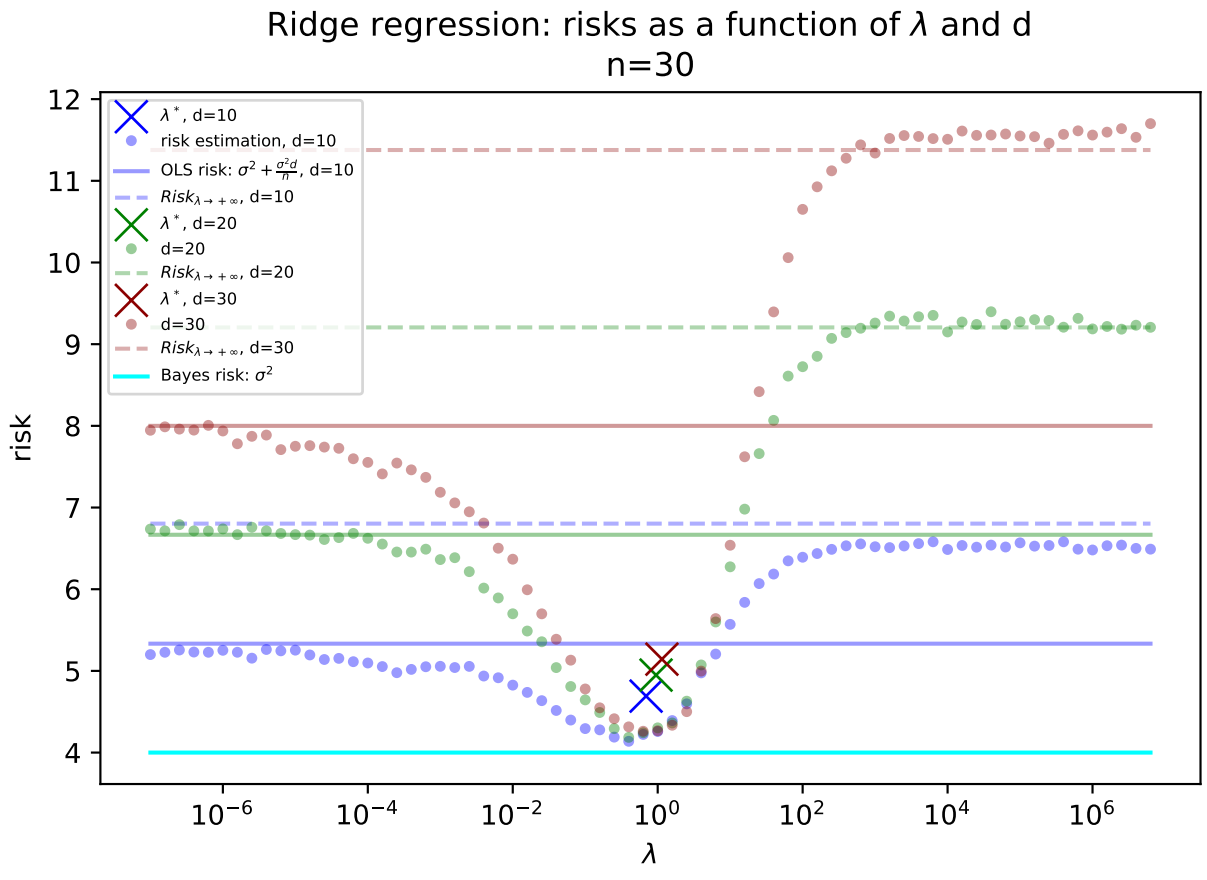
Fondamentaux théoriques du machine learning



Overview of lecture 4

Ridge regression

Gradient algorithms

Regularization of the empirical risk

In the previous practical session, we studied, Ridge regression, a form **regularization** of the objective function for the linear regression problem.
This enforces the unicity of the solution, even when $X^T X$ is not invertible (X is the design matrix like before) at the cost of introducing a **bias** in the estimator. The unicity is guaranteed by the **strong convexity** of the new loss function (studies in the next exercises).

Ridge regression estimator

We use the same notations as in the previous lectures.

- $X \in \mathbb{R}^{n,d}$
- $y \in \mathbb{R}^n$
- $\theta \in \mathbb{R}^d$

$$\hat{\theta}_\lambda = \underset{\theta \in \mathbb{R}^d}{\arg\min} \left(\frac{1}{n} ||y - X\theta||_2^2 + \lambda ||\theta||_2^2 \right) \tag{1}$$

with $\lambda > 0$.

Ridge regression estimator

Proposition

The Ridge regression estimator is unique even if $X^T X$ is not inversible and is given by

$$\hat{\theta}_\lambda = \frac{1}{n}(\hat{\Sigma} + \lambda I_d)^{-1} X^T y$$

with

$$\hat{\Sigma} = \frac{1}{n} X^T X \in \mathbb{R}^{d,d} \tag{2}$$

Statistical analysis of ridge regression

We can compare the excess risk (difference between risk and Bayes risk) to that of OLS.

Proposition

Under the linear model assumption, with fixed design setting, the ridge regression estimator has the following excess risk

$$E[R(\hat{\theta}_\lambda] - R^* = \lambda^2 \theta^{*T} (\hat{\Sigma} + \lambda I_d)^{-2} \hat{\Sigma} \theta^* + \frac{\sigma^2}{n} tr[\hat{\Sigma}^2 (\hat{\Sigma} + \lambda I_d)^{-2}] \quad (3)$$

Choice of λ

Is it possible that the excess risk is smaller with ridge regression than OLS ?

Proposition

With the choice

$$\lambda^* = \frac{\sigma\sqrt{tr(\hat{\Sigma})}}{||\theta^*||_2\sqrt{n}} \tag{4}$$

then

$$E[R(\hat{\theta}_\lambda] - R^* \leq \frac{\sigma\sqrt{tr(\hat{\Sigma})}||\theta^*||_2}{\sqrt{n}} \tag{5}$$

Choice of λ

Ridge

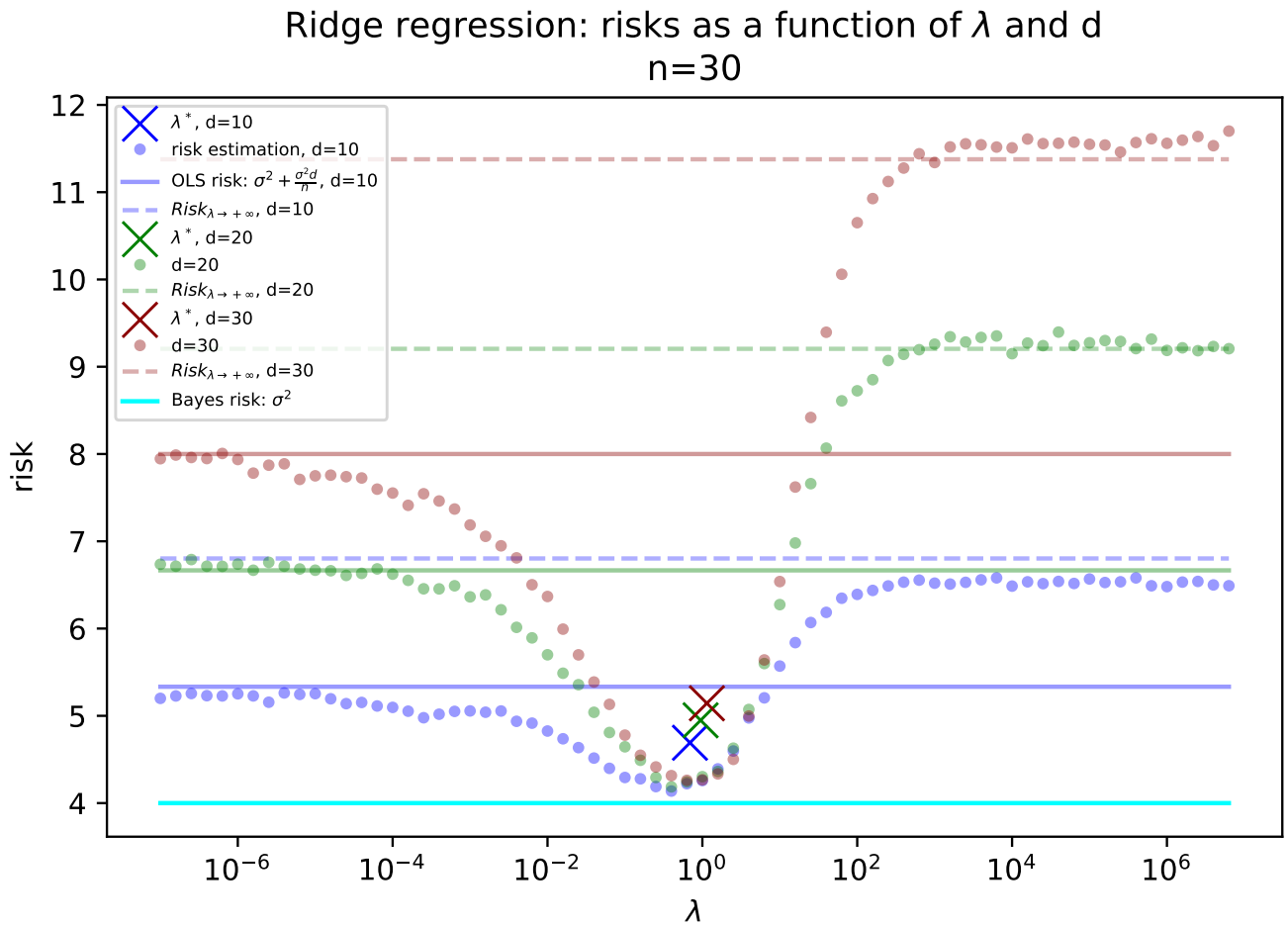
$$E[R(\hat{\theta}_\lambda] - R^* \leq \frac{\sigma\sqrt{tr(\hat{\Sigma})}||\theta^*||_2}{\sqrt{n}} \tag{6}$$

OLS

$$E[R(\hat{\theta}] - R^* = \sigma^2\frac{d}{n} \tag{7}$$

- $\frac{1}{n}$ (OLS) vs $\frac{1}{\sqrt{n}}$ (ridge), with different constants
- dimension-free bound for Ridge (maybe in the project)

Optimal λ



Hyperparameter search

- In practical situations, the quantities involved in the computation of λ^* in 4 are typically unknown. However this equation shows that there may **exist** a λ with a better prediction performance than OLS, which can be found by cross validation in practice (previous and next TP).
- λ is an example of **hyperparameter**.

Neural networks

With neural networks, it seems that it is possible to have $d >> n$ but no overfitting (simplicity bias).

Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions are not avaible, or where even if they are available, the necessary computation time is too large.

Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions are not avaible, or where even if they are available, the necessary computation time is too large.

Example 1 : Computing the OLS estimator requires a matrix inversion, which is $\mathcal{O}(d^3)$.

$$\hat{\theta} = (X^T X)^{-1} X^T y \tag{8}$$

High d might be prohibitive for a numerical resolution.

Context

In machine learning, we often encounter problems in high dimension, where closed-form solutions are not avaible, or where even if they are available, the necessary computation time is too large.

Example 2 : The cancellation of the gradient of the objective function with logistic loss has no closed-form solution.

Context

Instead, we often use **iterative** algorithm such as Gradient descent (GD) or Stochastic gradient descent (SGD). SGD is the standard optimization algorithm for large-scale machine learning because (brutal summary) :

- SGD is roughly n times faster than GD
- SGD's convergence is often satisfactory

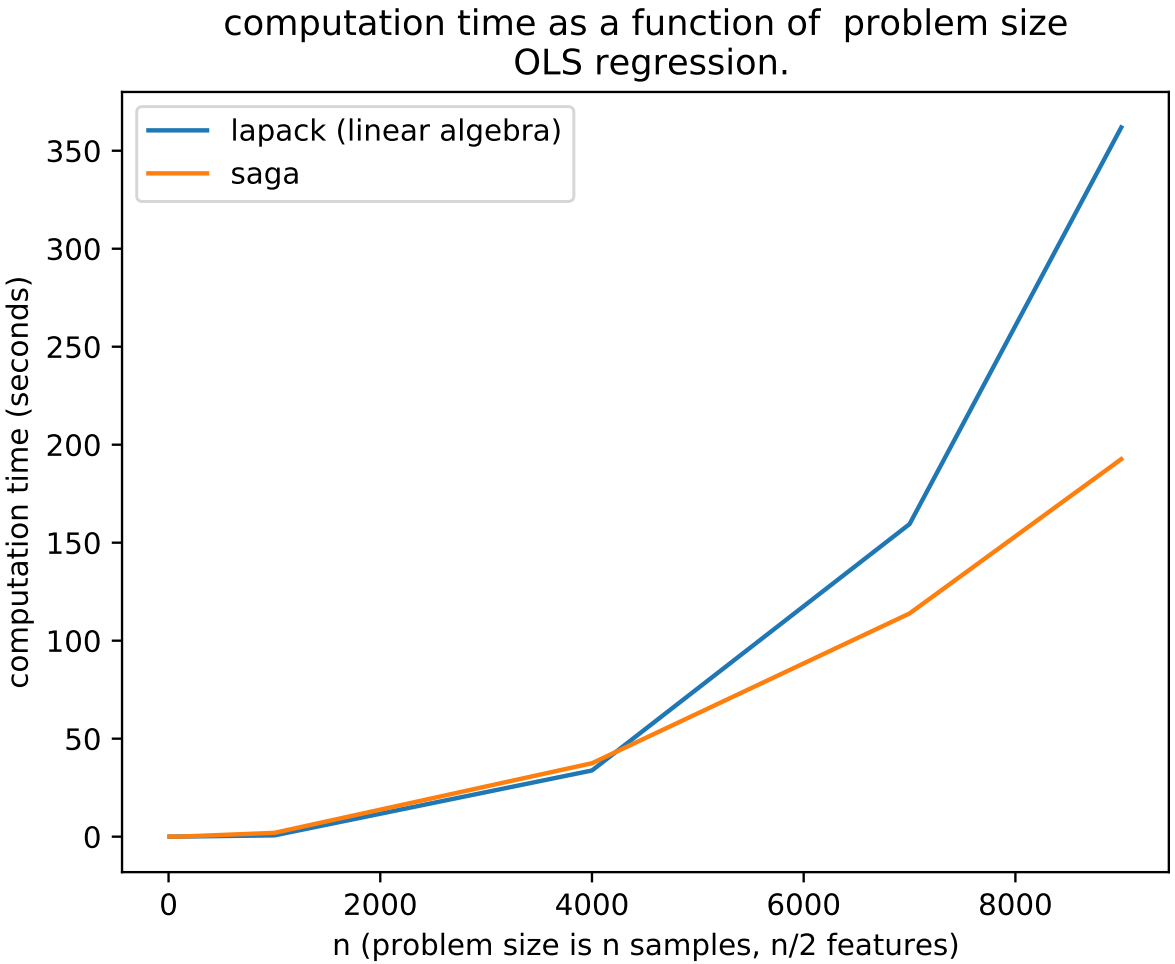
Convergence speed

The properties and speed of convergence of GD and SGD are important properties that are extensively studied. They typically depend on :

- the convexity or strong convexity of the objective function f (often the empirical risk).
- the regularity (smoothness constant) of the objective function f.
- the statistical properties of the dataset

We will study some of these aspects during the practical sessions.

SGD vs Lapack



Gradient descent

We want to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$. **Gradient descent** iteratively performs the following update, until some criterion is met.

$$\theta \leftarrow \theta - \gamma \nabla_f(\theta) \tag{9}$$

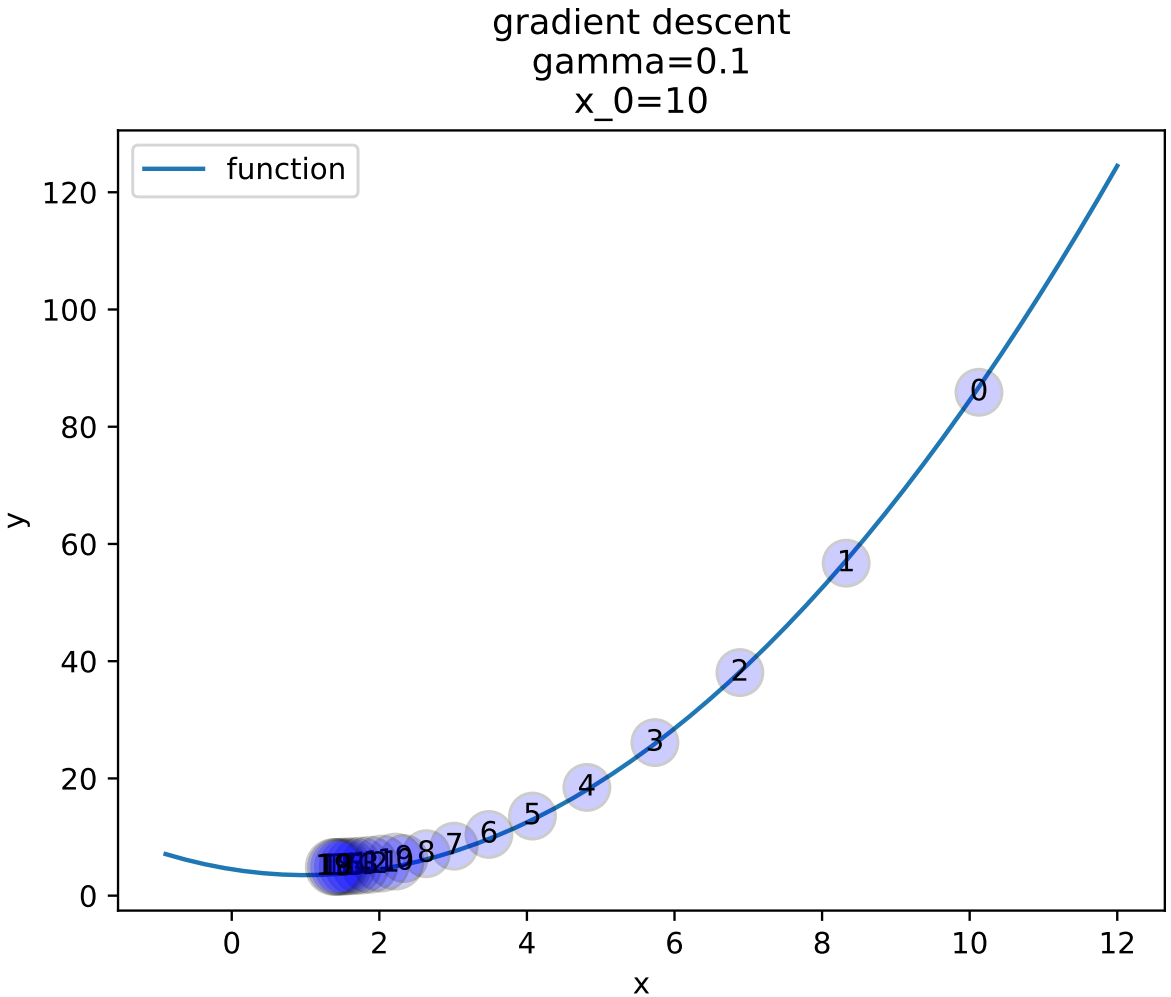
γ is called the **learning rate**, and is a very important hyperparameter. We will study some methods to tune it in practical sessions, like line seach.

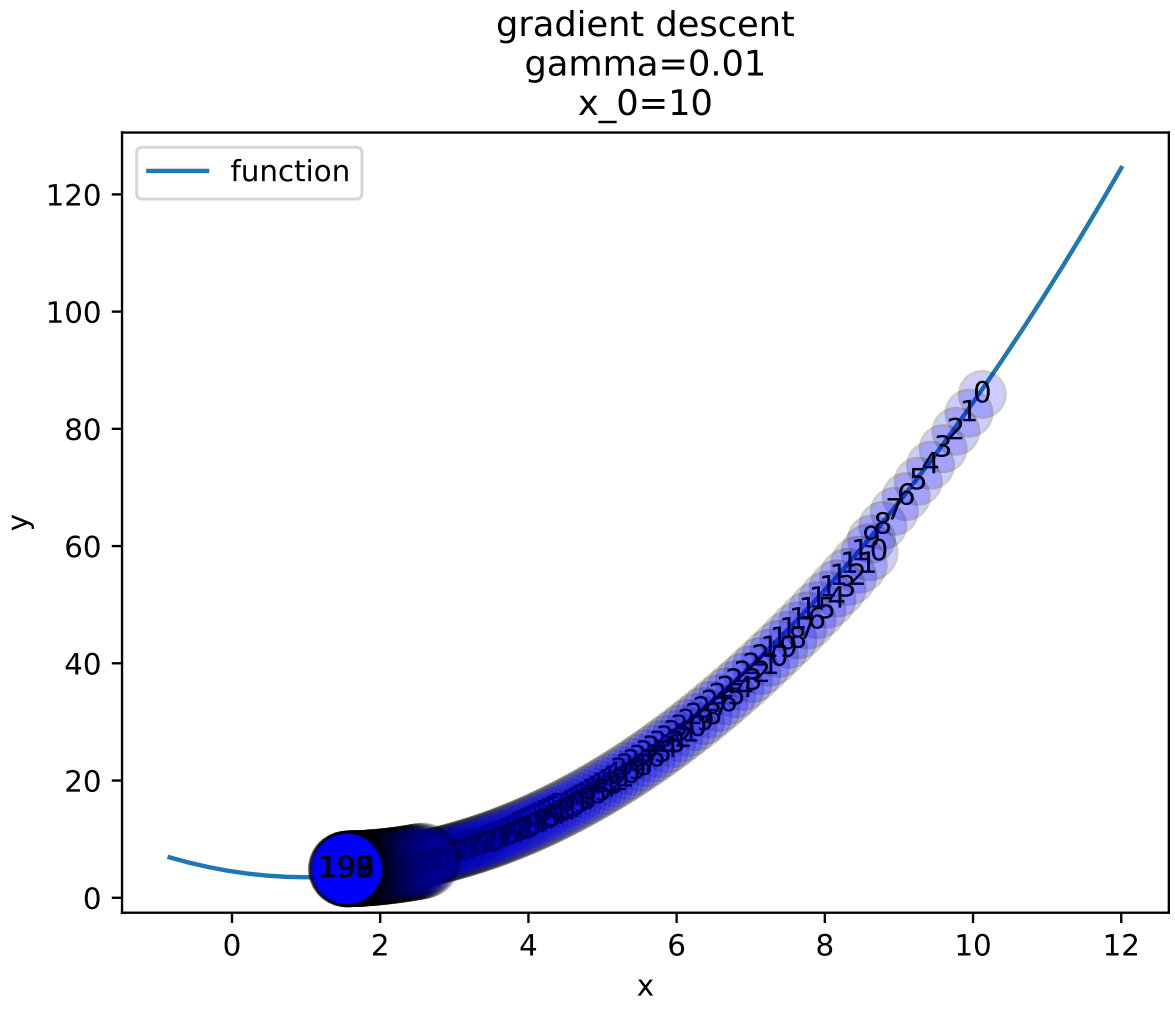
First-order method

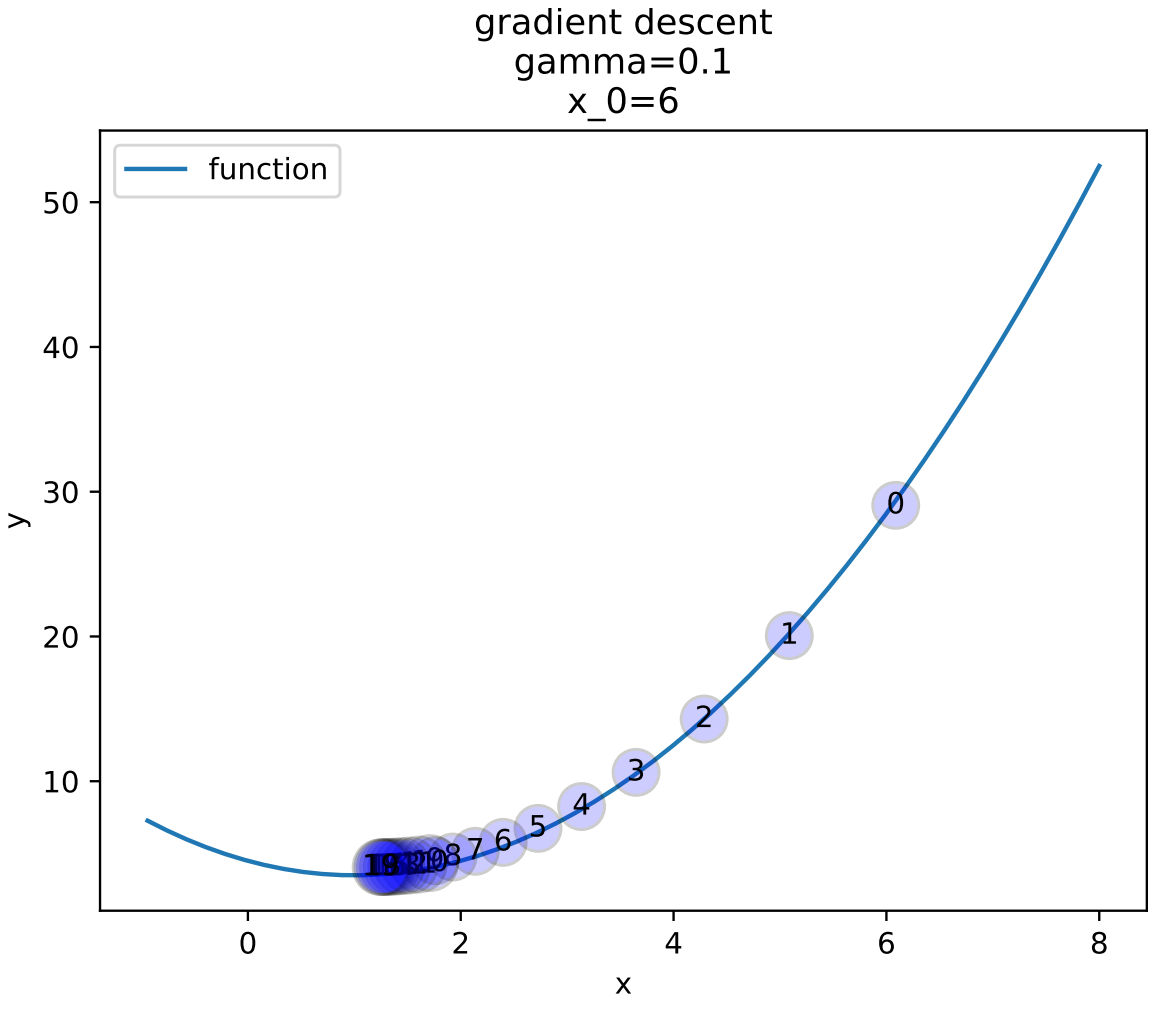
GD is called a first-order method because it makes use of the first order derivative.
The direction $-\nabla_\theta f(\theta)$ is the direction that minimizes the first order Taylor expansion of f in θ.

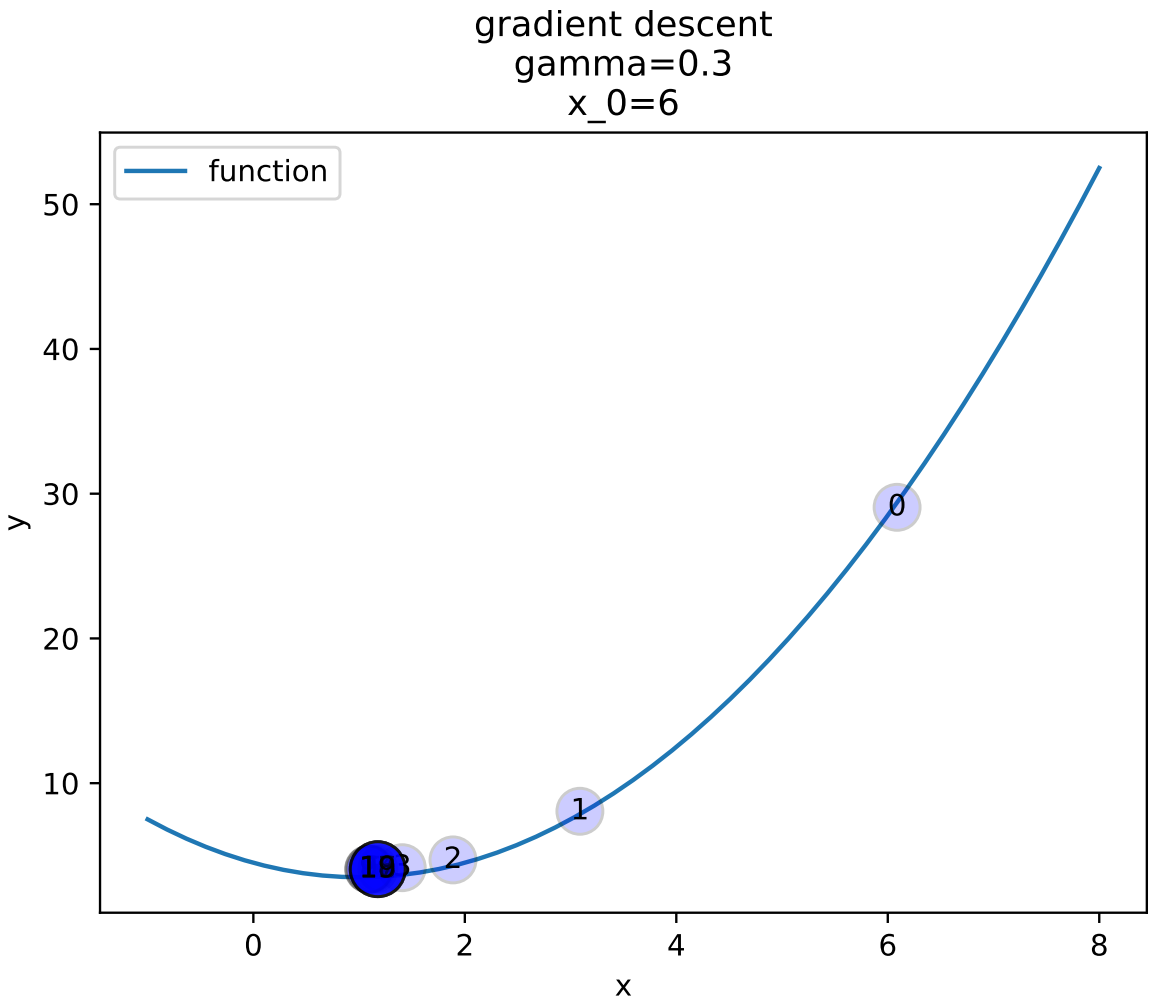
$$f(\theta + h) = f(\theta) + \langle \nabla_\theta f(\theta) | h \rangle + o(h) \tag{10}$$

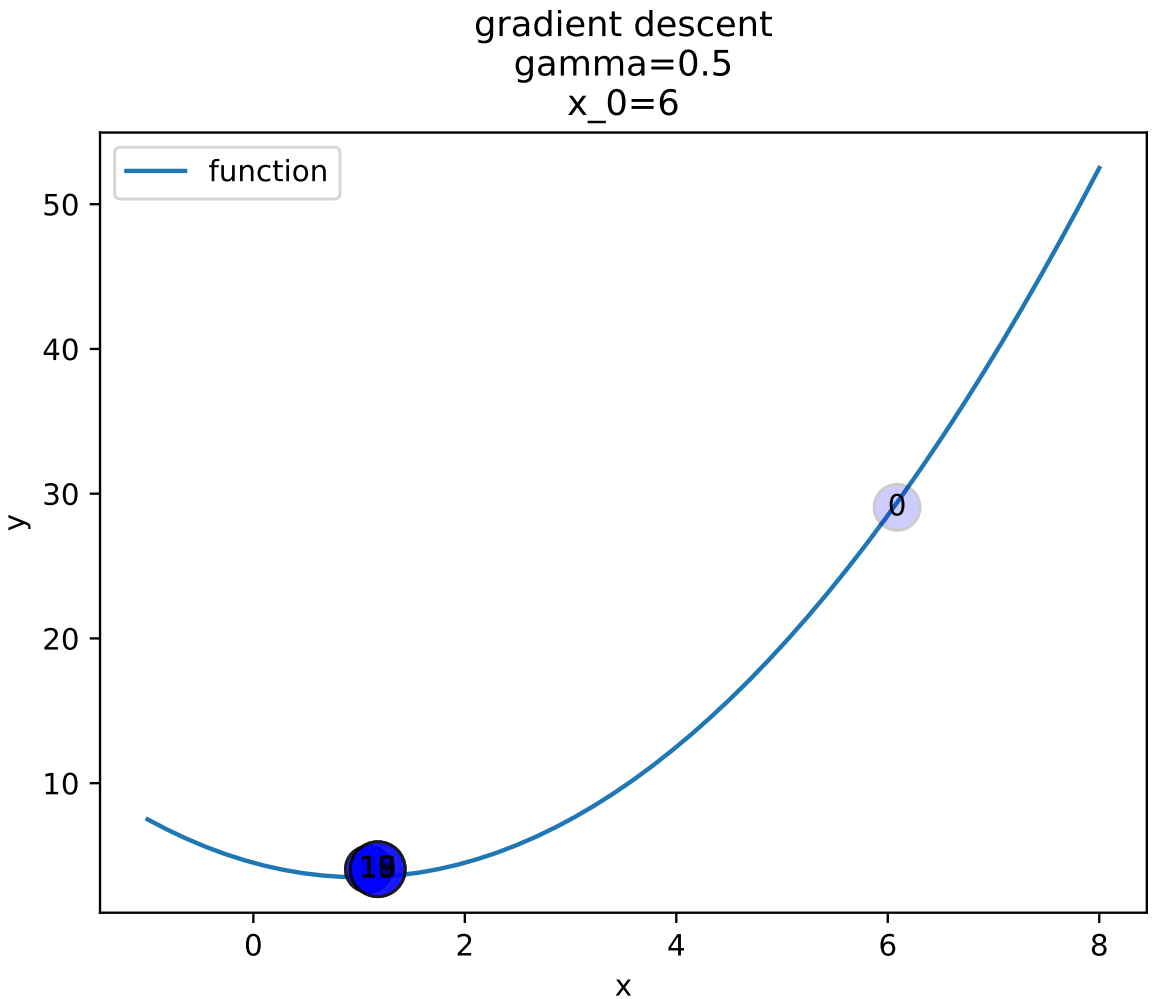
In the following examples, we use the function $f : x \to (x-1)^2 + 3$

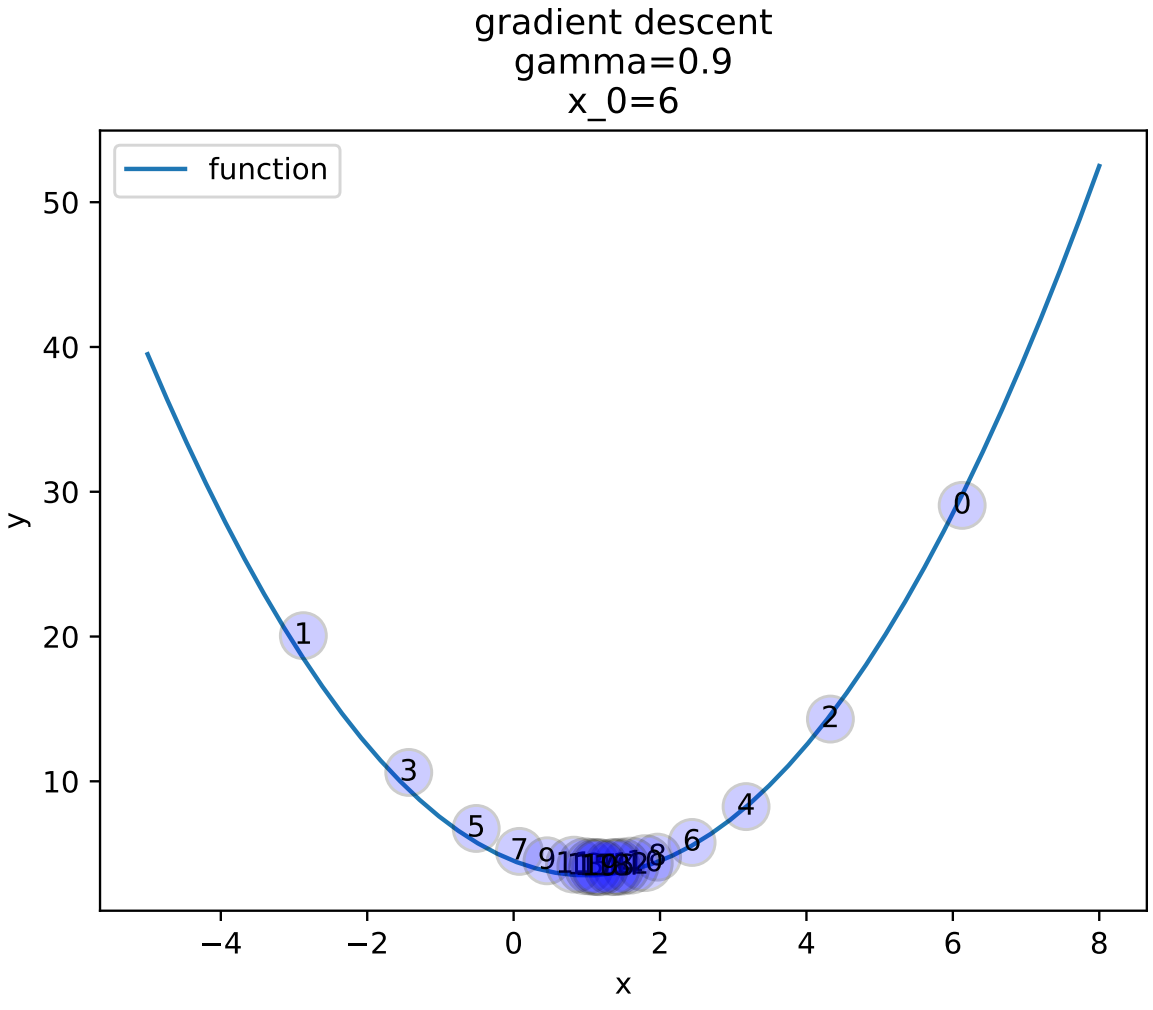


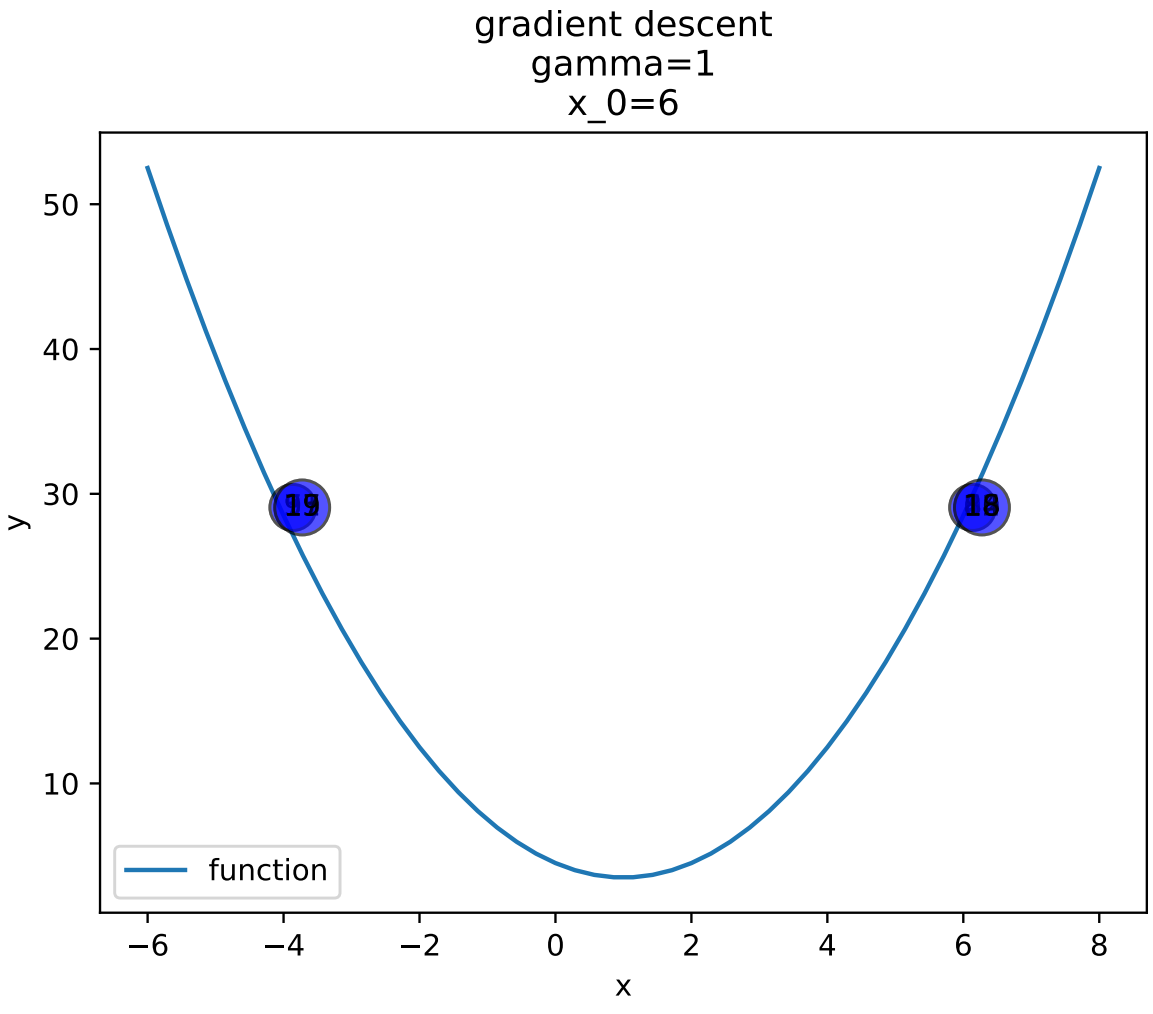

gradient descent
gamma=0.01
x_0=10
function
120
100
80
60
40
20
0
y
0
2
4
6
8
10
12
x

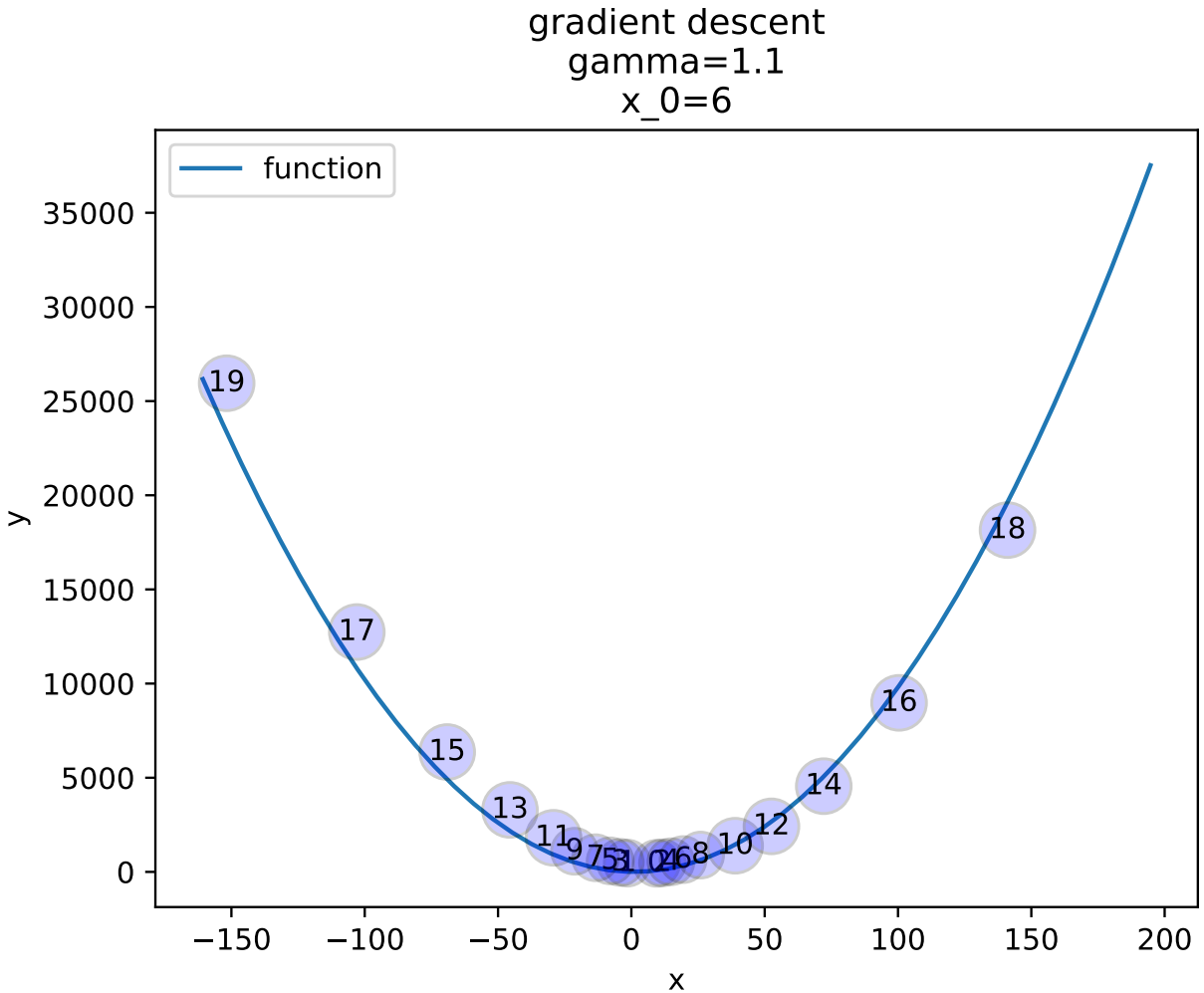

gradient descent
gamma=0.1
x_0=6
function
50
40
30
20
10
y
0
2
4
6
8
x
0
1
2
3
4
5
6
7

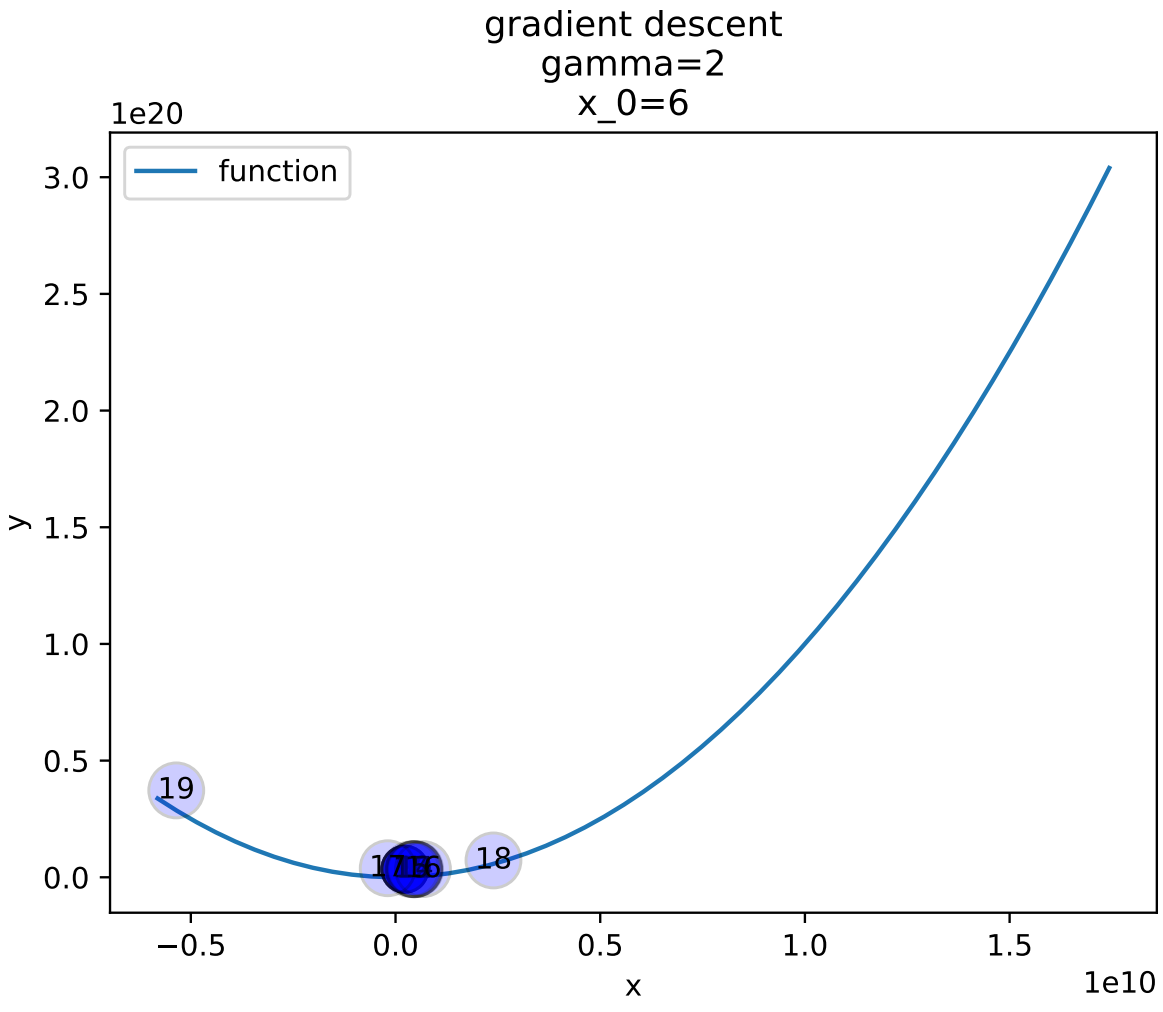

gradient descent
gamma=0.3
x_0=6
function
50
40
30
20
10
y
0
2
4
6
8
x
0
1
2


gradient descent
gamma=0.5
x_0=6
function
50
40
30
20
10
y
0
19
0
2
4
6
8
x


gradient descent
gamma=0.9
x_0=6
function
50
40
30
20
10
y
−4
−2
0
2
4
6
8
x
0
1
2
3
4
5
6
7

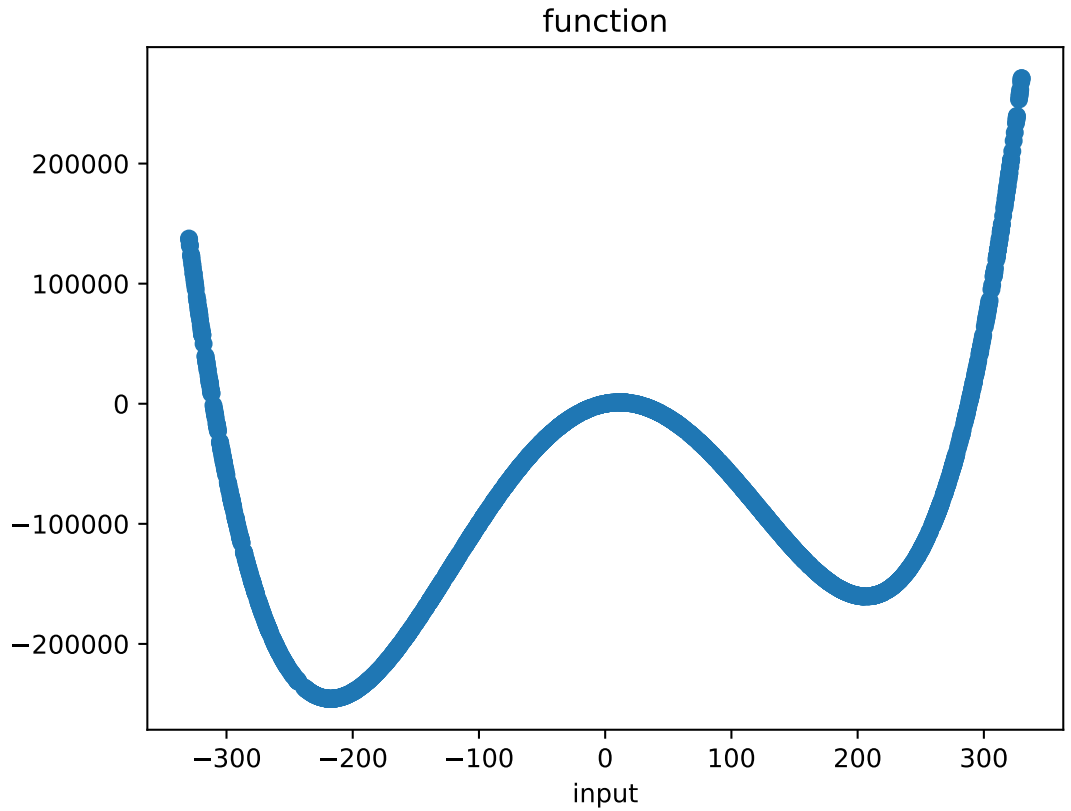

gradient descent
gamma=1
x_0=6
50
40
30
20
10
y
−6
−4
−2
0
2
4
6
8
x
function


gradient descent
gamma=1.1
x_0=6
function
35000
30000
25000
20000
15000
10000
5000
0
y
−150
−100
−50
0
50
100
150
200
x
19
17
15
13
11
9
7
5
3
1
0
2
4
6
8
10
12
14
16
18


gradient descent
gamma=2
x_0=6
1e20
function
3.0
2.5
2.0
1.5
1.0
0.5
0.0
y
19
18
−0.5
0.0
0.5
1.0
1.5
x
1e10

Local minima

GD is a greedy algorithm, and if f is non-convex, it might fall in a local minimum.



Stochastic gradient descent

In machine learning, we often consider an objective function of the form

$$f(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, g_\theta(x_i)) + \Omega(\theta) \tag{11}$$

Example :

- $\Omega(\theta) = \lambda||\theta||^2$.
- $l(y_i, g_\theta(x_i)) = ||y_i - \langle\theta|x_i\rangle||^2$

Batch gradient

In machine learning, we often consider an objective function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, g_\theta(x_i)) + \Omega(\theta) \tag{12}$$

Gradient descent computes the **batch gradient** (also called **full gradient**) of f, which requires at least n calculations, and each calculation also has a complexity that depends on the dimension d. When n and d are large, this might not be the best way to proceed.

SGD

In machine learning, we often consider an objective function of the form

$$f(\theta) = \frac{1}{n} \sum_{i=1}^{n} l(y_i, g_\theta(x_i)) + \Omega(\theta) \tag{13}$$

SGD replaces the full gradient $\nabla_\theta f(\theta)$ by :

$$\nabla_\theta \Big[l(y_i, g_\theta(x_i)) + \Omega(\theta) \Big](\theta) \tag{14}$$

with i being **randomly sampled** in $[1, n]$ (this is why it is called **stochastic**.)

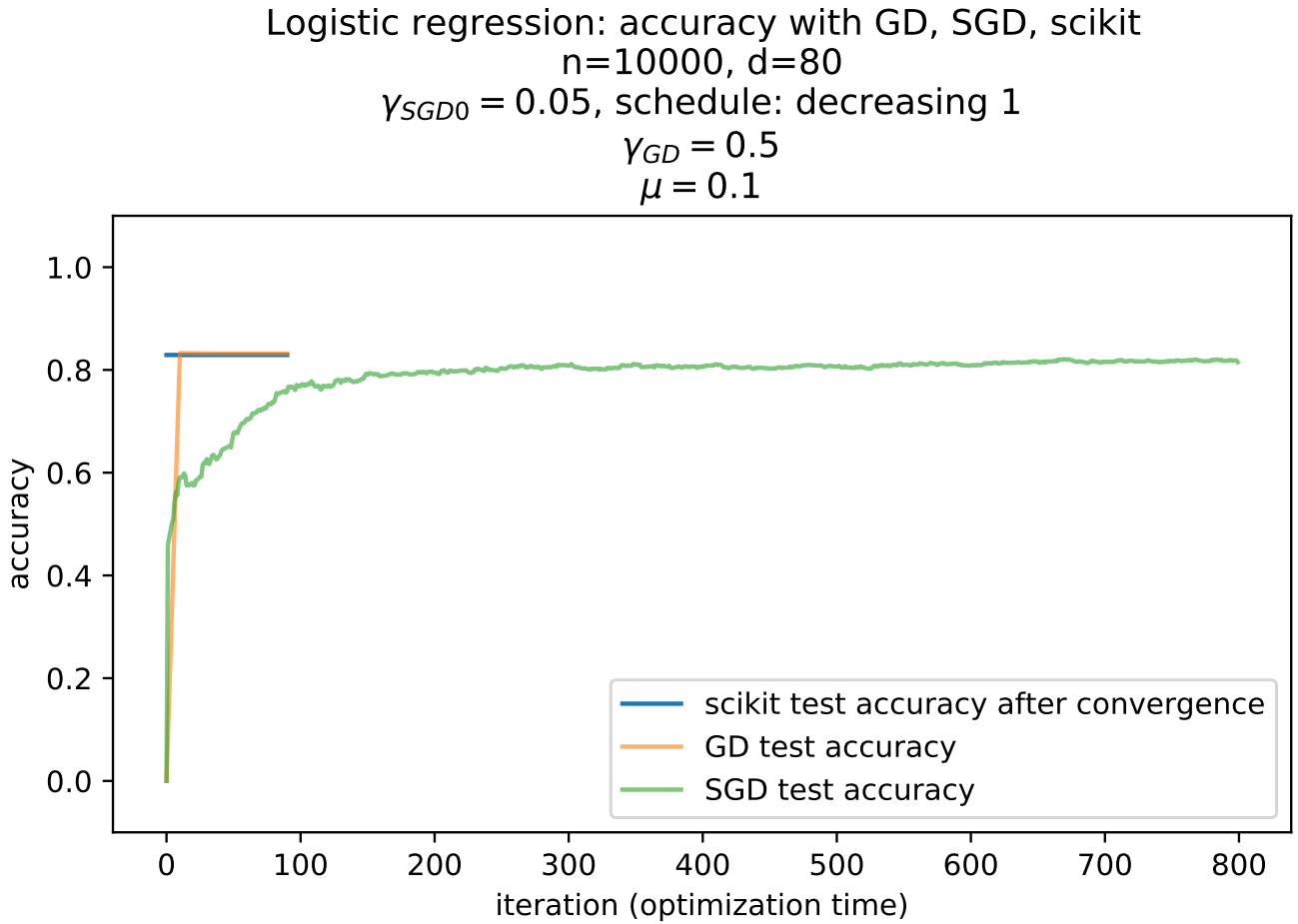


Figure – accuracy

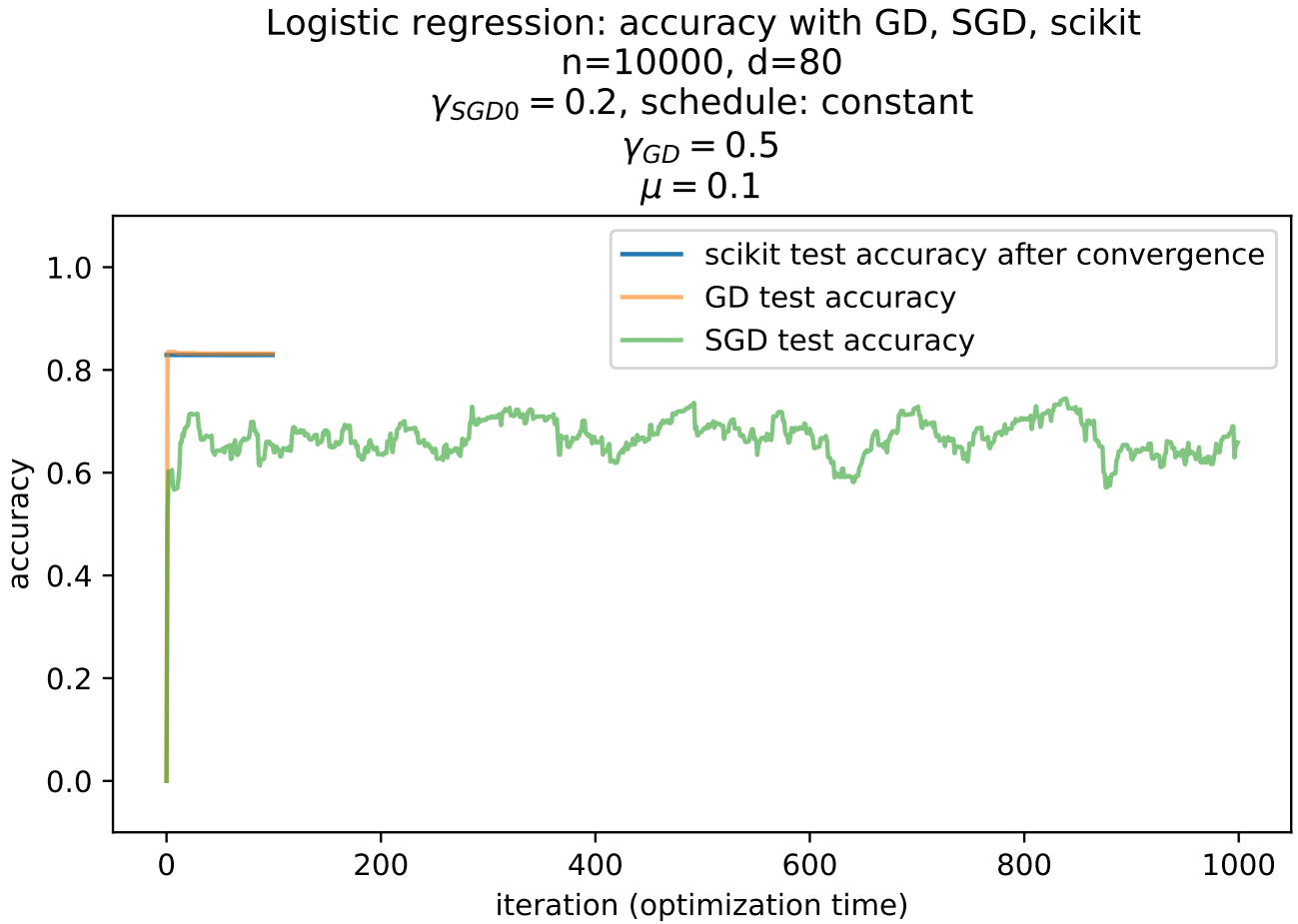


Figure – accuracy

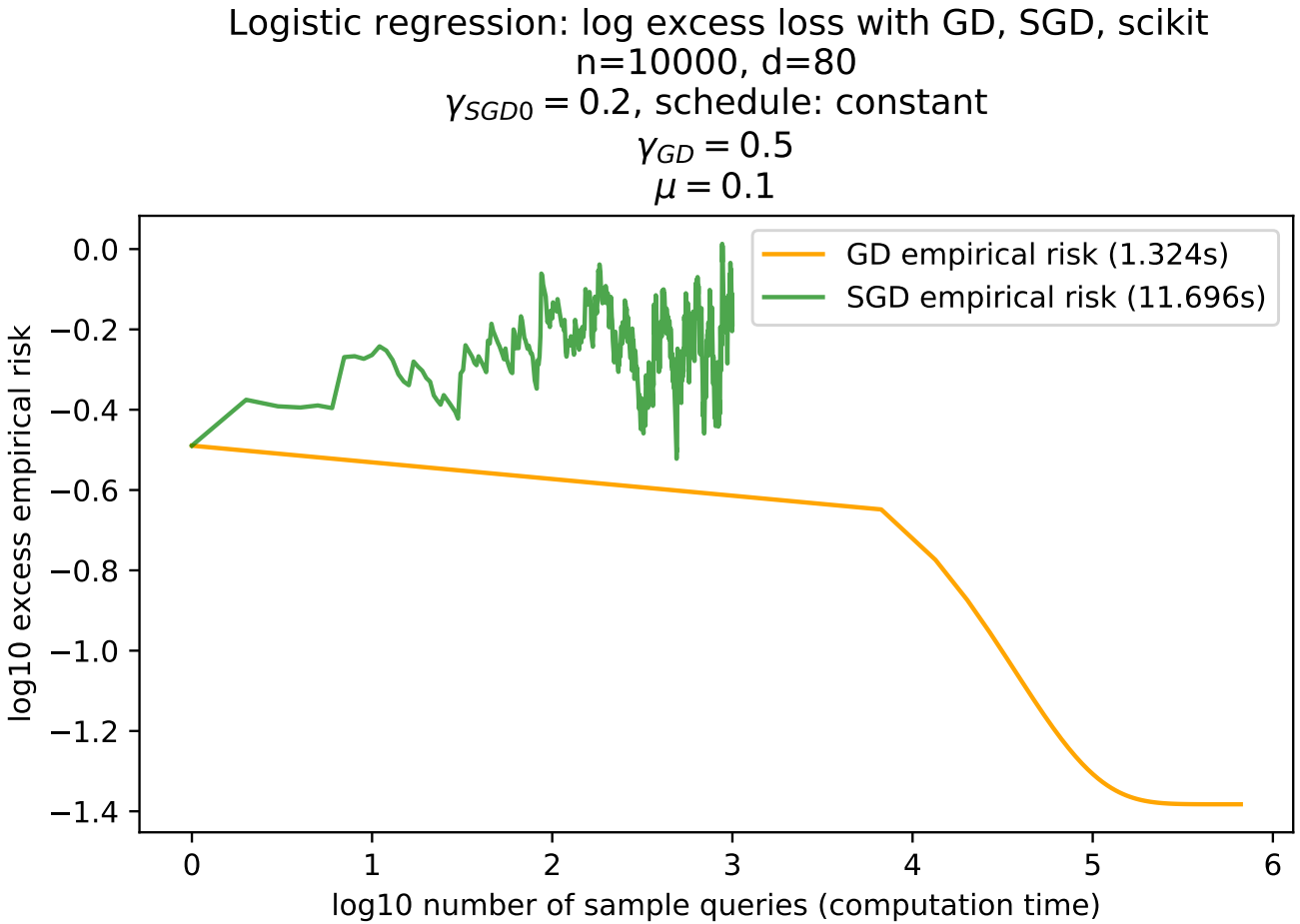


Figure – accuracy

References I