PTML 4: 15/04/2022

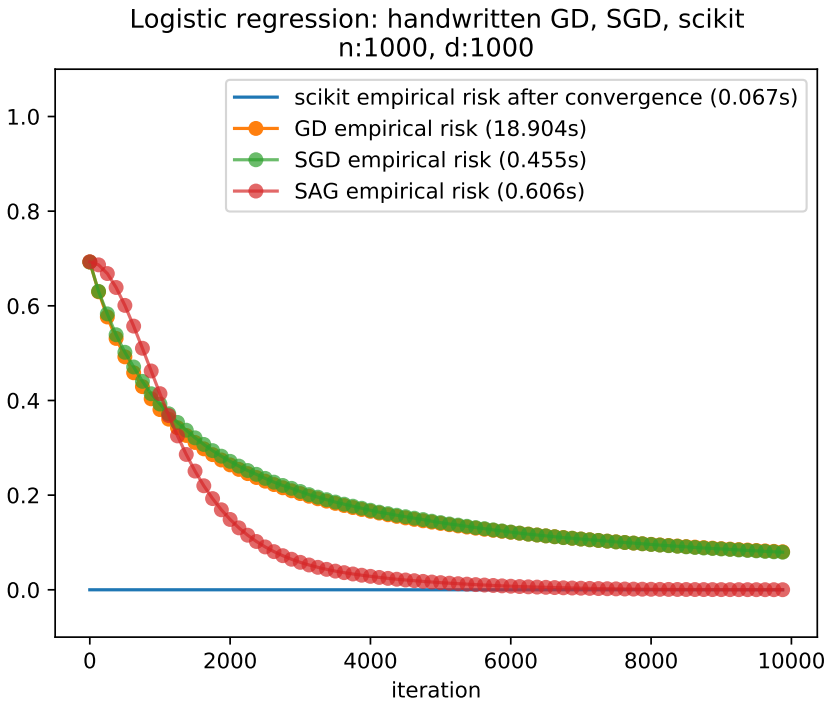


TABLE DES MATIÈRES

1 NUMPY MATRIX OPERATIONS

Run the notebook **np _matrix _operations.ipynb**. The goal is to emphasize the difference between matrix multiplication (produit matriciel) and element-wise multiplication (Hadamard product). Both are often useful.

— matrix multiplication is performed with **numpy.matmul**. https://numpy.org/doc/stable/reference/generated/numpy.matmul.html
"@" can be used as a shorthand.

— Hadamard multiplication is performed with **numpy.multiply**.
https://numpy.org/doc/stable/reference/generated/numpy.multiply.html
"*" can be used as a shofthand.
https://en.wikipedia.org/wiki/Hadamard_product_(matrices)

Conclusion : be careful with the shapes of the input arrays that you give to these methods !

2 GRADIENT DESCENT ON A LEAST-SQUARES PROBLEM

In this section the setting is the same as in the first section of TP3. If you prefer you can do part 3 before.

2.1 The heavy-ball method

When κ is very large, the convergence might become very slow. Some methods exist in order to speed it up, such as **Heavy-ball.** This method consists in adding a **momentum term** to the gradient update term, such as the iteration now writes

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

The update $\theta_{t+1} - \theta_t$ is then a combination of the gradient $\nabla_{\theta_t} f$ and of the previous update $\theta_t - \theta_{t-1}$. The goal of this method it might balance the effet of oscillations in the gradient.

We will use these parameters :

$$\gamma = \frac{4}{(\sqrt{L} + \sqrt{\mu})^2} \tag{2}$$

and

$$\beta = \left(\frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}\right)^2 \tag{3}$$

The heavy-ball method is called an *inertial method.* When f is a general convex function (not necessary quadratic), some generalizations exist, such as **Nesterov acceleration.**

2.1.1 *Impact on convergence rate*

Assuming $\mu > 0$, we will show that the characteristic convergence time with the heavy-ball momentum term is $\sqrt{\kappa}$ instead of κ.

Let λ be an eigenvalue of H and u_λ a eigenvector for thie eigenvalue. We are interested in the evolution of $\langle \theta_t - \eta^*, u_\lambda \rangle$.

We note

$$a_t = \langle \theta_t - \eta^*, u_\lambda \rangle \tag{4}$$

Exercice 1 : Show that

$$a_{t+1} = (1 - \gamma\lambda + \beta)a_t - \beta a_{t-1} \tag{5}$$

Exercice 2 : Compute the constant-recursive sequence a_t, and show that there exists a constant C_λ that depends on the initial conditions (trough A and B, and a_0), such that

$$\forall t, a_t \leqslant (\sqrt{\beta})^t C_\lambda \tag{6}$$

https://en.wikipedia.org/wiki/Constant-recursive_sequence

If u_i is a basis of orthogonal normed vectors with eigenvalues λ_i, we have that

$$\begin{aligned} \|\theta_t - \eta^*\|^2 &= \sum_{i=1}^{d} (\langle \theta_t - \eta^*, u_i \rangle)^2 \\ &\leqslant \sum_{i=1}^{d} (\sqrt{\beta})^{2t} C_{\lambda_i} \\ &= (\sqrt{\beta})^{2t} D \end{aligned} \tag{7}$$

with

$$D = \sum_{i=1}^{d} C_{\lambda_i} \tag{8}$$

We can now remark that

$$\begin{aligned} \sqrt{\beta} &= \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}} \\ &= \frac{1 - \sqrt{\frac{\mu}{L}}}{1 + \sqrt{\frac{\mu}{L}}} \\ &\leqslant 1 - \sqrt{\frac{\mu}{L}} \\ &= 1 - \frac{1}{\sqrt{\kappa}} \end{aligned} \tag{9}$$

Finally, as $1 - \frac{1}{\sqrt{\kappa}} \leqslant \exp(-\frac{1}{\sqrt{\kappa}})$,

$$\|\theta_t - \eta^*\|^2 = \mathcal{O}(\exp(-\frac{2t}{\sqrt{\kappa}})) \tag{10}$$

Conclusion : with the heavy-ball momentum term, we changed the convergence rate of $\mathcal{O}(\exp(-\frac{2t}{\kappa}))$ to a convergence rate of $\mathcal{O}(\exp(-\frac{2t}{\sqrt{\kappa}}))$. This means that characteristic convergence time went from κ to $\sqrt{\kappa}$. If κ is large, which is the case we are interested in, this can be a great improvement.

Remember that $\kappa = \frac{L}{\mu}$, and that μ may be very small when n or d is large. For instance, in the case of Ridge regression, we have seen in the previous session that for instance, μ can be of order $\mathcal{O}(\frac{1}{\sqrt{n}})$ (see the computation of the optimal regularisation parameter). Hence, κ may be of order $\sqrt{n}$ or higher.

2.1.2 *Simulation*

Exercice 3 : Use the file **TP4_heavy_ball.py** to implement the Heavy-ball method and compare the convergence speed results to that of GD. You should obtain something like figures 1 and 2.

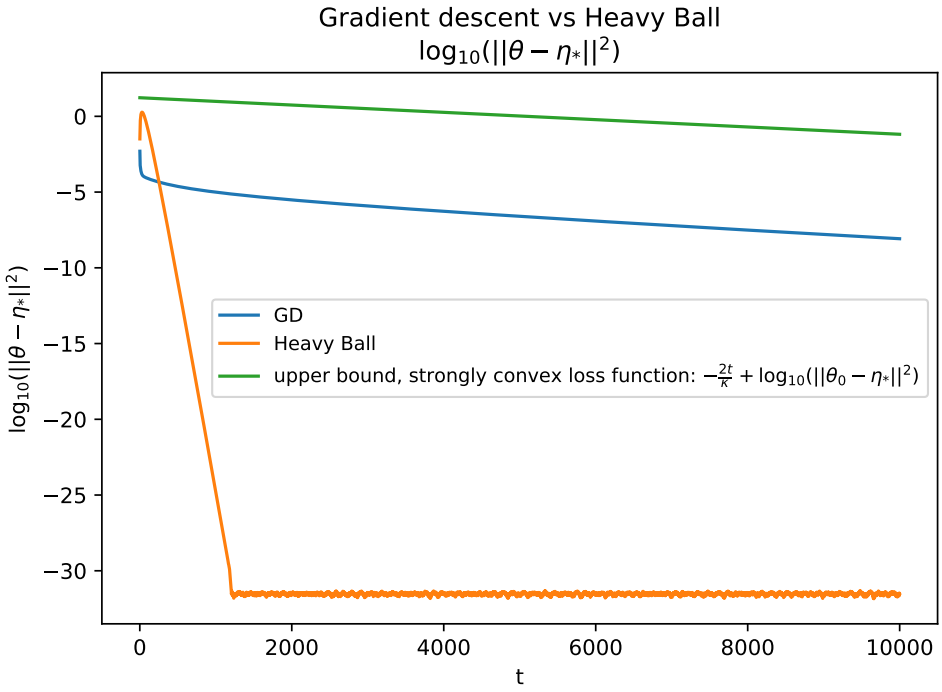


FIGURE 1 – Heavy-ball vs GD

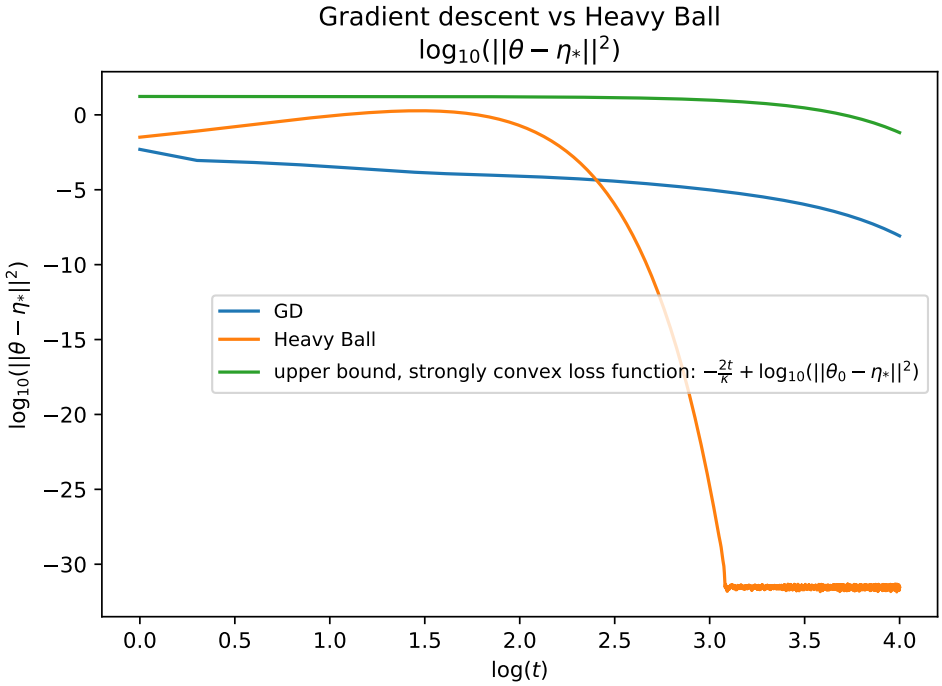
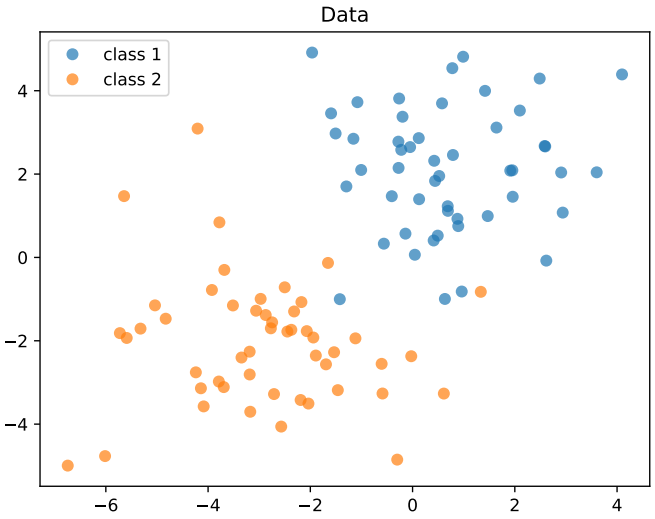


FIGURE 2 – Heavball vs GD, logarithmic scale

3 GD AND SGD FOR LOGISTIC REGRESSION

We will now perform GD, and Stochastic Gradient Descent (SGD) on Logistic regression. We will work on a binary classification problem, for which you can tweak the data using **generate _gaussian _data.py**.



3.1 Setup

We recall the setting of the problem.

— $\mathcal{X} = \mathbb{R}^2$

— $\mathcal{Y} = \{-1, 1\}$ (sometimes it is $\mathcal{Y} = \{0, 1\}$, in which case we change the loss)

— Logistic loss :

$$l(\hat{y}, y) = \log(1 + e^{-\hat{y}y}) \tag{11}$$

$$\begin{aligned} \frac{\partial l}{\partial \hat{y}}(\hat{y}, y) &= \frac{-ye^{-\hat{y}y}}{1 + e^{-\hat{y}y}} \\ &= \frac{-ye^{-\hat{y}y}e^{\hat{y}y}}{(1 + e^{-\hat{y}y})e^{\hat{y}y}} \\ &= \frac{-y}{1 + e^{\hat{y}y}} \\ &= -y\sigma(-\hat{y}y) \end{aligned} \tag{12}$$

The empirical risk writes :

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(x_i^T\theta, y_i) + \frac{\mu}{2}\|\theta\|^2 \tag{13}$$

We note $g_i(\theta) = l(x_i^T\theta, y_i)$.

$$\nabla_\theta g_i = -y_i\sigma(-x_i^T\theta y_i)x_i \tag{14}$$

Hence

$$\begin{aligned}\nabla_\theta R_n &= \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta g_i + \mu\theta \\ &= \frac{1}{n}\sum_{i=1}^{n} -y_i\sigma(-x_i^T\theta y_i)x_i + \mu\theta\end{aligned} \tag{15}$$

3.2 Files

— **generate _gaussian _data.py** (dimension 2) and **dgenerate _gaussian_data.py** (dimension d) : generate the data

— **TP4 _LR.py** : main file, launches the algorithms. You can change several parameters there, like the maximum number of iterations. The file uses the data generated by **generate _gaussian _data.py** and the other file.

— and **TP4 _algorithms.py** : the file containing the algorithms. You will have to edit it.

— **TP4 _utils.py** : other utilities.

3.3 Profiling

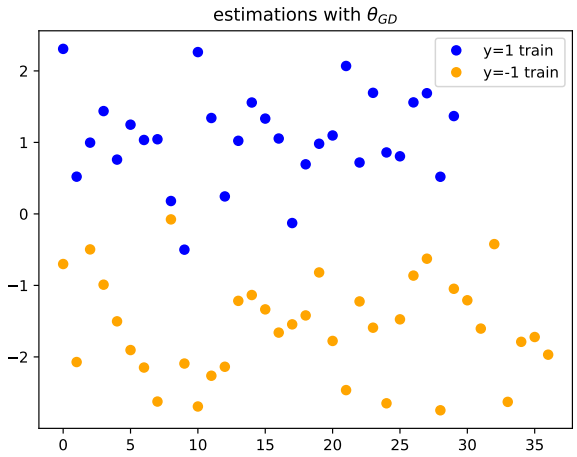
Note that some **profiling** is done by the algorithm file and is output to text files, for instance **profiling _gd.txt**. This can be useful to monitor the behavior of the algorithms.

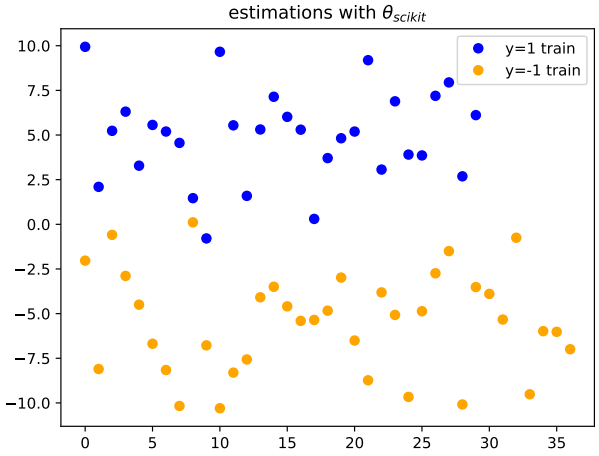
https://docs.python.org/3/library/profile.html

3.4 GD

Exercice 4 : Run GD (full gradient / batch gradient / deterministic gradient) on the LR problem, by running **TP4 _LR.py**. This should work without having to edit the files.

You can monitor the algorithm with the following images.





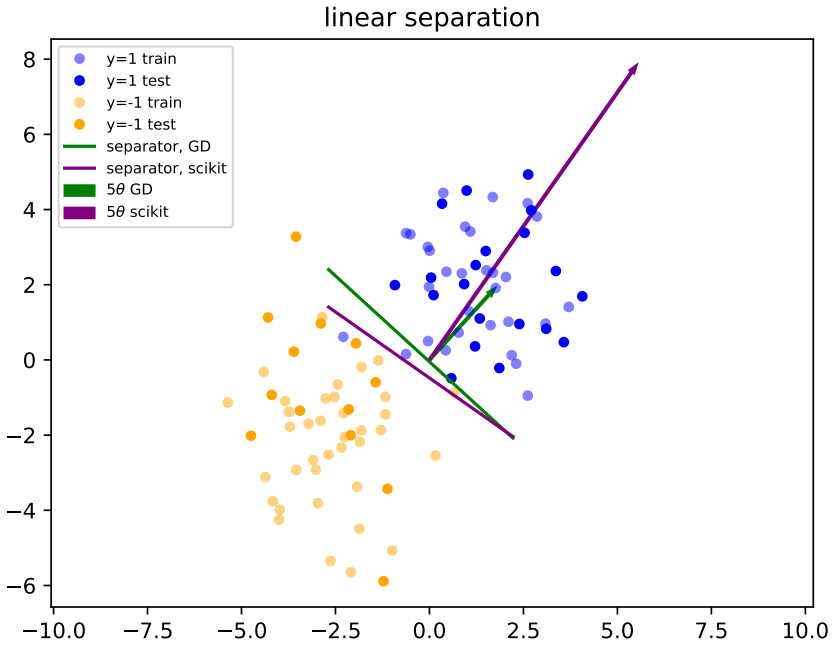


FIGURE 3 – Linear separators returned by the algorithms.

3.5 SGD

Exercice 5 : Implement SGD and monitor the convergence. You will need to uncomment the relevant lines in the main file and edit the file containing the algorithms.

3.5.1 *Averaging or last-iterate*

In the most general setting, upper bounds on the convergence speeds are obtained for an average of the parameter θ at the end of optimization, such as

$$\hat{\theta} = \frac{1}{t}\sum_{i=1}^{t}\theta_k \tag{16}$$

However, in some situations it is possible to have garantees **without** averaging, for example in the least-squares setup, in a statistical model called the noiseless model [Varre et al., 2021].

When no averaging is used, the method is called *last-iterate* SGD.

Exercice 6 : (Optional) You can experiment with averaging and monitor the impact on convergence speed.

3.5.2 *SAG*

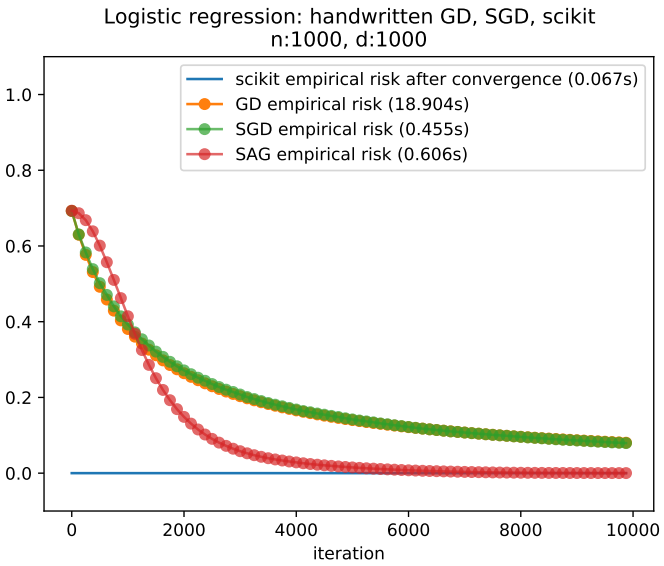
Stochastic Average Gradient (SAG) is a **variance reduction** method [Schmidt et al., 2013,]. The goal of these methods is to gather the advantages of GD (convergence speed in terms of the number of algorithm iterations) and SGD (low computational cost of one iteration, and more precisely a cost that does not depend on n). SAG keeps in memory and updates an estimation of the full gradient, built by averaging the previous estimates. The expected value of the estimate is still the same, but the variance is smaller than for SGD. See also : SAGA, an extension of SAG [Defazio et al., 2014].

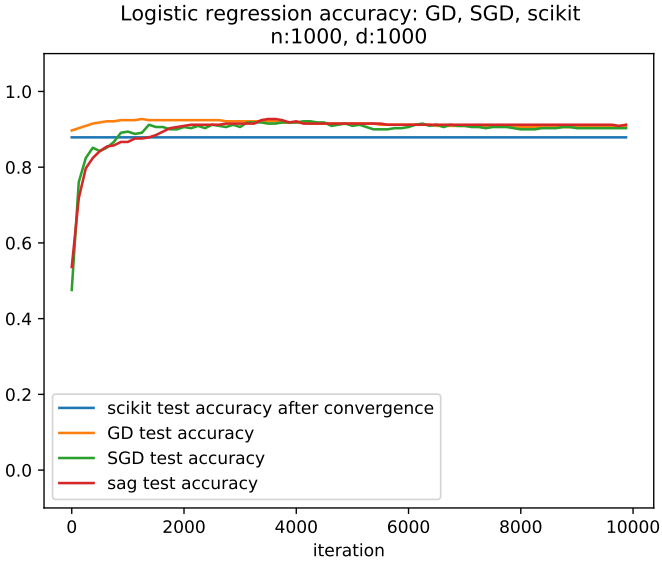
To be fully understood and optimized, these methods require some tuning and preparation, which will depend on the considered problem (do not hesitate to try to read the mentioned papers). However, we can implement the core SAG method with a constant step-size on our example (Logistic regression).

Exercice 7 : Implement SAG. You can find the algorithm in [Schmidt et al., 2013], page 10, the pdf is in the TP folder. You will need to uncomment the relevant lines in the main file and edit the file containing the algorithms.

3.6 Dimensions n and d

Experiment with n and d and with the data, in order to study their influence on the relative speed of the methods. You can delete the lines after the computation times printing.





RÉFÉRENCES

[Defazio et al., 2014] Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA : A fast incremental gradient method with support for non-strongly convex composite objectives. Advances in Neural Information Processing Systems, 2(January) :1646–1654.

[Schmidt et al., 2013] Schmidt, M., Le Roux, N., and Bach, F. (2013). Minimizing finite sums with the stochastic average gradient. Mathematical Programming, 162(1-2) :83–112.

[Varre et al., 2021] Varre, A., Pillaud-Vivien, L., and Flammarion, N. (2021). Last iterate convergence of SGD for Least-Squares in the Interpolation regime. CoRR, abs/2102.0.