Internship report Specialty:

Biostatistics

High dimensional optimization for penalized linear models with first order interactions using graphics card computational power

Tanguy Lefort

Supervised by:

M. Joseph Salmon	University of Montpellier
M. Benjamin Charlier	University of Montpellier

Presented Thursday, July 20th 2021 with jury:

M. Jean-Michel Marin	University of Montpellier
M. Paul Bastide	University of Montpellier
Mme. Elodie Brunel-Piccinini	University of Montpellier

Remerciements

Je tiens tout d'abord à remercier mes encadrants Joseph Salmon et Benjamin Charlier pour leur supervision tout au long de ce stage. Des parties plus théoriques à la programmation numérique, leurs conseils et explications auront été très utiles et le seront tout autant pour la suite.

Je remercie également Florent Bascou pour m'avoir introduit à son travail de thèse. Nos nombreuses discussions ont permis de comprendre rapidement certaines subtilités du sujet mais aussi de découvrir d'autres méthodes possibles.

Je souhaite aussi remercier Sophie Lèbre et Thomas Moreau. Sophie pour ses explications par rapport à la partie génomique de ce rapport. Cela a permis d'amener un apport pratique à ce travail. Thomas pour m'avoir guidé dans mes contributions à la librairie `BenchOpt` et la confiance accordée pour apporter des modifications au site de la librairie.

Enfin merci à Jean-Michel Marin pour le temps consacré à ce rapport.

Abstract

French: Le modèle linéaire est utilisé en statistiques pour sa simplicité, et l'interprétabilité des résultats obtenus. Sur des données génomiques, les dimensions très grandes imposent d'utiliser des méthodes robustes qui sélectionnent les variables actives pour avoir des résultats interprétables pour des biologistes. En plus de l'effet de nos variables on cherche aussi à capturer les effets des interactions, ce qui augmente encore la dimension du problème et les phénomènes de colinéarité. Pour pallier à cela, nous considérons l'Elastic-Net sur le problème augmenté. La descente par coordonées (Wu and Lange, 2008) est très utilisée pour résoudre ce type de problèmes, mais ce n'est pas l'unique possibilité. Nous utilisons la structure du problème avec interactions du premier ordre pour paralléliser des algorithmes de descente de gradient proximal. Ceux-ci sont connus pour être plus lents à converger du point de vue de la complexité, mais utiliser la parallélisation sur carte graphique permet dans certaines situations d'être plus rapide. Ce travail sur les méthodes d'optimisation s'inscrit dans le développement de la librairie `BenchOpt` permettant de comparer facilement différents algorithmes.

English: Linear models are used in statistics for their simplicity and the interpretability of the results. On genomics datasets, large dimensions need robust methods that induce sparsity to select interpretable active features for biologists. In addition to the main features, we also capture the effects of the interactions, which increase the dimension of the problem and the multicolinearity. To counteract these issues, we use the Elastic-Net on the augmented problem. Coordinate Descent (Wu and Lange, 2008) is mostly used nowadays for that, but there are other methods available. We exploit the structure of our problem with first order interactions to use parallelized proximal gradient descent algorithms. Those are known to be more computationally demanding in order of magnitude, but parallelizing on a graphics card let us be faster in some situations. This work is set in the development of the `BenchOpt` library. This library let us easily compare different optimization algorithms.

Github repository The code to generate the Figures and other results in this report is available at:

https://github.com/tanglef/interactionsmodel

Contents

1 Introduction

1.1 Building models, but at what cost?

Supervised learning uses a cost function that we aim to minimize. However, minimizing such functions with the data available can sometimes be costly both in term of time and memory. In a regression setting, the popular least-squares estimator benefits from great interpretability. For a response $y \in \mathbb{R}^n$ and some data $X \in \mathbb{R}^{n \times p}$ a least-squares estimator $\hat{\beta}^{ls}$ is any solution of the optimization problem:

$$\hat{\beta}^{ls} \in \underset{\beta \in \mathbb{R}^p}{\arg\min} \frac{1}{2n} \left\| y - X\beta \right\|_2^2 \ .$$

Some issues can arise like the curse of dimensionality. When p is larger than n, uniqueness of the solution is lost, leading to non interpretable coefficients. For this reason, penalties can be added to the cost function to recover targeted structures. We focus on the Elastic-Net (Zou and Hastie, 2005) estimator $\hat{\beta}^{en}$. It uses a trade-off between the ℓ_1 (LASSO (Tibshirani, 1996)) and ℓ_2 (Ridge (Tikhonov, 1943)) penalties on the coefficients to induce sparsity and regularize the problem. We remind that the ℓ_1 norm is the sum of the magnitude of the coefficients and the squared ℓ_2 norm is the sum of squares of the coefficients. The Elastic-Net is thus

$$\hat{\beta}^{en} \in \underset{\beta \in \mathbb{R}^p}{\arg\min} \frac{1}{2n} \left\| y - X\beta \right\|_2^2 + \lambda_{\ell_1} \left\| \beta \right\|_1 + \frac{\lambda_{\ell_2}}{2} \left\| \beta \right\|_2^2 \ . \tag{1.1}$$

In our model, we want to handle first order interactions. To do so, we have to consider a matrix Z whose columns are involving a function that creates new covariates from couples of original data. This function in all generality could be a max, min,.... We chose to focus on the element-wise product. This is not the most interpretable for biologists, but it creates a base that is easy to implement and easy to modify afterwards for new functions. More practically, it means that Z has n samples and d features with d the number of unique interactions. As for the estimation, in addition to the vector $\beta \in \mathbb{R}^p$, we need to estimate a vector $\Theta \in \mathbb{R}^d$. For the 1^{st} feature there are p other features to interact with, $(p-1)$ for the 2^{nd},..., two for the $(p-1)^{th}$ and one for the last. So in total there are $d = p(p+1)/2$ unique interactions to consider.

The memory footprint of Z is in practice quickly too large for the computer. From Figure 1.1, we see that the memory footprint of Z with 500 features and 20000 samples in X exceeds 10Gb. So there are difficulties not only for the statistics part, but also for the implementation of the solvers.

Research for fast algorithms to solve Elastic-Net or LASSO problems is very active. These problems search for the active variables *i.e.*, without interactions $\{\beta_i : \beta_i \neq 0,\ i = 1,\dots,p\}$. Some strategies like the working set are to build up the support by including features that are likely active (Johnson and Guestrin, 2015). Other methods (safely) discard features that are not relevant using the duality gap (Ndiaye et al., 2017). Solvers can also be accelerated. Inertial acceleration like Nesterov (Nesterov, 2012) lead to theoretically faster methods (but sometimes not in practice). Anderson extropolation (Bertrand and Massias, 2021) can also be used. It is not an inertial acceleration but considers the structure of the iterates to converge faster. Stochastic methods are also used to determine the direction to optimize (Chen, Li, and Lu, 2021).

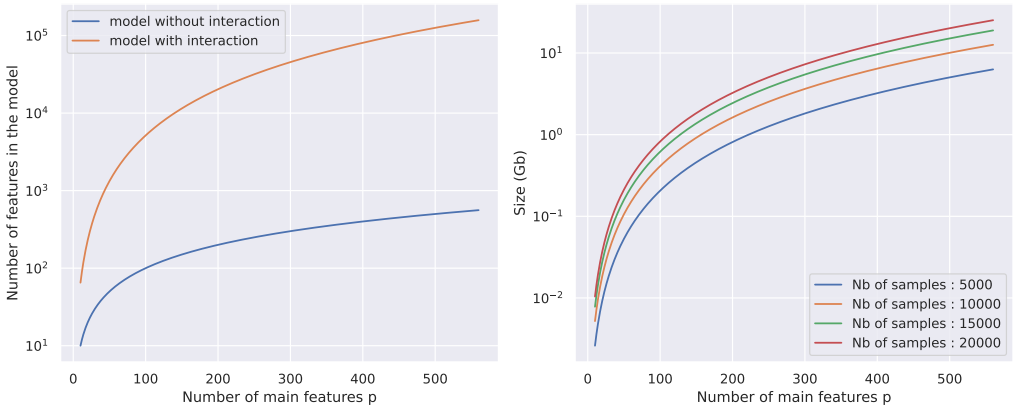


Figure 1.1: Number of features in the models with and without interactions (left) and size of the interaction matrix in Gigabytes for varying number of features in X (right).

On top of the theoretical accelerations, libraries like `Numba` (Lam, Pitrou, and Seibert, 2015) lead to some computational speed up. This is what is used in (Bascou, Lèbre, and Salmon, 2020) to solve the Elastic-Net with interactions. In our work, we propose to use the GPU with inertial acceleration.

1.2 Computation with a GPU

With the rise of new technologies for always better graphical rendering in video games, graphics cards have become a computational powerhouse. Generally, computations are made on CPUs (Central Processing Unit). Standard programs are designed for them, so there is no extra manipulation needed. This is different for GPUs (Graphic Processing Unit).

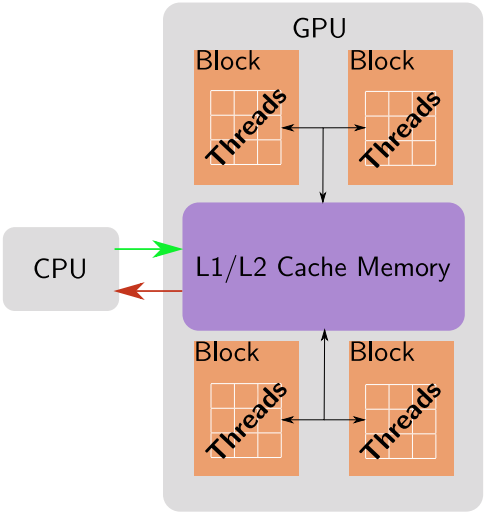


Figure 1.2: Simplified architecture of a GPU adapted from (Feydy et al., 2020, chapter 2). Typically, the data is first loaded onto the CPU. Then it is copied (sometimes in part) to the GPU device. This operation is very costly. There is a hierarchy of memory from here. Each GPU has an L2 cache which communicates with several independant L1 caches for each block. Threads operations are then executed simultaneously, organized in blocks. After all operations are done, the results are transferred back to the CPU.

A GPU has a very large amount of cores allowing heavy parallelizations, possibly asynchronously (see Figure 1.2). This means that operations are kept in a queue and computed when needed. In the

corporate world of GPUs, there is mostly only NVidia that is investing for faster devices for the AI research. They have developed the `CUDA` toolkit for developers to leverage GPUs performances. At first code in `C / C++`, libraries like `PyTorch`, `TensorFlow` in `Python` or `RCUDA` in `R` created APIs to use `CUDA` acceleration more easily.

There are some changes that need to be addressed with GPUs. Some of them are discussed in Chapter 3. But one (if not the main) to remember, is that transfers between CPUs and GPUs are usually creating the computation bottleneck. Transferring data from one device to another is very costly. So even if Z was storable on the CPU, making transfers to the GPU by parts would be very slow.

1.3 Leveraging GPUs computations for the optimization

State-of-the-art solvers consider coordinate descent type algorithms for sparse regression. The strength of the GPU comes from the possibility to process a lot of data simultaneously. These algorithms treat each feature separately, so very few data at a time. Our goal is thus to determine if some methods generally slow on CPUs could be competitive using `CUDA` acceleration. After presenting the (proximal) gradient descent methods we will explore some pros and cons of using GPUs in such settings. Then we apply our solvers to simulated datasets and a real genomics dataset to explore different behaviors. One of the difficulties encountered when dealing with such algorithms is the convergence. Indeed, we need a stopping criterion in order to decide when the convergence is reached. For that, we derive the KKT violation criterion for the Elastic-Net with interactions.

This work was made during an internship supervised by Joseph Salmon[1] and Benjamin Charlier[2] at the Institut Montpellierain Alexender Grothendieck. The base material is from the ongoing work of Florent Bascou[3]. The genomics dataset was provided by Sophie Lèbre[4].

[1] http://josephsalmon.eu/
[2] https://imag.umontpellier.fr/~charlier
[3] https://bascouflorent.github.io/
[4] https://www.univ-montp3.fr/miap/~lebre/

2 Elastic-Net estimator with interactions

In a linear model, when the number of features is too large, we need to perform some kind of feature selection to keep the benefits of the interpretability. To do so, it is now common to penalize the problem. Methods such as LASSO (Tibshirani, 1996) or Elastic-Net (Zou and Hastie, 2005) use penalties to counteract different issues. The ℓ_1 penalty is used to induce some sparsity in the obtained estimation and the ℓ_2 penalty to take into account the correlation amongst the features and regularize the data.

With a dataset $X \in \mathbb{R}^{n\times p}$, the interactions matrix Z has $\mathcal{O}(p^2)$ features, and thus quickly becomes very large. This leads to solvers very time consuming in practice. Since our problem has a specific architecture, we can use methods that exploit it: like block optimization solvers. We compare a Coordinate Descent with interactions (Bascou, Lèbre, and Salmon, 2020) with Proximal Gradient Descent algorithms (Beck, 2017). We exploit the structure by block of the interaction matrix (Section 2.4) to use the Cyclic Block Proximal Gradient method.

2.1 Notations and reminders

We denote $[k] = \{1,\dots,k\}$. For any matrix A, A_{ij} is its (i,j)-entry, a_j the j^{th} column. For a subset $\mathcal{J} \subset [k]$, $A_{\mathcal{J}}$ is the sub-matrix $(a_j)_{j\in\mathcal{J}}$ with $\#\mathcal{J}$ columns. We use the same notation on a vector to consider the components of those indexes. The 2-norm $\|A\|_2$ is the largest singular value of A in magnitude. For two vectors $u, v \in \mathbb{R}^k$, $(u \odot v)_i = u_i v_i$ is the element-wise product. The ℓ_1 norm is $\|u\|_1 = \sum_{j\in[k]} |u_i|$, and the squared ℓ_2 norm is $\|u\|_2^2 = \sum_{j\in[k]} u_j^2$.

In the followings, we will consider our dataset the matrix $X = [x_1|\dots|x_p] \in \mathbb{R}^{n\times p}$ the concatenation of the p features. Following the ideas from (Le Morvan and Vert, 2018), we use $\tau_K : [p]^2 \longrightarrow [K]$ a function to index our interactions. Meaning that in the interaction matrix Z, $Z_{\tau_K(i,j)} = x_i \odot x_j$. In our situation, we have $K = q = p(p+1)/2$. And finally, we define the index of the block generated by x_j as the branch $\mathcal{B}_K(j) := \{\tau_K(j,l),\ l \in [p]\}$. So for example

$$Z_{\mathcal{B}_q(2)} = [x_2 \odot x_2|\dots|x_2 \odot x_p] \ .$$

2.2 Presentation of the model

For a dataset $X \in \mathbb{R}^{n\times p}$, the Elastic-Net estimator with first order interactions Z reads:

$$\begin{aligned}\widehat{p} &\in \min_{\substack{\beta\in\mathbb{R}^p\\ \Theta\in\mathbb{R}^q}} \frac{1}{2n}\left\|y - X\beta - Z\Theta\right\|_2^2 + \lambda_{\beta,\ell_1}\left\|\beta\right\|_1 + \lambda_{\Theta,\ell_1}\|\Theta\|_1 + \frac{\lambda_{\beta,\ell_2}}{2}\left\|\beta\right\|_2^2 + \frac{\lambda_{\Theta,\ell_2}}{2}\|\Theta\|_2^2 \qquad (\mathcal{P})\\ &\in \min_{\substack{\beta\in\mathbb{R}^p\\ \Theta\in\mathbb{R}^q}} f(\beta,\Theta) + g^{(1)}_{\lambda_{\beta,\ell_1},\lambda_{\beta,\ell_2}}(\beta) + g^{(2)}_{\lambda_{\Theta,\ell_1},\lambda_{\Theta,\ell_2}}(\Theta) \ .\end{aligned}$$

Splitting the two penalty functions allow us to consider a gradient based descent algorithm for the minimization, alternating the optimization on β and Θ.

2.2.1 Optimization tools

Since our objective to minimize is a convex non-differentiable function, it is necessary for us to remind some mathematical tools. In the following, the functions considered are proper (extended real function

that is not identically equal to $+\infty$ and cannot take the value $-\infty$). They are also closed (their epigraph (see Definition 2.2.3) is a closed set) and convex.

Definition 2.2.1. *Let $f : \mathcal{X} \to \mathbb{R} \cup \{+\infty\}$. The proximal operator of f evaluated at $u \in \mathcal{X}$, denoted by $\text{prox}_{\mu f}(u)$, with $\mu > 0$ is defined as:*

$$\text{prox}_{\mu f}(u) = \underset{x \in \mathcal{X}}{\arg\min} \left\{ f(x) + \frac{1}{2\mu} \|x - u\|_2^2 \right\}, \quad . \tag{2.1}$$

Proximal operators are essential in optimization methods. It is indeed possible to show (Beck, 2017, p. 270) that using a proximal operator to find the next iterate in a problem to minimize formulated as a sum is a generalization of solving a constrained-set objective minimization with projected gradient method (Combettes and Pesquet, 2011). Closely related to the proximal operators is the infimal convolution:

Definition 2.2.2. *Let f and g two functions in $\mathbb{R} \cup \{+\infty\}$. Then the infimal convolution $f \square g$ is:*

$$f \square g(x) = \inf_{u \in \mathcal{X}} \{ f(u) + g(x - u) \} \quad . \tag{2.2}$$

It is easy to see how the infimal convolution $f \square g$ is a regularization of the function f in the convex case by looking at the epigraphs (Fajardo, Vicente-Pérez, and Rodríguez, 2012).

Definition 2.2.3. *For a real function f over $\mathcal{X}$, the epigraph $\text{epi}(f)$ is:*

$$\text{epi}(f) = \{(x, y) \in \mathcal{X} \times \mathbb{R} \,|\, f(x) \leqslant y\} \quad .$$

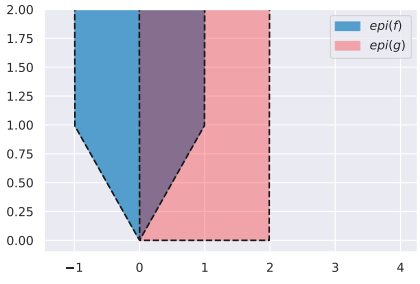
The link with the infimal convolution is given thanks to the Minkowski sum. As a reminder, the Minkowski sum of two sets of vectors A and B is: $A \oplus B = \{a + b, a \in A, b \in B\}$.

Proposition 1. *For two real functions f and g:*

$$\text{epi}(f \square g) = \text{epi}(f) \oplus \text{epi}(g) \quad . \tag{2.3}$$

Let us consider an example to visualize Equation (2.3). For $x \in \mathbb{R}$, we take $f(x) = |x|\,(1 + \iota(|x| \leqslant 1))$ and $g(x) = \iota(0 \leqslant x \leqslant 2)$, with ι the convex indicator function that equals 0 if the condition is met and $+\infty$ o.w. Then

$$f \square g(x) = \min_{u \in \mathbb{R}} \{ |u|\,(1 + \iota(-1 \leqslant u \leqslant 1)) + \iota(0 \leqslant x - u \leqslant 2) \} \quad .$$



(a) Epigraphs of $f(x) = |x|\,(1 + \iota(|x| \leqslant 1))$ and $g(x) = \iota(0 \leqslant x \leqslant 2)$.

(b) Epigraph of $f \square g(x)$.

Figure 2.1: Visualization that the epigraph of the infimal convolution is the Minkowski's sum of the epigraphs.

This implies that $f \square g(x)$ is not infinite for $u \in [-1, 1] \cap [x - 2, x]$ *i.e.*, for $x \in [-1, 3]$. We can then decompose the problem:

- if $x = -1$, then $u = -1$ and $f \square g(-1) = 1$ and if $x = 3$, then $u = 1$ and $f \square g(1) = 1$,
- if $x \in]0, 2]$, then $f \square g(x) = 0$,
- if $x \in]-1, 0]$ then $f \square g(x) = f(x)$,
- and finally if $x \in]2, 3[$, then $u \in]2 - x, 1[$ and $f \square g(x) = |2 - x|$.

Taking in particular g in Definition 2.2.2 as $g(u) = \frac{1}{2\mu}\|u\|_2^2$ we obtain the Moreau envelope.

Definition 2.2.4. *For a function $f : \mathcal{X} \to \mathbb{R} \cup \{+\infty\}$, the Moreau envelope is:*

$$\mathrm{M}_f^\mu(x) = \min_{u \in \mathcal{X}} \left\{ f(u) + \frac{1}{2\mu}\|x - u\|_2^2 \right\}, \quad \mu \geqslant 0 \ . \tag{2.4}$$

Note that the proximal operator is simply the solution to the Moreau envelope. And this envelope is simply a lower, regularized, version of the function f. Thus it can be rewritten:

$$\mathrm{M}_f^\mu(x) = \left\{ u \in \mathcal{X},\ f(u) + \frac{1}{2\mu}\|x - u\|_2^2 = f \square \frac{1}{2\mu}\|\cdot\|_2^2 (x) \right\}, \quad \mu \geqslant 0 \ . \tag{2.5}$$

Moreau's envelope is the key of our minimization algorithms. Indeed, it is equivalent to minimize f or M_f^μ. And it is possible to prove (Beck, 2017, chapter 6) that $\mathrm{prox}_{\mu f}(x) = x - \mu \nabla \mathrm{M}_f^\mu(x)$ which is simply a gradient step to minimize M_f^μ and thus f. Another key concept in optimization of non-differentiable functions (such as the ℓ_1 norm present in the Elastic-Net) is the subdifferential.

Definition 2.2.5. *Let $f : \mathcal{X} \to \mathbb{R} \cup \{+\infty\}$ be a real convex function. Then the subdifferential ∂f is:*

$$\forall x \in \mathrm{dom}(f) \quad \partial f(x) = \{ f^* \in \mathcal{X}^*,\ f(u) \geqslant f(x) + \langle f^*, u - x \rangle \ \forall u \in \mathcal{X} \} \ . \tag{2.6}$$

For example, it is well known that the absolute function is not differentiable at the origin. But as we can see in Figure 2.2,

$$\partial |x| = \begin{cases} -1, & \text{if } x < 0, \\ 1, & \text{if } x > 0, \\ [-1, 1], & \text{else .} \end{cases}$$

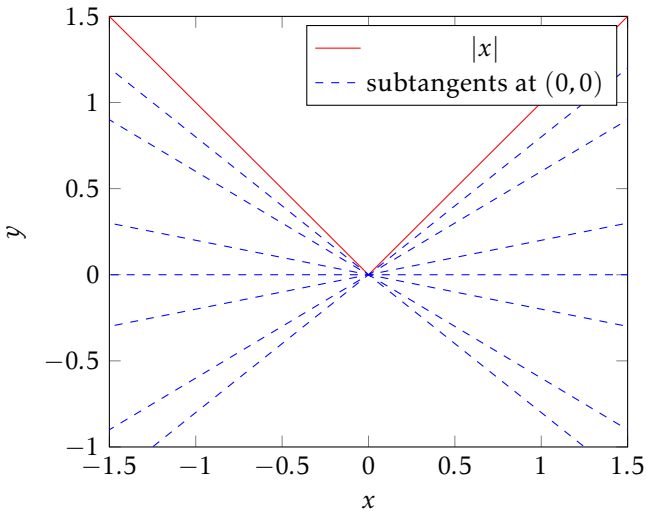


Figure 2.2: Visualization of the subtangents of the absolute value at $(0, 0)$. The slopes of the subtangents are in $\partial |x|_{|_{x=0}} = [-1, 1]$.

2.2.2 Coordinate descent algorithm for differentiable functions

If we consider the minimization of a differentiable function f, Coordinate Descent optimization is a possibility.

Algorithm 1: Coordinate Descent for a differentiable function F

Input : $\eta > 0$ step-size, $x = (x_i)_{i=1}^p$ initial vector, $\epsilon > 0$ the tolerance
while *stopping criterion* $> \epsilon$ **do**
  For $i = 1, \dots, p$ **do**
    $x_i \longleftarrow x_i - \eta \dfrac{\partial F}{\partial x_i}(x)$
Output : x_i

Algorithm 1 performs a gradient descent step on each coordinate. So instead of minimizing everything at once like we would with the gradient descent update:

$$\beta \longleftarrow \beta - \eta X^\top (X\beta - y) \ , \tag{2.7}$$

it reduced our problem of dimension p to p problems of dimension 1. This strategy for the steps direction is visible in Figure 2.3. In practice, the step size used is the squared ℓ_2 norm of each feature. In the case of the least squares $\left\|y - X\beta\right\|_2^2$, with $X \in \mathbb{R}^{n \times p}$, we update the i^{th} coordinate of β with step $\eta > 0$ using Equation (2.8):

$$\beta_i \longleftarrow \beta_i - \eta x_i^\top (X\beta - y) \ . \tag{2.8}$$

As we will see in Section 2.2.3, proximal operators allow us to minimize our objective even if it is defined by an ℓ_1 norm. Meaning that the principle of Coordinate descent, which is minimizing each coordinate, can be kept.

Note that in Algorithm 1 we chose to update the coordinates cyclically. Other strategies exist, such as a greedy implementation that will optimize the coordinate with the largest decrease. Some alternatives also use random choices for the updates (Nesterov, 2012). One important detail is about the residuals update. Equation (2.8) uses the residuals $X\beta - y$ to update the coordinate. Instead of recomputing the residuals from scratch at each step, it is more efficient to see that between two iterates β^{new} and β^{old} for which j_0 is updated:

$$X\beta^{new} - y = X\beta^{old} - y - X_{j_0}\beta_{j_0}^{old} + X_{j_0}\beta_{j_0}^{new} \ .$$

So denoting r^{old} and r^{new} the residuals associated to β^{old} and β^{new}, it follows that

$$r^{new} = r^{old} - X_{j_0}(\beta_{j_0}^{old} - \beta_{j_0}^{new}) \ . \tag{2.9}$$

By doing so, an update with Coordinate Descent is made of two operations of complexity $\mathcal{O}(n)$.

2.2.3 Proximal Gradient Descent algorithms

Where Coordinate Descent minimizes each coordinate at a time, the idea behind Proximal Gradient Descent is to minimize them all at once. In a Least-Square problem with coefficients β with a convex penalty function g, the Proximal Gradient Descent algorithm is:

Algorithm 2: Proximal Gradient Descent for $X \in \mathbb{R}^{n \times p}$ (regularized OLS)

Input : $\beta = 0_p$ initial vector, $\epsilon > 0$ the tolerance
$L \longleftarrow \dfrac{\|X^\top X\|_2}{n}$ // step size (see Section 3.2)
while *stopping criterion* $> \epsilon$ **do**
  For $i = 1, \dots,$ **do**
    $\beta \longleftarrow \text{prox}_{\frac{1}{L}g}\left(\beta - \frac{1}{Ln}X^\top(X\beta - y)\right)$
Output : β

The convergence rate is of course slower. However exploiting the architecture of the problem and using the GPU can make Proximal Gradient Descent based methods competitive. One important difference in the computation is for the residuals. Since we update all coordinates at once, Equation (2.9) will result in:

$$r^{new} = r^{old} - X(\beta^{old} - \beta^{new}) \ , \tag{2.10}$$

resulting in one $\mathcal{O}(np)$ operation. Figure 2.3 shows that instead of having steps performed parallelly with the axes, we can advance in diagonals (optimize multiple coordinates at once).

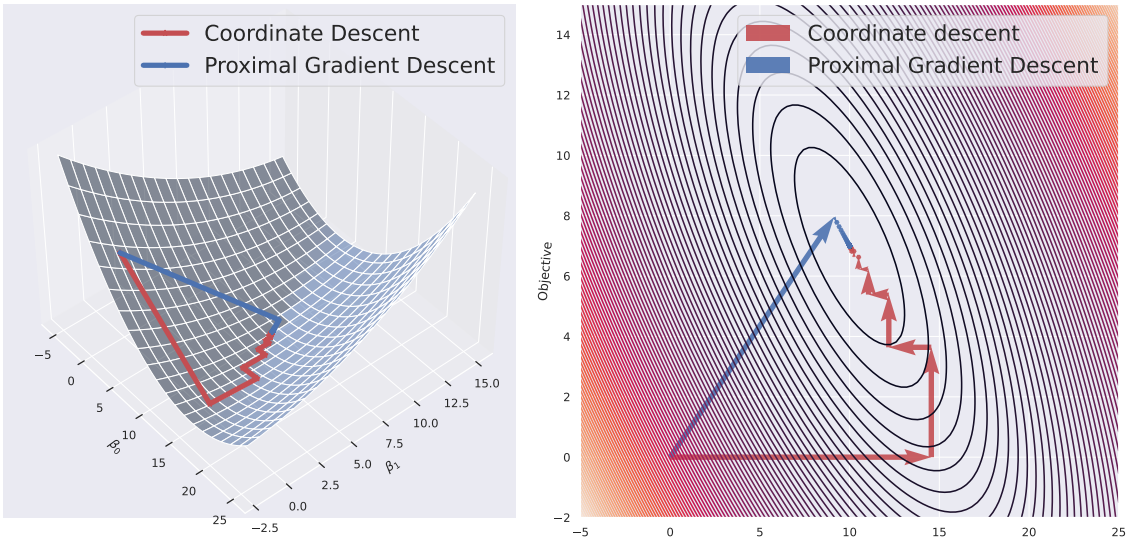


Figure 2.3: Path of the iterates of PGD and CD to minimize the cost function in a LASSO problem with a random matrix $n = 200$ and $p = 2$.

In our case, Problem ($\mathcal{P}$) is not to minimize a single function but a sum of convex functions with two variables β and Θ. Using Proximal Gradient Descent (Beck, 2017) in alternating setting, we obtain the following iterates for the k^{th} step

$$\beta^{k+1} = \text{prox}_{\frac{1}{L_X} g^{(1)}} \left(\beta^k - \frac{1}{L_X} \frac{\partial}{\partial \beta} f(\beta^k, \Theta^k) \right) \ , \tag{2.11}$$

$$\Theta^{k+1} = \text{prox}_{\frac{1}{L_Z} g^{(2)}} \left(\Theta^k - \frac{1}{L_Z} \frac{\partial}{\partial \Theta} f(\beta^{k+1}, \Theta^k) \right) \ , \tag{2.12}$$

where L_X and L_Z are Lipschitz constants for the partial derivatives to be determined. We still need to calculate the derivatives of f w.r.t. each component. Direct computation shows that:

$$\frac{\partial f}{\partial \beta}(\beta, \Theta) = \frac{1}{n} X^\top (X\beta + Z\Theta - y) \quad \text{and} \quad \frac{\partial f}{\partial \Theta}(\beta, \Theta) = \frac{1}{n} Z^\top (X\beta + Z\Theta - y) \ . \tag{2.13}$$

Now we can compute the Lipschitz constants. For L_X, let $\beta_1, \beta_2 \in \mathbb{R}^p$, we have

$$\begin{aligned} \left\| \frac{\partial}{\partial \beta} f(\beta_1, \Theta) - \frac{\partial}{\partial \beta} f(\beta_2, \Theta) \right\|_2 &= \frac{1}{n} \left\| X^\top (X\beta_1 + Z\Theta - y) - X^\top (X\beta_2 + Z\Theta - y) \right\|_2 \\ &= \frac{1}{n} \left\| X^\top X\beta_1 - X^\top X\beta_2 \right\|_2 \\ &\leqslant \frac{1}{n} \left\| X^\top X \right\|_2 \left\| \beta_1 - \beta_2 \right\|_2 \ . \end{aligned}$$

This bound is the smallest possible. Let $L' > 0$ be another Lipschitz constant. Indeed, if we take β_1 the unit eigenvector associated to the largest eigenvalue of $X^\top X$ and $\beta_2 = 0_p$ then:

$$\begin{aligned}\frac{\left\|X^\top X\right\|_2}{n} &= \frac{\left\|X^\top X\beta_1\right\|_2}{n} \\ &= \frac{1}{n}\left\|X^\top X\beta_1 - X^\top X 0_p\right\|_2 \\ &\leqslant L'\left\|\beta_1 - 0\right\|_2 = L' \ .\end{aligned}$$

We thus obtain that $L_X = \frac{\|X^\top X\|_2}{n}$ and with the same method $L_Z = \frac{\|Z^\top Z\|_2}{n}$.

2.3 Calculating the proximal operator

With the Elastic-Net, the proximal operators allow us to minimize the objective even though the penalty is not differentiable. We now need to compute this operator. For a function $h(x) = \|x\|_1 + \frac{\gamma}{2}\|x\|_2^2$, we know (Parikh and Boyd, 2014, p. 189) that for $\mu > 0$:

$$\text{prox}_{\mu h}(x) = \frac{1}{1+\mu\gamma}\text{prox}_{\mu\|\cdot\|_1}(x) = \frac{\text{sign}(x)}{1+\mu\gamma}(|x| - \mu)_+ \ ,$$

where $(\cdot)_+ = \max(\cdot, 0)$. With the penalty on β: $g^{(1)}$ (the second function $g^{(2)}$ is identical) we get

$$\text{prox}_{\mu g^{(1)}}(\beta) = \text{prox}_{\mu\lambda_{\beta,\ell_1} h^{(1)}}(\beta) = \frac{\text{sign}(\beta)}{1+\mu\lambda_{\beta,\ell_2}}(|\beta| - \mu\lambda_{\beta,\ell_1})_+ \ , \tag{2.14}$$

with $h^{(1)}(\beta) = \|\beta\|_1 + \frac{\lambda_{\beta,\ell_2}/\lambda_{\beta,\ell_1}}{2}\|\beta\|_2^2$ such that $\lambda_{\beta,\ell_1} h^{(1)}(\beta) = g^{(1)}(\beta)$.

Definition 2.3.1. *We denote* ST *the soft-tresholding operator such that:*

$$\text{ST}(x,\lambda) = \text{sign}(x)(|x| - \lambda)_+ = \begin{cases} x - \lambda & \text{if } x > \lambda, \\ x + \lambda & \text{if } x < -\lambda, \\ 0 & \text{else .} \end{cases}$$

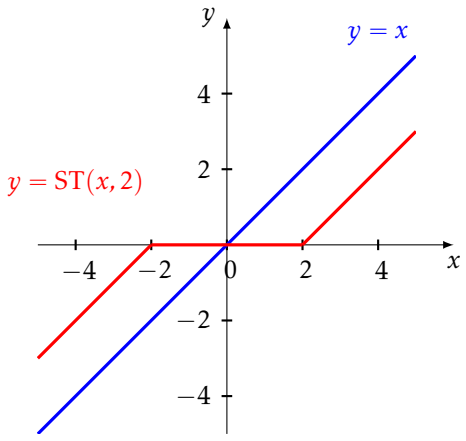


Figure 2.4: Soft-thresholding applied with $\lambda = 2$.

Updating β thus leads to:

$$\beta^{k+1} = \frac{1}{1 + \frac{n}{\left\|X^\top X\right\|_2}\lambda_{\beta,\ell_2}}\text{ST}\left(\beta^k - \frac{n}{\left\|X^\top X\right\|}\frac{1}{n}X^\top(X\beta^k + Z\Theta^k - y), \frac{n}{\left\|X^\top X\right\|_2}\lambda_{\beta,\ell_1}\right) \ . \tag{2.15}$$

For the update on Θ, considering

$$u = \Theta^k - \frac{n}{\left\|Z^\top Z\right\|}\frac{1}{n}Z^\top(X\beta^{k+1} + Z\Theta^k - y) \ ,$$

the gradient step associated, is then:

$$\Theta^{k+1} = \frac{1}{1 + \frac{n}{\left\|Z^\top Z\right\|_2}\lambda_{\Theta,\ell_2}}\mathrm{ST}\left(u, \frac{n}{\left\|Z^\top Z\right\|_2}\lambda_{\Theta,\ell_1}\right) \ . \tag{2.16}$$

2.4 Cyclic Block Proximal Gradient (CBPG)

With the Proximal Gradient Descent, we update all of the coordinates of β and Θ at once. With Coordinate Descent with interactions (Bascou, Lèbre, and Salmon, 2020), in each epoch we update each coordinate separately. Some kind of compromise between the two algorithms is possible using the block structure of the interaction matrix. Indeed, we can do block updates of Θ separately using the CBPG method (Massias, 2019; Beck, 2017). Doing so, we need the expression of the derivative involved for each block and the associated Lipschitz constants. Both can be retrieved using the same method as before, so we get:

$$\frac{\partial}{\partial\Theta_{\mathcal{B}_q(i)}}f(\beta,\Theta) = \frac{1}{n}Z^\top_{\mathcal{B}_q(i)}(X\beta + Z\Theta - y) \ , \tag{2.17}$$

$$\left\|\frac{\partial}{\partial\Theta_{\mathcal{B}_q(i)}}f(\beta,\Theta_1) - \frac{\partial}{\partial\Theta_{\mathcal{B}_q(i)}}f(\beta,\Theta_2)\right\|_2 \leqslant \frac{1}{n}\left\|Z^\top_{\mathcal{B}_q(i)}Z\right\|_2\left\|\Theta_1 - \Theta_2\right\|_2 \ . \tag{2.18}$$

We denote $L_i = \frac{1}{n}\left\|Z^\top_{\mathcal{B}_q(i)}Z\right\|_2$ the Lipschitz constant associated to the update of the i^{th} block. Notice that the residuals needed in the gradient can be updated by block as in Equation (2.9) to reduce the complexity.

Algorithm 3: Cyclic block proximal gradient for one epoch at iteration $k \geqslant 1$

Input : $X \in \mathbb{R}^{n\times p}$, $y \in \mathbb{R}^n$

$\beta^{k+1} = \mathrm{prox}_{\frac{1}{L_X}g^{(1)}}\left(\beta^k - \frac{1}{L_X}\frac{\partial}{\partial\beta}f(\beta^k,\Theta^k)\right)$,

For $i = 1,\dots,p$ **do**

$\quad \Theta^{k+1}_{\mathcal{B}_q(i)} = \mathrm{prox}_{\frac{1}{L_i}g^{(2)}}\left(\Theta^k_{\mathcal{B}_q(i)} - \frac{1}{L_i}\frac{\partial}{\partial\Theta_{\mathcal{B}_q(i)}}f(\beta^{k+1},\Theta^k)\right)$

Output : β,Θ

2.5 Stopping criterion

Finding a stopping criterion for iterative solvers is not a trivial task. Necessary conditions for a point to be optimal exist such as the Karush-Kuhn-Tucker (KKT) conditions. For an objective f penalized with $m > 0$ inequality constraints h_i, $i = 1,\dots,m$ the minimization problem writes:

$$\min_x f(x) \ s.t. \ h_i(x) \leqslant 0, \ i = 1,\dots,m \ .$$

This can be rewritten as minimizing the following Lagrangian function with $\mu_1,\dots,\mu_m \in \mathbb{R}$:

$$\mathcal{L}(x,\mu_1,\dots,\mu_m) = f(x) + \sum_{i=1}^m \mu_i h_i(x) \ .$$

Proposition 2.5.1. *The KKT conditions state that at the optimum x^**

1. $0 \in \partial f(x^*) + \sum_{i=1}^m \mu_i \partial h_i(x^*)$ *(stationarity)*,

2. $\forall i \in [m],\ \mu_i h_i(x^*) = 0$ *(complementary slackness),*

3. $\forall i \in [m],\ \mu_i > 0$ *and* $h_i(x^*) \leqslant 0$ *(feasibility).*

To stop our solvers, we can look at how much we violate these conditions. Simplifying the lines hereafter, let us consider an Elastic-Net problem with $W \in \mathbb{R}^{n \times p}$:

$$\underset{w \in \mathbb{R}^p}{\arg\min} \frac{1}{2n}\|y - Ww\|_2^2 + \lambda_1 \|w\|_1 + \frac{\lambda_2}{2}\|w\|_2^2 = \arg\min F_{enet}(w) \ . \tag{$\mathcal{P}_{enet}$}$$

KKT conditions dictate that at the optimum w^*:

$$0 \in W^\top (Ww^* - y) + \lambda_2 w^* + \lambda_1 \partial \left\|w^*\right\|_1 \ .$$

For the primal ($\mathcal{P}_{enet}$), we stop when current iterate w is such that $d_{\|\cdot\|}(0, \partial F_{enet}(w)) \leqslant \epsilon$ for $\epsilon > 0$. This means that we are at most violating the KKT conditions by ϵ for the considered norm. Since the subdifferential is a set, the distance considered is the distance of a vector to a set. For a vector $u \in \mathbb{R}^n$ and a set $E \subset \mathbb{R}^n$ it means:

$$d_{\|\cdot\|}(u, E) = \inf_{v \in E} \|u - v\| \ .$$

We will consider the ∞-norm to have all coordinates below ϵ in absolute value:

$$\inf_{h \in \partial F_{enet}(w)} \|h\|_\infty \leqslant \epsilon \ . \tag{2.19}$$

We can consider the problem coordinate-wise as we will take the maximum over the coordinates. If $h \in \partial F_{enet}(w)$ then for $j \in [p]$,

$$h_j \in \frac{1}{n} W_j^\top (Ww - y) + \lambda_1 \partial_{|\cdot|}(w_j) + \lambda_2 w_j \ .$$

The distance for each coordinate we want to compute is then:

$$\begin{aligned} d(0, \partial F_{enet}(w)_j) &= d\left(0, \frac{1}{n} W_j^\top (Ww - y) + \lambda_1 \partial_{|\cdot|}(w_j) + \lambda_2 w_j\right) \\ &= d\left(-\frac{1}{n} W_j^\top (Ww - y) - \lambda_2 w_j, \lambda_1 \partial_{|\cdot|}(w_j)\right) \ . \end{aligned}$$

Denoting $r = y - Ww$ the residuals, we get:

$$\begin{aligned} d\left(\frac{1}{n} W_j^\top r - \lambda_2 w_j, \lambda_1 \partial_{|\cdot|}(h_j)\right) &= \inf_{\eta_j \in \partial_{|\cdot|}(h_j)} \frac{1}{n} \left| W_j^\top r - n\lambda_2 w_j - n\lambda_1 \eta_j \right| \\ &= \frac{1}{n} \left| \mathrm{ST}\left(W_j^\top r - n\lambda_2 w_j, n\lambda_1\right) \right| \ . \end{aligned} \tag{2.20}$$

We can stop the solver when the maximum value for $j \in [p]$ of Equation (2.20) is below ϵ[1]. This strategy can be applied to our interaction problem considering that $W = [X\,|\,Z]$ and $w = [\beta\,|\,\Theta]$. Equation (2.20) then writes as:

$$d(0, \partial F_{enet}(w)_j) = \frac{1}{n} \left| \mathrm{ST}\left(W_j^\top r - n[\lambda_{\beta,\ell_2} \mathbb{1}(j \leqslant p) + \lambda_{\Theta,\ell_2} \mathbb{1}(j > p)] w_j, n[\lambda_{\beta,\ell_1} \mathbb{1}(j \leqslant p) + \lambda_{\Theta,\ell_1} \mathbb{1}(j > p)]\right) \right| \ . \tag{2.21}$$

[1] see Appendix A for convergence rates results.

2.5.1 Reformulating the criterion from a LASSO perspective

Since the Elastic-Net can be rewritten as a LASSO problem, Equation (2.20) can also be derived from the KKT violation. Indeed, we have:

$$\begin{aligned}\frac{1}{2n}\left\|y - Ww\right\|_2^2 + \lambda_1\|w\|_1 + \frac{\lambda_2}{2}\|w\|_2^2 &= \frac{1}{2n}\left\|y - Ww\right\|_2^2 + \lambda_1\|w\|_1 + \sum_{j=1}^{p}\left(\frac{\sqrt{\lambda_2}}{2}w_j\right)^2 \\ &= \frac{1}{2n}\left[\left\|y - Ww\right\|_2^2 + \sum_{j=1}^{p}\left(\sqrt{\lambda_2 n}w_j\right)^2\right] + \lambda_1\|w\|_1 \\ &= \frac{1}{2n}\left\|\widetilde{y} - \widetilde{W}w\right\|_2^2 + \lambda_1\|w\|_1 \ ,\end{aligned}$$

with $\widetilde{W} = \begin{bmatrix} W \\ \sqrt{\lambda_2 n}\mathrm{Id}_{p\times p}\end{bmatrix} \in \mathbb{R}^{(n+p)\times p}$ and $\widetilde{y} = [y\,|\,0_p]^\top \in \mathbb{R}^{(n+p)\times p}$. Equation (2.19) renders as:

$$\max_{j\in[p]}\frac{1}{n}\left|\mathrm{ST}\left(\widetilde{W}_j^\top\widetilde{r}, n\lambda_1\right)\right| \leqslant \epsilon \ . \tag{2.22}$$

Working with interactions, Problem ($\mathcal{P}$) can be rewritten as a weighted LASSO objective:

$$\min_{\beta,\Theta}\frac{1}{2n}\|\widetilde{y} - \widetilde{X}\beta - \widetilde{Z}\Theta\|_2^2 + \lambda_{\beta,\ell_1}\left\|\beta\right\|_1 + \lambda_{\Theta,\ell_1}\|\Theta\|_1 \ ,$$

with

$$\widetilde{y} = [y\,|\,0_p\,|\,0_q]^\top \in \mathbb{R}^{n+p+q}, \quad \widetilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda_{\beta,\ell_2}n}\mathrm{Id}_{p\times p} \\ 0\end{bmatrix} \in \mathbb{R}^{(n+p+q)\times p} \text{ and } \widetilde{Z} = \begin{bmatrix} Z \\ 0 \\ \sqrt{\lambda_{\Theta,\ell_2}n}\mathrm{Id}_{q\times q}\end{bmatrix} \in \mathbb{R}^{(n+p+q)\times q} \ .$$

To recover a weighted-LASSO, we concatenate both matrices. We denote $\widetilde{W} = [\widetilde{X}\,|\,\widetilde{Z}]$, $\widetilde{w} = [\beta\,|\,\Theta]$, and $\lambda = [\lambda_{\beta,\ell_1}\mathbf{1}_p|\lambda_{\Theta,\ell_1}\mathbf{1}_q]$, with $\mathbf{1}_p$ the vector of ones of dimension p. Problem Equation ($\mathcal{P}$) renders as:

$$\underset{\widetilde{w}\in\mathbb{R}^{p+q}}{\arg\min}\frac{1}{2n}\|y - \widetilde{W}\widetilde{w}\|_2^2 + \sum_{j=1}^{p+q}\lambda_j\left|\widetilde{w}_j\right| \ .$$

This leads to Equation (2.21) with a LASSO problem. We can stop the solver when:

$$\max_{j\in[p+q]}\frac{1}{n}\left|\mathrm{ST}\left(\widetilde{W}_j^\top\widetilde{r}, n\lambda_j\right)\right| \leqslant \epsilon \ .$$

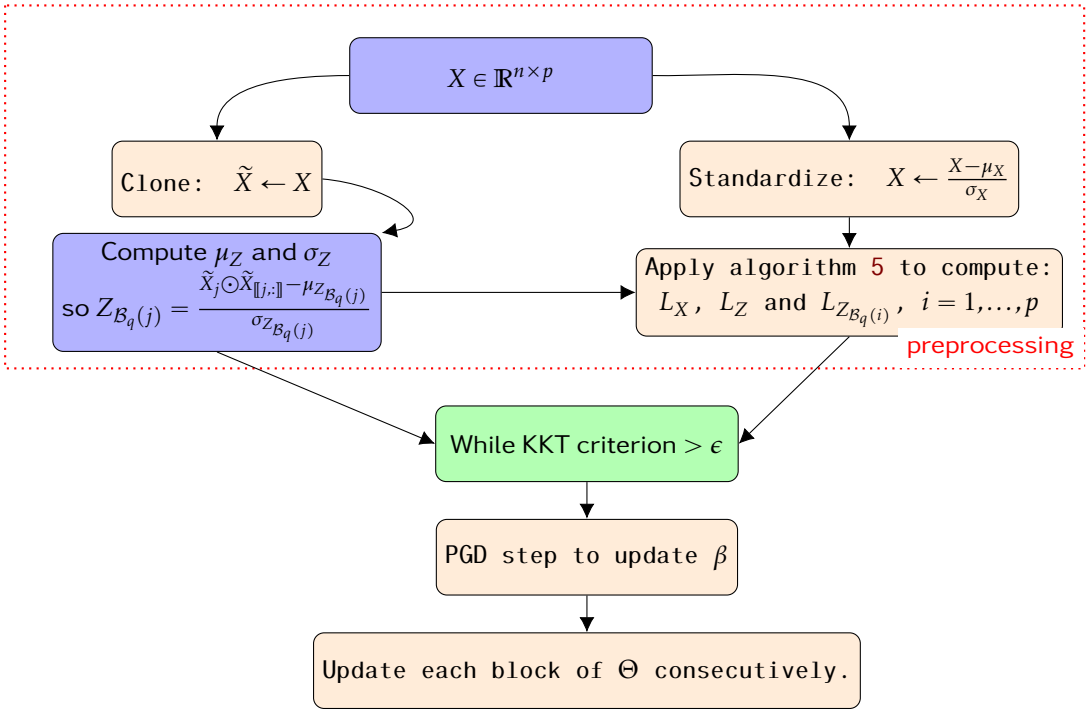
2.6 Pipeline and discussion about the algorithm used

2.6.1 Main steps for the Cyclic Block pipeline

Here, we synthetize the pipeline for the Cyclic Block Proximal Gradient algorithm. In Chapter 3, we will see how in practice we can use this algorithm, especially the computation of the steps and how it performs on datasets.

The values in μ_X, σ_X, μ_Z and σ_Z can be stored once and reused later by the solver. So there we can also not clone the data at the beginning. This implies keeping the non-standardized data during the run and standardize on-the-fly when a matrix-vector product is computed. The same applies to the Lipschitz constants that can be reused if different regularizations are tested (like in a grid search).

The cost of the KKT criterion is also very low because in Equation (2.21), we only need the residuals. Those are already computed so we do not need to recompute them. So the most costly operations are the products with $X^\top$ and $Z^\top$ respectively in $\mathcal{O}(np)$ and $\mathcal{O}(nq)$.



2.6.2 Discussion of using PGD/CBPG instead of CD

As an example, we can consider the usual LASSO problem (as the Elastic-Net is just an extension of the LASSO as we saw in Section 2.5.1). We benchmark the convergence of the objective function for CD and PGD methods, with and without acceleration in Figure 2.5.

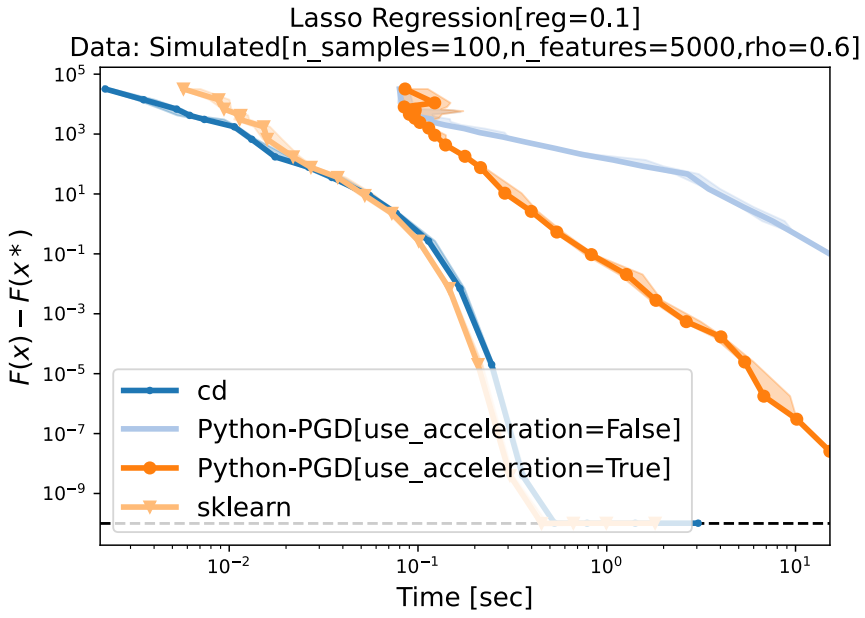


Figure 2.5: Suboptimality curves of CD and PGD to solve a simulated LASSO problem of 100 samples and 5000 features. We also compare with the solver from `Scikit-learn` (which is CD-based). The ℓ_1 penalty is set to 0.1. The parameter ρ controls cross-correlations: for features i and j, the cross-correlation is $C_{i,j} = \rho^{|i-j|}$. Note that no GPU was used.

Figure 2.5 was made with the BenchOpt library[2]. As we can clearly see, PGD methods converge slowly, and even slower without inertial acceleration. Hereafter, the acceleration considered is Nesterov inertial acceleration (Nesterov, 1983). This takes into account with a specific weight the previous iterate value.

So trying to compete with CD is neither easy nor manageable with CPUs. And if GPUs can, not necessarily beat CD, but only become a possible contestant, then maybe other good statistical algorithms judged too time consuming could be used again. Not only old methods, new methods may be even faster using GPUs.

[2] see https://benchopt.github.io/ for the main page, https://benchopt.github.io/results for the results visualization (interactive with Plotly!) and Chapter 5 for more.

3 Parallel numerical scheme for linear models with interactions

3.1 Handle memory issues with the interaction matrix

In the linear model with interactions, we need to build aforementioned interactions. We store them in the matrix $Z \in \mathbb{R}^{n \times q}$, with $q = \mathcal{O}(p^2)$. For large numbers of features p, it quickly becomes problematic to store this data. We thus consider a block approach that will only need to store a $n \times p$ matrix upmost at a time. Indeed, we can decompose Z into element-wise products with slices of X: $Z = \left[X \odot x_1 | X_{[\![2,p]\!]} \odot x_2 | \cdots | X_{[\![p,p]\!]} \odot x_p\right] = \left[Z_{\mathcal{B}_q(1)} | \cdots | Z_{\mathcal{B}_q(p)}\right]$. So the product of Z by a vector $\Theta \in \mathbb{R}^{p(p+1)/2}$ is:

$$Z\Theta = \left[Z_{\mathcal{B}_q(1)}\Theta_{\mathcal{B}_q(1)} + \cdots + Z_{\mathcal{B}_q(p)}\Theta_{\mathcal{B}_q(p)}\right] .$$

The first sum is a product with an $n \times p$ matrix, the second sum is with a $n \times (p-1)$ matrix, and so on until the last which is with a $n \times 1$ matrix. Each block is computed efficiently using `CUDA` acceleration.

For the product $Z^\top \xi$, $\xi \in \mathbb{R}^n$ (used in the gradient step), we can also divide the blocks along the axis of size q. Here, we sum over the axis of size n which is quite large. We can divide it to handle smaller reductions (see Figure 3.1).

$$Z^\top \xi = \begin{bmatrix} \blacksquare \\ \blacksquare \\ \vdots \\ \blacksquare \end{bmatrix} = \begin{bmatrix} 1 + 2 + \cdots + K \\ 1 + 2 + \cdots + K \\ \vdots \\ 1 + 2 + \cdots + K \end{bmatrix}$$

Figure 3.1: Matrix-vector product for $Z^\top$ where K should be big enough to avoid any error but also small enough to keep the power of a matrix product.

And for each row, the k^{th} sub-block, $1 \leqslant k \leqslant K$ is computed as follows (here for the first main row):

$$\blacksquare = \left[(Z_{\mathcal{B}_q(1)})^\top_{[\![n_k, n_{k+1}]\!]} \xi_{[\![n_k, n_{k+1}]\!]}\right] .$$

For the j^{th} main row, the first line uses the coefficients from $x_j \odot x_j$ but the last line always uses those from $x_j \odot x_p$. Each sub-block k is of size n_k (for example the quotient of n by k, ± 1 to reach n at the end).

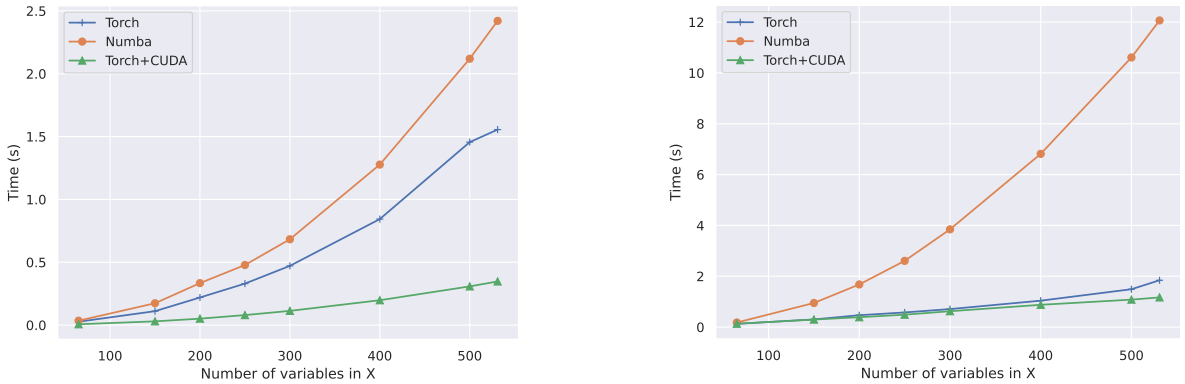
Remark Having an efficient memory management is very important when working with GPUs. There is very fewer available storage on a GPU than on a CPU. Let us take as reference the NVIDIA GeForce RTX2080 (a standard graphical card that was used for the followings experiments). The data can be stored in arrays (or tensor) of different types. If stored with float-32, each element has a memory

footprint of 4 bytes. In float-64, it is 8 bytes. The buffer memory of the GPU has $8GB$ available. So for a float-32 tensor of size $n \times p$, we need

$$4np \leqslant 8 \times 10^9 \Longrightarrow \sqrt{np} \leqslant 4.5 \times 10^4 \ .$$

And more generally considering the variance accross GPUs (especially their date of release), we should have $\sqrt{np} \in [10000, 50000]$. So the genomics data on which we will apply our methods (see Chapter 4) generates a matrix Z of size 20000×141246 that does not fit.

Recall that the algorithm we are trying to compete with (Bascou, Lèbre, and Salmon, 2020) uses the `Numba` library to compile and accelerate the code. Now that we know how to make a matrix-vector product with the matrix Z, is there a chance that, using GPU acceleration, we can perform operations faster than `Numba`? To see the time consumption of products of Z with an array for different sizes, we used the data from Chapter 4 with more than 19900 samples and a large number of features available. We vary the number of features from X and compute a product with the matrix Z created on-the-fly. We test it for `Numba` with a double loop (the way it is implemented in the CD solver to update each coordinate). We compare it with `PyTorch` using blocks (implemented for PGD/CBPG) with and without `CUDA` to check that there is indeed an interest to perform these operations with a GPU. Also, because in the algorithms we use both products with Z and $Z^\top$ and the dimensions can be very different, we propose to monitor both. Each operation has been repeated 20 times.



(a) Product with the matrix Z.

(b) Product with the matrix $Z^\top$.

Figure 3.2: Comparison on a genomics dataset of the median time to perform matrix vector products with the interaction matrix over 20 repetitions.

Figure 3.2 shows us that indeed the time performance of matrix vector products with `CUDA` as backend are performed way faster as the number of features considered increases. So can we actually increase the performance of Proximal Gradient type descent algorithms this way? To answer this question, we still need some important bricks to our foundations. One of them being: at each step we want to move towards the solution: but how do we choose the step size?

3.2 Step-size for the interaction matrix

One element needed for methods like PGD and CBPG is the computation of the step size. Essentially, we need to be able to compute or have a good upper bound for $\left\|Z^\top Z\right\|_2$ or for a fixed $i \leqslant p$, $\left\|Z^\top_{\mathcal{B}_q(i)} Z\right\|_2$. To do so, several methods are available. Below we present the one used. We compare the upper bounds obtained with different methods.

First in the PGD, for $\left\|Z^\top Z\right\|_2$, we could simply use the power method (Algorithm 4, Golub and Van der Vorst, 2000) to get the exact value. This algorithm iterates through what is called the Krylov

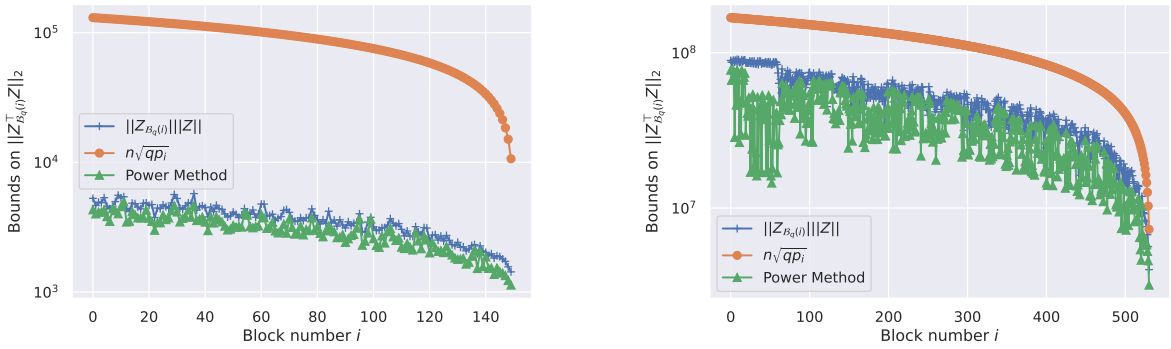
subspace of the matrix A and an associated vector v_1. More formally, we denote the Krylov subspace of dimension $j \in \mathbb{N}^*$ $\mathcal{K}_j(A, v_1)$ and define it as:

$$\mathcal{K}_j(A, v_1) = \left\{ v_1, Av_1, \dots, A^{j-1}v_1 \right\} \ .$$

Algorithm 4: Power method on a matrix A

Input : $A \in \mathbb{R}^{n \times n}$, m number of iterations
v_1 random unit vector
For $j = 1, \dots, m-1$ **do**
  $v_{j+1} \longleftarrow Av_j$ // visit a new direction
  $v_{j+1} \longleftarrow \frac{v_{j+1}}{\|v_{j+1}\|}$ // normalize the output
Output : v_m

However, using this method on CBPG with p blocks becomes very costly (especially considering that $Z_{\mathcal{B}_q(i)}^\top Z$ is not squared so we have to apply the power method on $Z^\top Z_{\mathcal{B}_q(i)} Z_{\mathcal{B}_q(i)}^\top Z$). An alternative is to consider not the exact value but an upper bound with reasonable computation cost. Such a bound could be computed from the consistency of the operator norm, *i.e.*, $\left\| Z_{\mathcal{B}_q(i)}^\top Z \right\|_2 \leqslant \left\| Z_{\mathcal{B}_q(i)} \right\|_2 \left\| Z \right\|_2$.



(a) $p = 150$ and $n = 100$ on simulated dataset from the `make_regression` function of `Scikit-learn`.

(b) $p = 531$ and $n = 19393$ on a genomics dataset (see Chapter 4)

Figure 3.3: Comparison of upper bounds on two datasets for $\left\| Z_{\mathcal{B}_q(i)}^\top Z \right\|_2$.

As we see in Figure 3.3, the upper bound produced can lead to quite far results. Those could imply very slow convergence (see Figure 3.4). Contrary to the power method, it is actually very fast to compute an upper bound using the Cauchy-Schwarz inequality. So it is reasonable for some situations to explain how to compute it.

For the next part, recall that $Z \in \mathbb{R}^{n \times q}$ and $Z_{\mathcal{B}_q(i)} \in \mathbb{R}^{n \times p_i}$. For any matrix $K = (k_{ij})_{i,j} \in \mathbb{R}^{n_1 \times n_2}$, we have (Vandebril, Van Barel, and Mastronardi, 2007, p. 27):

$$\|K\|_2 \leqslant \sqrt{n_1 n_2} \max_{i,j} |k_{ij}| \ . \tag{3.1}$$

Applying (3.1) to $K = Z_{\mathcal{B}_q(i)}^\top Z$, for each $1 \leqslant i \leqslant p$, we could compute K by blocks and get the maximum component each time. But we can even go faster using Cauchy-Schwarz inequality. Indeed, we want to perform our algorithms on the standardized matrix Z. So for any i, K is only a subset of the Gram matrix made from Z. Thus for $l \in [p_i]$ and $m \in [q]$:

$$|k_{lm}| = \left| \left\langle Z_{\tau_q(i,l)} \,|\, z_m \right\rangle \right| \leqslant \left\| Z_{\tau_q(i,l)} \right\|_2 \left\| z_m \right\|_2 \leqslant n \ ,$$

because with a standardized matrix Z, $\|z_m\|_2^2 = n \,\mathbb{V}\mathrm{ar}(z_m) = n$. And this bound is reached when $\tau_q(i,l) = m$, so for each block $Z_{\mathcal{B}_q(i)}$, we have:

$$\left\| Z_{\mathcal{B}_q(i)}^\top Z \right\|_2 \leqslant n\sqrt{qp_i} \ .$$

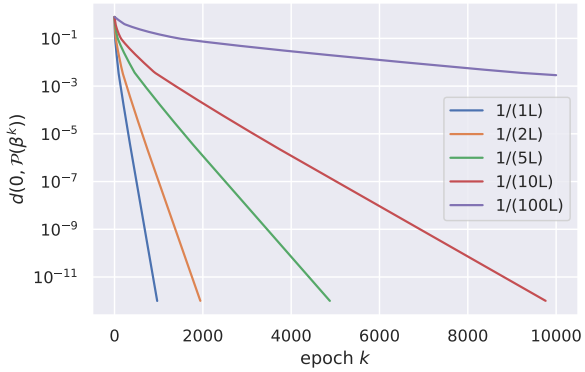


Figure 3.4: Importance of the step size for ridge regularization (`Numpy` with `float64` precision with $\lambda = 0.1$) on the Leukemia dataset ($n = 72$, $p = 7130$). L is the Lipschitz constant $\|X^\top X\|_2/n$. The linear convergence rate (see Proposition A.2.1) is obtained with steps near L^{-1}. Smaller steps lead to $\mathcal{O}(1/k)$ convergence rates.

The goal now is to improve the time consumption of the estimation of the largest singular value without losing too much in precision. We can indeed do better (see Figure 3.5) than the power method using the Lánczos algorithm (Lánczos, 1952) without diminishing the precision. The core idea is quite simple. Where the power method iterates through the Krylov space and only keeps the last iterate to generate a new direction, Lánczos uses more deeply the Krylov space exploration made in all the past iterations. Note that this is not a method to directly compute the largest eigenvalue, but more generally it is a way to approximate all of the extreme eigenvalues of a (potentially very large) hermitian matrix by considering a smaller matrix T. We have the relation

$$T = V^* H V \ ,$$

thus if x is an eigenvector of a matrix H, then Vx is an eigenvector of T for the same eigenvalue, where V and T are constructed by Lánczos method.

Algorithm 5: Lánczos algorithm on an hermitian matrix H

Input : $H \in \mathbb{R}^{n\times n}$, m dimension of the Krylov subspace visited
v_1 random unit vector
For $j = 1, \dots, m-1$ **do**
  $w_{j+1} \longleftarrow H v_j$ // visit a new direction
  $t_{k,j} \longleftarrow \langle v_k, w_{j+1} \rangle$, $k = 1, \dots, j$
  $w_{j+1} \longleftarrow w_{j+1} - \sum_{k=1}^{j} t_{k,j} v_k$ // orthogonalization
  $t_{j+1,j} \longleftarrow \|w_{j+1}\|_2$
  $v_{j+1} \longleftarrow \frac{w_{j+1}}{t_{j+1,j}}$
Output : T

Algorithm 5 is quite general but shows that the basis fot this method is indeed the power method, slightly modified. In the case of H hermitian, it is possible to show that T is a tridiagonal matrix. Because it is an important result but not trivial just by looking at Algorithm 5, let us reprove it quickly by induction.

Lemma 3.2.1. *The basis generated by the columns of V: $(v_1, \dots, v_m)_m$ is orthonormal.*

Proof. The proof of this result can be written by induction. Note that the fact that the columns of V

have unit norm is simply the last line of the algorithm. Let $v_1 \in \mathbb{R}^n$ be a unit vector. Then:

$$\begin{aligned}\langle v_2, v_1\rangle &= \left\langle \frac{w_2}{\|w_2\|_2}, v_1 \right\rangle \\ &= \frac{1}{\|w_2\|_2}\langle Hv_1 - \langle v_1, Hv_1\rangle v_1, v_1\rangle \\ &= \frac{1}{\|w_2\|_2}\langle Hv_1, v_1\rangle - \frac{1}{\|w_2\|_2}\langle v_1, Hv_1\rangle \underbrace{\|v_1\|_2^2}_{=1} \\ &= 0 \ .\end{aligned}$$

If $(v_1, \dots, v_j)_{j\in\mathbb{N}^*}$ is an orthonormal basis, then for $i \in [j]$,

$$\begin{aligned}\langle v_{j+1}, v_i\rangle &= \frac{1}{\|w_{j+1}\|_2}\left\langle Hv_j - \sum_{k=1}^{j}\langle v_k, Hv_j\rangle v_k, v_i \right\rangle \\ &= \frac{1}{\|w_{j+1}\|_2}\left[\langle Hv_j, v_i\rangle - \sum_{k=1}^{j}\langle v_k, Hv_j\rangle\langle v_k, v_i\rangle\right] \ .\end{aligned}$$

Finally, we only need to notice that $\langle v_i, v_k\rangle = \|v_i\|_2^2 = 1$ if $i = k$ and 0 o.w. using to the induction hypothesis. Thus $\|w_{j+1}\|_2\langle v_{j+1}, v_i\rangle = \langle Hv_j, v_i\rangle - \langle v_i, Hv_j\rangle = 0$. So the columns of V form an orthonormal basis of a subspace of the Krylov subspace. □

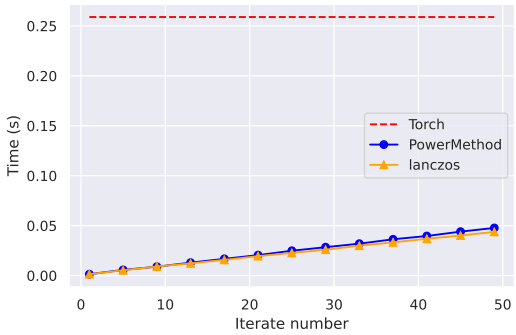
To finalize, because we are in the Krylov subspace, the following relation is of course verified:

$$\forall i < j-1, \exists \alpha_1, \dots, \alpha_k \in \mathbb{R}, \quad Hv_i = \sum_{k=1}^{j-1}\alpha_k v_k \ .$$

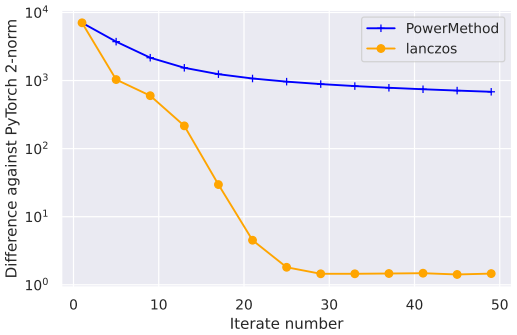
Thus, using Lemma 3.2.1

$$\begin{aligned}t_{i,j} &= \langle v_i, w_{j+1}\rangle = 0, \ |i| < j-1, \\ t_{j-1,j} &= \langle v_{j-1}, w_{j+1}\rangle = v_{j-1}^\top H v_j = v_j^\top H^\top v_{j-1} = \langle v_j, w_j\rangle = t_{j,j-1} \ .\end{aligned}$$

We simulate using standard gaussians $X \in \mathbb{R}^{n\times p}$ and consider $H = X^\top X$. We compare the time and



(a) Time to produce the result (PyTorch `linalg` reference stops without possibility to stop)



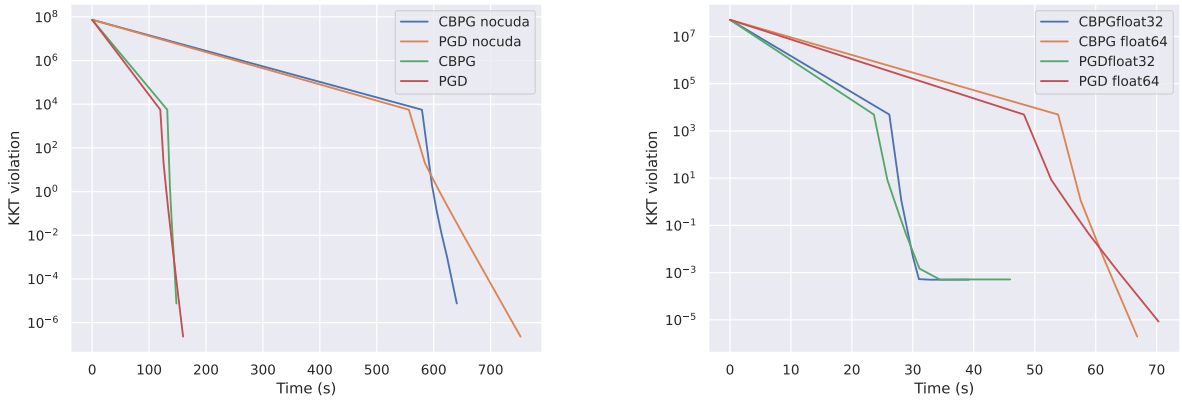
(b) Difference with `linalg` reference value for our iterative methods.

Figure 3.5: Comparison of the linalg module of `Pytorch`, the power method and Lánczos algorithms to get the largest singular value of an hermitian matrix with $n = 20000$ and 500 features. With same time performances Lánczos is much more precise faster.

precision for the power method, Lánczos algorithm and `linalg` module of PyTorch to get $\|T\|_2$. We consider several number of iterations for the power method, and visit a subspace of same dimension with Lánczos method.

3.3 Data types and GPU

For our solvers, the CD method uses `Numba` with `Numpy` as backend. Other solvers use `Pytorch` with `CUDA` enabled. The steps are computed using the Lánczos method and after applying `linalg` function on the matrix T, with a subspace of dimension 20.



(a) KKT violation criterion with CUDA vs no CUDA for PGD and CBPG with simulated data of 20000 samples and 500 features.

(b) Effect of 32 or 64-bits precision on CBPG and PGD methods in a simulated dataset with $n = 25000$ and $p = 400$.

Figure 3.6: Comparison of data types and motivation to use `CUDA` acceleration.

One can always wonder if there is an interest about using a more complex architecture such as GPUs in this framework. To test that, we simulated a dataset with $n = 20000$, $p = 500$. We look at the difference between using `CUDA` or stay purely on CPU for the PGD and CBPG methods. It is clear from Figure 3.6a that, when working with these solvers, using GPU acceleration lead to results available in much fewer time.

It is also well known and documented by libraries (Paszke et al., 2019) that GPUs should be used with `float32` precision for better performance. However, our methods can accumulate some rounding errors that are amplified. Figure 3.6b shows the time-precision trade-off we need to choose from in our computation. As a side-note, it should be noted that to this day in `Numba` when accessing one coefficient from a `Numpy` array the type can be lost. If the original array is in 32-precision, the accessed values are casted to 64-precision internally because of `CPython`.

3.4 Application to simulated dataset

Simulated dataset entries are made from Gaussian variables. On KKT criterion curves, if two consecutive values are close, then we stop the solver. The signal to noise ratio is fixed. For a random centered gaussian noise ϵ of variance σ^2 and a target y, we use:

$$\text{SNR} = \frac{\text{var}(y)}{\sigma^2} \quad .$$

We also need to choose the penalties. For that, let us first look at a upper bound in our search.

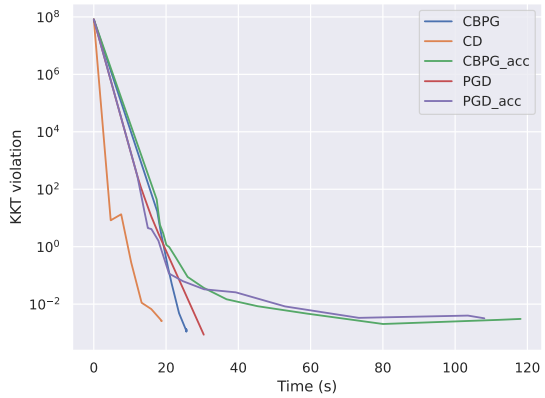
Definition 3.4.1. *For the ℓ_1 penalties, we denote $\lambda_{\max} > 0$ the regularization such that*

$$\forall \lambda > \lambda_{\max},\ \beta = 0_p \text{ and } \Theta = 0_q \quad . \tag{3.2}$$

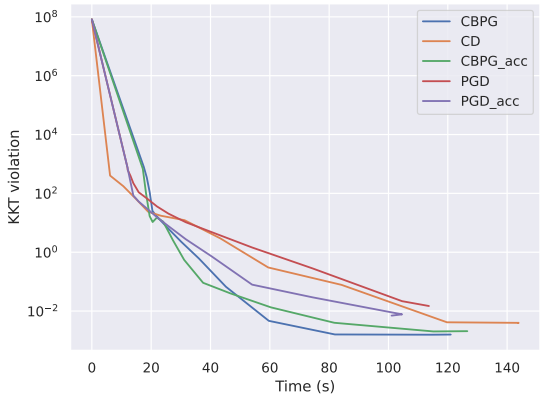
It can be derived from the LASSO (see Section 2.5.1 and (Fercoq, Gramfort, and Salmon, 2015)) that

$$\lambda_{\max} = \max\left(\frac{\left\| X^\top y \right\|_\infty}{n}, \frac{\left\| Z^\top y \right\|_\infty}{n} \right) \quad . \tag{3.3}$$

When a specific ℓ_1 ratio is used (meaning the ℓ_1 penalty is $\alpha\ell_{1_{\text{ratio}}}$ and the ℓ_2 penalty is $\alpha(1-\ell_{1_{\text{ratio}}})$, $\alpha > 0$), then $\lambda_{\max}^{\text{ratio}} = \frac{\lambda_{\max}}{\alpha}$.



(a) ℓ_1 regularizations are set to $\frac{\lambda_{\max}}{10}$ and ℓ_2 regularizations to $\frac{\lambda_{\ell_1}}{10}$.



(b) ℓ_1 regularizations are set to $\frac{\lambda_{\max}}{100}$ and ℓ_2 regularizations to $\frac{\lambda_{\ell_1}}{10}$.

Figure 3.7: KKT criterion curves on the simulated regression dataset with $n = 20000$ and $p = 500$, $q = p(p+1)/2$. Maximum number of epochs is set to 100. Float types are 32-bytes.

For now, we chose to use a simulated regression dataset with $p = 500$ features and $n = 20000$. We also fixed the signal to noise ratio to 10. We denote the regularizations $\lambda_{\max}$ as the maximum between λ_{Θ,ℓ_1} and λ_{β,ℓ_1} ($\simeq 843.228$).

As we see in Figure 3.7a, Coordinate Descent is still faster for high penalties. But decreasing the regularizations to $\lambda_{\max}/100$ with the same signal to noise ratio leads to methods on GPU being equal or faster than coordinate descent. These simulations were made with a sparsity in β and Θ of 25% (meaning 25% of the coefficients were active). We wondered how sparser solutions could affect the solvers.

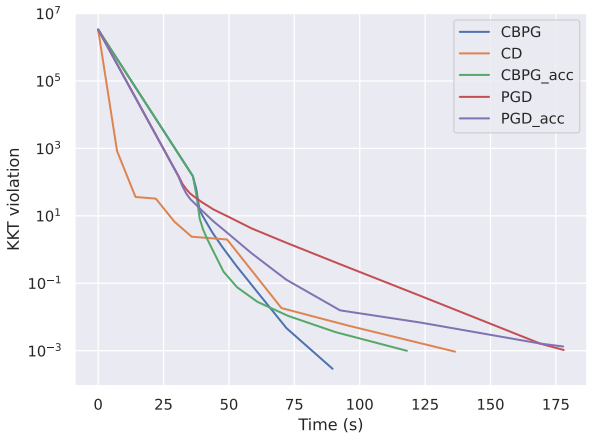


Figure 3.8: KKT criterion curves on the simulated regression dataset with $n = 20000$, $p = 500$ Sparsity levels are set at 1% in β and Θ. The SNR $= 10$, regularizations are set to $\frac{\lambda_{\max}}{100}$, Float-32 types are used and 100 epochs are allowed.

With very sparse solutions, Figure 3.8 shows that Coordinate Descent methods have more advantages and lead over PGD. Here however CBPG methods are still better. We can wonder if dividing

regularizations by such factors are realistic (especially since we set the sparsity level to 1% in our simulated dataset). To investigate that we traced the surface plot of the sparsity level obtained in the vector Θ (very large) against the regularizations. Figure 3.9 simply illustrates on an application how important the choice of the LASSO penalty in the Elastic-Net is to get a good sparsity level in the estimated coefficients. We denote $\text{nnz}(\Theta) = \#\{j \in [q] : \Theta_j \neq 0\}$. The higher the ℓ_1 penalty, the sparser the solution

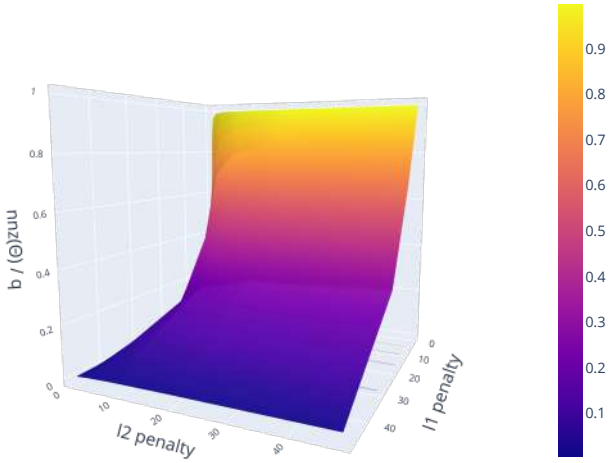
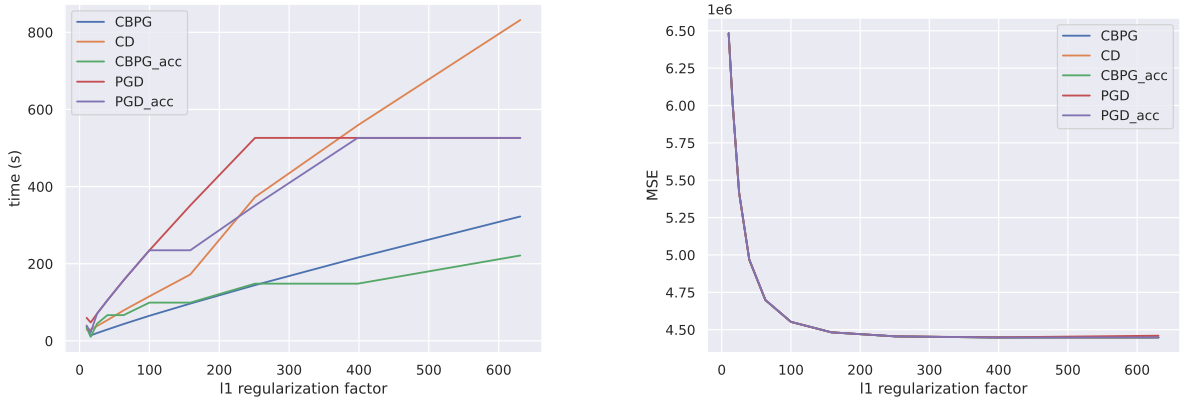


Figure 3.9: Sparsity obtained in Θ with $n = 20000$, $p = 500$, SNR $= 10$, sparsity level simulated of 1% in β and Θ, Float-32 types and 100 epochs allowed.

is and on datasets with p large, we want very sparse solutions. For example, even having 1% of non zero values for $p = 500$ means around 1252 active features in the interactions, which in practice is still a lot and too difficult to interpret.

We see that our GPU-accelerated methods are faster with smaller regularizations and might seem not very useful, albeit it is not often that we know per advance the right regularization to choose from. In most cases, a set (or grid) of parameters is given to our method, we look at the obtained Mean Squared Error (or any other criterion) and select the regularization that lead to the best performance. The key



(a) Time consumption with varying ℓ_1 penalty.

(b) MSE with varying ℓ_1 penalty.

Figure 3.10: On simulated regression dataset with $n = 20000$ and $p = 500$, $q = \frac{p(p+1)}{2}$. ℓ_1 penalty is $\lambda_{\max}/\ell_1$ factor. Maximum number of epochs is set to 500 but solvers stop if the criterion is achieved. Float types are 32-bytes. KKT criterion is set to 10^{-3}. SNR is set to 10. Sparsity levels are at 1%.

hypothesis is that we might not have any prior idea about the location of the best parameters. In this situation, one can choose a grid search of $\lambda_{\max}/\ell_1$ factor for the ℓ_1 regularization. Let us use $\lambda_{\ell_2} = \frac{\lambda_{\max}}{10}$

and see for varying ℓ_1 regularizations the time spent and the MSE obtained. Also, doing so we don't need to compute each time the Lipschitz constants of all the blocks in the Cyclic Block method, they are computed for one run and the next ones can reuse them.

As we see in Figure 3.10b all our errors are close. And with Figure 3.10a we see the three situations we encountered before. Around a division of $\lambda_{\max}$ by 10 and before, CD is faster. After that CD is better than PGD-type methods but not CBPGs. And then it needs to search even longer. Note that in this experiment, only PGD methods did not reach the $\epsilon = 10^{-3}$ precision in the 500 epochs allowed for the last regularizations. Hence the flattening curve meaning we only see the time it takes to make the maximum number of epochs. And although it appears the MSE curve is strictly decreasing, in fact the MSE for the last regularization is higher than the one for the penultimate's.

With warmstart and residuals recomputation We describe in Section 4.2 ways to reach lower precision. The warmstart uses solutions from one penalty to be injected as first guess for the next penalty. Using a ℓ_1 ratio of 0.9 we can, on simulated data, compute a path and compare times to reach a precision of 10^{-4} (see Figure 3.11).

Table 3.1: Non zero values in β and Θ from the simulate experiment of Figure 3.11. Both methods lead to the same results. The ℓ_1 ratio is set to 0.9. Higher values of α lead to λ_{β,ℓ_1} and λ_{Θ,ℓ_1} larger and result to fewer features selected.

α	851.7	516.1	312.6	189.4	114.8	69.5	42.1	25.5	15.5	9.4
$nnz(\beta)$	1	4	5	5	5	5	22	53	93	124
$nnz(\Theta)$	0	0	0	0	89	706	5068	13899	22844	29767

Both solvers keep the same number of non zeros coefficients (see Table 3.1) and result to the same MSE. We could also consider different values for the ℓ_1 ratio and more values of α in the grid to select the penalties resulting with the least MSE on the test set.

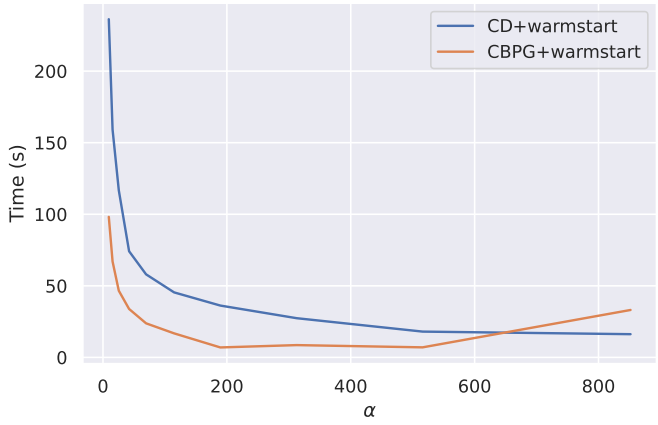


Figure 3.11: Use of the recomputation of the residuals on simulated dataset with a ℓ_1 ratio of 0.9, $n = 20000$, $p = 500$ (15000 samples for the train phase and the rest for the test), 1% of non zero values in the coefficients vectors and solvers are stopped when the KKT criterion reaches 10^{-4}. The ℓ_1 penalty is $\alpha\ell_{1\,\text{ratio}}$ with 10 values of $\alpha \in [\lambda_{\max}/100, \lambda_{\max}/1.1]$ log-spaced (consider $\lambda_{\max}$ as upper bound would lead to zero-filled vectors which is not interesting).

Note that in this situation, for the last penalty ($\alpha = 10^{-2}\lambda_{\max}$) which is the hardest to compute, both solvers converged. Coordinate Descent reached 10^{-4} precision in 21 epochs, and Coordinate Block Gradient Descent in 57 epochs. So more than twice the number of epochs, only each one is done faster thanks to the GPU so *in fine* and counterintuitively, the method doing more epochs is faster.

4 Application to genomics dataset

4.1 Data presentation

As a real high-dimensional dataset, we used a genomics dataset (Bessière et al., 2018). It has a sample size of $n = 19393$ and $p = 531$ features. This means that the matrix Z would have 141246 features, *i.e.*, way too many to handle directly. The n samples each represent a gene. We chose the first response of the dataset to work, meaning one patient. The goal of this data is to identify the active regions for the expression of different genes. The predictive features are:

- 20 nucleotides/dinucleotides for the Core region promoter,
- 20 nucleotides/dinucleotides for the DU region promoter (Distal Upstream),
- 20 nucleotides/dinucleotides for the DD region promoter (Distal Downstream),
- 471 motif scores computed from the Core region (JASPAR 2016 PWM scores).

Definition 4.1.1. *The Position Weight Matrix (PWM) for nucleotide is a matrix with rows A, T, C and G. In each column (position) we count the number of nucleotides of each type present from our samples and from there compute the probability distributions by dividing by the total number.*

The features from the three regions represent the percentages of nucleotide in the sequences divided by the length of the sequence. The motives scores are computed as the sum over a motif w of the log of:

$$\frac{\mathbb{P}(s_{i+j} \,|\, w_j)}{\mathbb{P}(s_{i+j})} \,,$$

where the numerator is the probability for the nucleotide at position $i + j$ of a sequence s to be in position j of the motif w. The maximum is taken over the sequence. This is computed from the PWM (see Section 4.1.2).

4.1.1 Crash course in biology

To understand a little more the idea of the importance of the data considered, and why feature selection is necessary, we need to go back and understand the biological phenomenon measured. This is in fact linked very closely to the process of going from the double helix shaped DNA gene to the protein. This process is visualized in Figure 4.1.

A gene is composed of nucleotide sequences. Very simplified, it is made of a promoter region, followed by exon and introns regions. The promoter is where the transcription is initiated, it controls how the gene is expressed. Its Core region is known to be the one to put in place the transcription factors while distal regions are regulatory elements mainly. During the transcription, an helix is copied from the DNA to make the RNA from the introns and exons. After that, the maturation of the RNA to mRNA removes the introns. This is made in two step with the pre-mRNA to only keep the regions needed (exons) and translate some regions. From there, we go to the cytoplasm of the cell to execute the translation. This is where the ribosome reads the mRNA codons (groups of three nucleotides) until reaching a ”stop codon” and forms the associated amino acid chain. The amino acid elements are linked

with peptic bounds. From there, post translational modifications are made to reach the protein, but the polypeptic chain is the main resource for that.

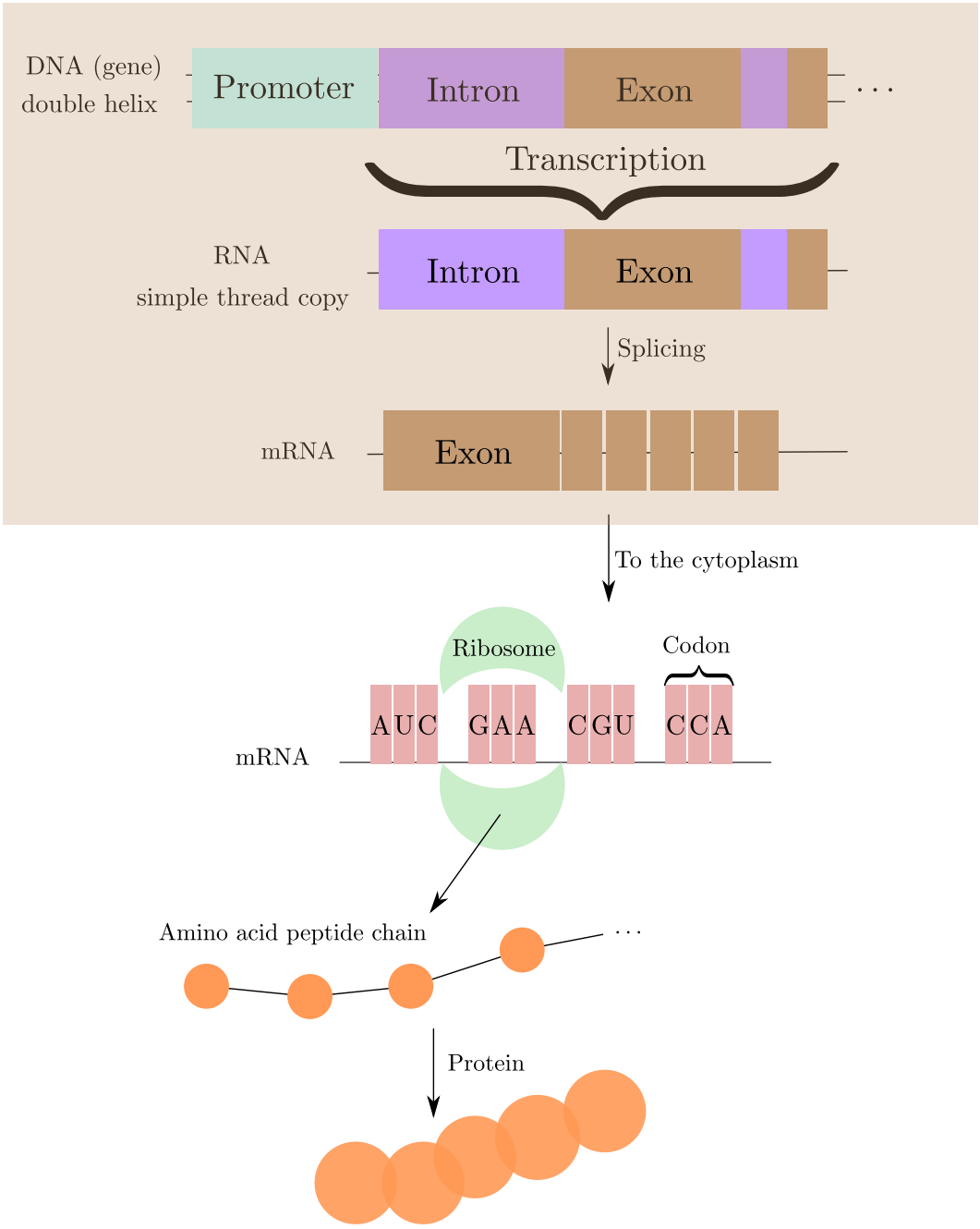


Figure 4.1: From the DNA to the protein: transcription and traduction steps. Note that in the RNA, the uracile replaces the thymine.

Our response variable is the gene expression. Quantifying it means counting the number of corresponding messenger RNA inside the cytoplasm. So technically it should be in $\mathbb{N}$. However, to make it a little more Gaussian-like, a log transformation was applied to the measurements. Of course, since there were zeros inside the original counts, the logarithm was not applied directly but to an $\epsilon > 0$ translated quantity. This creates a bimodal distribution (see Figure 4.2) instead of the unimodal Gaussian.

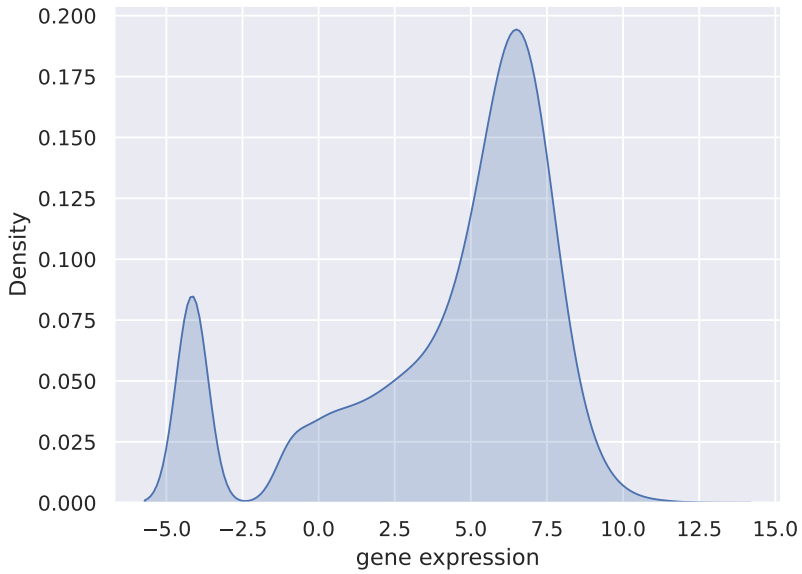


Figure 4.2: Distribution of the gene expression for the first patient. The first peak corresponds to the shift of ϵ to avoid $-\infty$ values in the log.

4.1.2 Construction example for the PWM

Say we have 6 individuals with the followings nucleotide sequences:

Table 4.1: Example of imagined sequences of nucleotide available

sample number	1	2	3	4	5	6
motif	AATCG	ACTCC	ATGGC	TCGCA	TAAGC	ATCCG

Then the PWM is simply (with ATCG as rows from top to bottom):

$$PWM = \begin{bmatrix} \frac{\sharp\{A \text{ in first place}\}}{6} & \frac{\sharp\{A \text{ in second place}\}}{6} & \frac{\sharp\{A \text{ in third place}\}}{6} & \frac{\sharp\{A \text{ in fourth place}\}}{6} & \frac{\sharp\{A \text{ in fifth place}\}}{6} \\ \frac{\sharp\{T \text{ in first place}\}}{6} & \frac{\sharp\{T \text{ in second place}\}}{6} & \frac{\sharp\{T \text{ in third place}\}}{6} & \frac{\sharp\{T \text{ in fourth place}\}}{6} & \frac{\sharp\{T \text{ in fifth place}\}}{6} \\ \frac{\sharp\{C \text{ in first place}\}}{6} & \frac{\sharp\{C \text{ in second place}\}}{6} & \frac{\sharp\{C \text{ in third place}\}}{6} & \frac{\sharp\{C \text{ in fourth place}\}}{6} & \frac{\sharp\{C \text{ in fifth place}\}}{6} \\ \frac{\sharp\{G \text{ in first place}\}}{6} & \frac{\sharp\{G \text{ in second place}\}}{6} & \frac{\sharp\{G \text{ in third place}\}}{6} & \frac{\sharp\{G \text{ in fourth place}\}}{6} & \frac{\sharp\{G \text{ in fifth place}\}}{6} \end{bmatrix}$$

$$\simeq \begin{bmatrix} 4/6 \simeq 0.67 & 2/6 \simeq 0.33 & 1/6 \simeq 0.17 & 0 & 1/6 \simeq 0.17 \\ 0.33 & 0.33 & 0.33 & 0 & 0 \\ 0 & 0.33 & 0.17 & 0.67 & 0.5 \\ 0 & 0 & 0.33 & 0.33 & 0.33 \end{bmatrix}.$$

From there, then a posteriori,

$$\begin{aligned} \mathbb{P}(ACCGA \,|\, w) &= \mathbb{P}(\{A \text{ in first place}\} \,|\, PWM) \times \mathbb{P}(\{C \text{ in second place}\} \,|\, PWM) \ldots \mathbb{P}(\{A \text{ in fifth place}\} \,|\, PWM) \\ &\simeq 0.67 \times 0.33 \times 0.17 \times 0.33 \times 0.17 \ . \end{aligned}$$

4.1.3 Numerical stability

With small visualizations, we can already foresee the issues we will face and discuss more in Section 4.1.3. For example, we can look at a kernel density estimation of the distribution of the first five features of our data and their joined distribution.

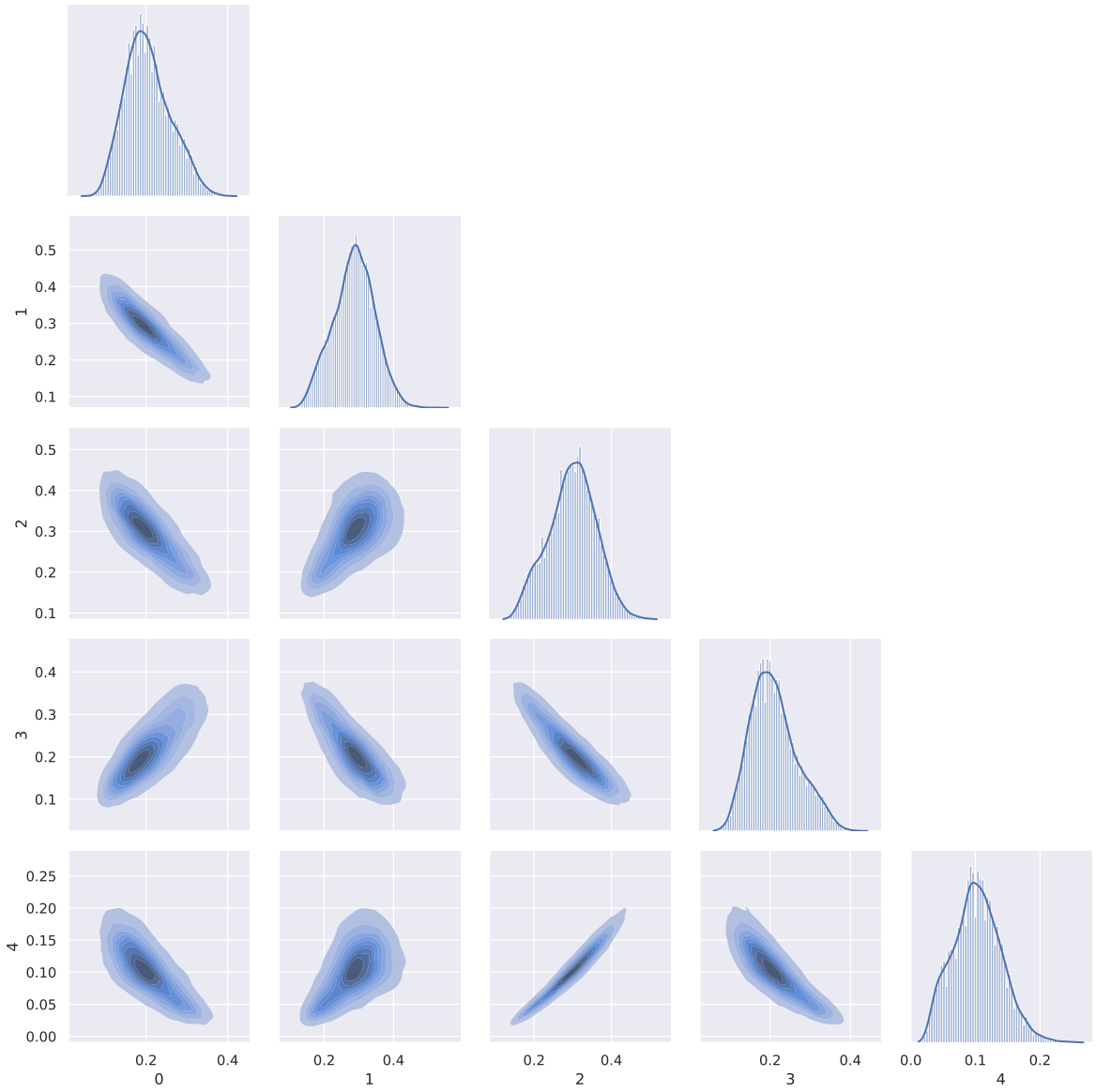


Figure 4.3: Estimated densities and joined densities of the first five features in Core region.

Figure 4.3 already shows us highly correlated features. However, we also notice that the features are unimodal, bell-shaped distributed. With Figure 4.6 we will see that it is not only a phenomenon on the first features but on the all first 60 features (the three regions). For the PMW scores, with Figure 4.4 we see that they are mainly close to one, mostly because what is used is the max of the sum of logs. This property is verified along all the 471 features.

We apply the usual way to preprocess the data $X \in \mathbb{R}^{n \times p}$ for a dataset: the standardization from Equation (4.1):

$$\frac{X - \mu}{\sigma} \ , \tag{4.1}$$

with μ and σ the mean and standard-deviation of X. We can look at the numerical stability of our data after standardization. One way to measure this is to consider the condition number of the data.

Definition 4.1.2. *The condition number of a rectangular matrix A denoted $\kappa(A)$ is:*

$$\kappa(A) = \left\|A\right\|\left\|A^{+}\right\| \ ,$$

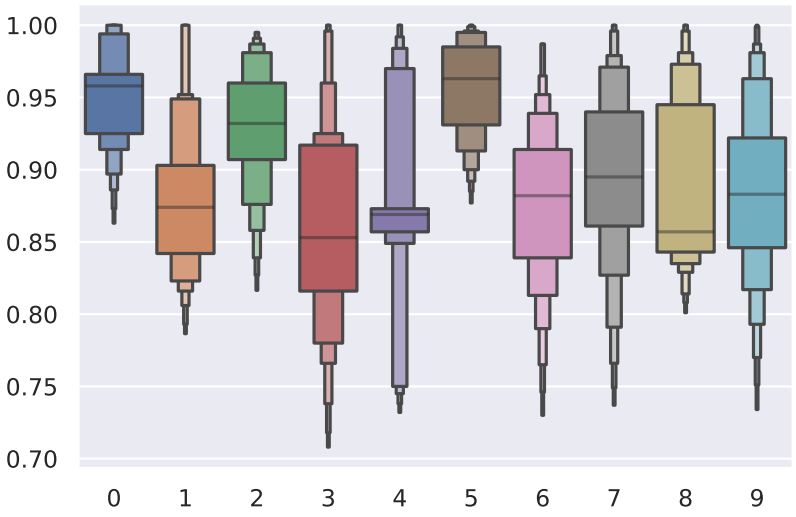


Figure 4.4: Boxplot with the 10 quantiles of the features 80 to 90 (motif scores).

where A^+ is the pseudo-inverse of A. Using the $2-$norm,

$$\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}} \ ,$$

the ratio of the largest and smallest singular values.

With simulated aforementioned data from Gaussian distributions, $\kappa(X) \simeq 1.7$ which is a very good condition number as the best possible is $\kappa(\mathrm{Id}) = 1$. And all the condition numbers of the blocks of the associated matrix were below 2. With Figure 4.5 we can look at the condition number of each block of the matrix Z for the genomics dataset. And for the first 60 blocks it is way too big.

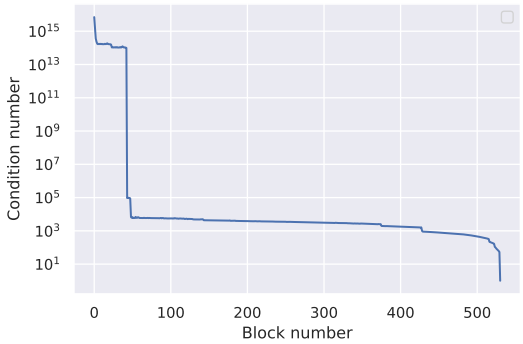
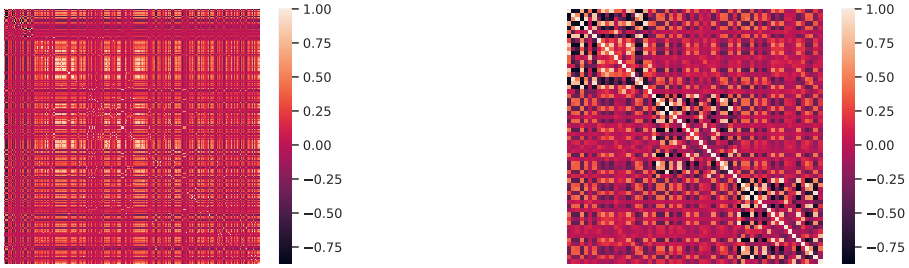


Figure 4.5: Condition number of the blocks of the interaction matrix with genomics data.

In fact, it is not even precise to talk about 10^{15} as a condition number for $Z_{\mathcal{B}_q(1)}$ as $\sigma_{\min} < 10^{-20}$ so we have already reached (by far) the zero-machine precision. If the smallest singular value is so close to zero, then the columns are almost linearly dependant. Figure 4.6 represents the Gram matrix of the genomics dataset, $\Sigma = X^\top X/n$. As we can see, the first 60 features have indeed a different behaviour and there are some very highly correlated features (in absolute value). This alone justifies the use of the Elastic-Net, the ℓ_2 norm being used as a regulator.



(a) All features

(b) First 60 features of the genomics dataset

Figure 4.6: Gram matrices of the genomics dataset.

4.2 Running solvers on the genomics dataset

When working with such data, it is not possible to consider one set of penalties directly. So what we need to look at is the time taken to complete a whole path. And it is interesting to consider different values of $\epsilon > 0$ as stopping values for Equation (2.19), the KKT criterion. There are also two new elements to consider:

- because we are working with a path, we can consider to use warm starts. Meaning that the path is computed from the largest ℓ_1 penalty to the smallest. And when starting to solve the problem with the next penalty, instead of starting from the zero-solution, we can use the solution we previously ended up with.
- to help with numerical stability on the GPU methods, we recompute every 100 epochs the residuals $r = X\beta + Z\Theta - y$. This is done for the first penalty only (to have a solver that converged when the warm start is tried in the accelerated version) or for all the penalties o.w..

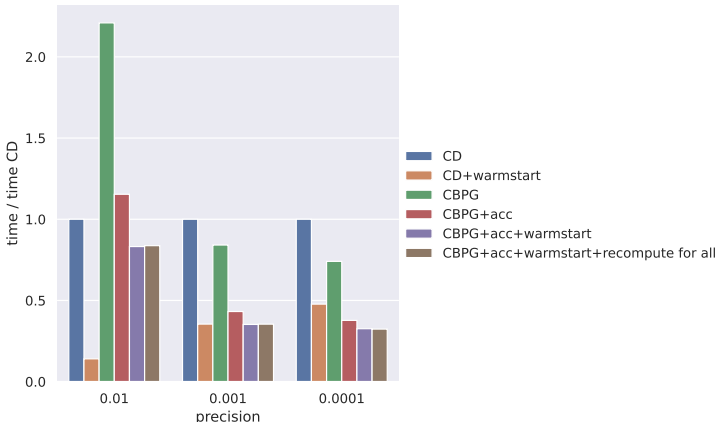


Figure 4.7: Barplot of the ratio of time taken to compute a full path w.r.t. vanilla CD method for different precision. We use the KKT criterion on the genomics dataset. As ℓ_1 penalty, 10 log-spaced values on a grid from $\lambda_{\max}$ to $\lambda_{\max}/100$ are used. The ℓ_2 penalty is set to $20\lambda_{\ell_1,\max}$. All solvers reached convergence criterion.

Figure 4.7 shows that CD methods are quick to find where to go to reach the 10^{-2} precision. However as the ϵ becomes smaller, CBPG based methods perform better. It is worth noticing that using a warm start and recompute residuals for all penalties lead to the same performance results. So the proximal gradient descent based algorithms with GPU can indeed become competitive against Coordinate Descent.

5 The BenchOpt library

Throughout this report, multiple benchmarks were executed, with different datasets, solvers and parameters. As a general rule, scientific results should be reproducible, but in practice, this can become quite tideous and very time consuming. This is where BenchOpt can help.

https://benchopt.github.io/

Created by Thomas Moreau[1], Alexandre Gramfort[2], Joseph Salmon[3], Tom Dupré la Tour[4] and Mathurin Massias[5], this Python library enables the users to easily compare cross-languages optimization solvers and with easy reproducibility. In practice, we already saw an example with Figure 2.5 of what this library can produce. And how it works is quite simple if we look at what is an optimization problem. Indeed, it is made of (at least) three components:

- an objective to minimize,
- a solver that will run until a criterion stops it (the criterion can be a maximum number of epochs or a precision for example),
- a dataset which can be simulated or downloaded.

Each of these components can be matched with a single file. And to make this even easier, templates for each are available on the website and repository. Several optimization problems like the LASSO, Logistic Regression with penalties, Ordinary Least Squares ... are already available on the main repository of the library. And after cloning the chosen one (for example `benchmark_lasso`), a simple

```
$ benchopt run ./benchmark_lasso
```

will run all solvers on the datasets. Option flags are available to only run on some. But sometimes one might want to see how behaves a solver on some examples before using it. And this, without needing to code it and take a certain time to run the comparisons. This is where the other side of the BenchOpt website comes in. This website had a pre-existing base that we worked on during the internship to present more informations to the users.

https://benchopt.github.io/results/

Choose the problem to solve. The `results` part's strength of BenchOpt results in the easy access and visualizations of benchmarks quickly, on a website, but with all the informations that one might need. Currently there are 8 different optimization problems available (see Figure 5.1) and several benchmarks in each.

[1] https://tommoral.github.io/about.html
[2] http://alexandre.gramfort.net/
[3] http://josephsalmon.eu/
[4] https://tomdlt.github.io/
[5] https://mathurinm.github.io/

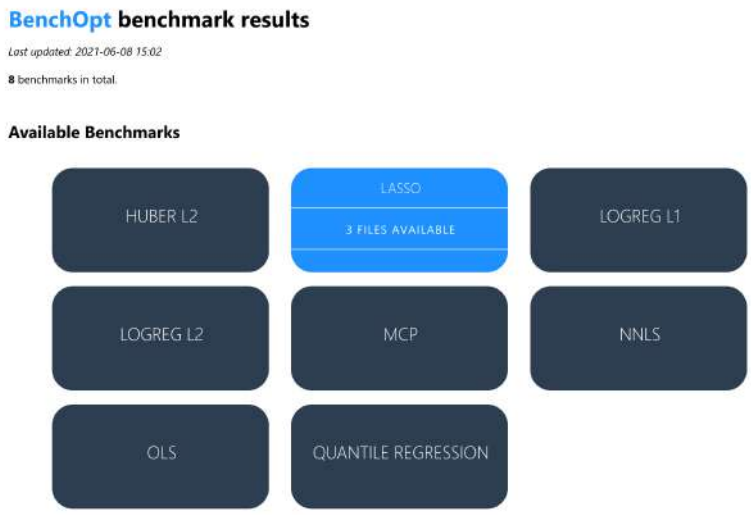


Figure 5.1: Index page of the BenchOpt results webpage. Each problem can be clicked on and lead to the available files. Hovering on one displays the number of files inside.

Choose the benchmark As aforementioned, on one problem we can benchmark several solvers on several datasets. Sometimes, the hardware plays an important role in the resulting performances. For example, with CPUs, an Intel Core i3 can not be expected to be as fast as an Intel Core i9. So for the benchmarks to be useful, hardware related informations are displayed (Figure 5.2), with more informations one click away. As more users come, more examples are needed for the community. With a single command[6] users can publish their results on the website. This participative building can lead to many available files. This is why we created a filter for the main system information in the sidebar (another filter is available above the table where user can directly write words). Users only have to click

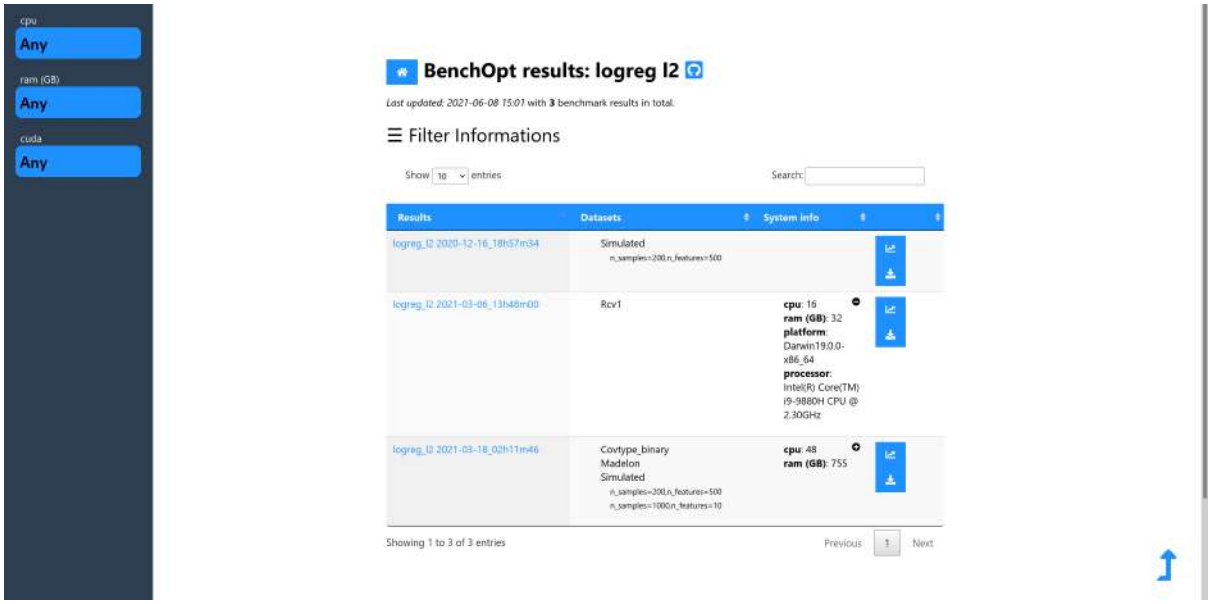


Figure 5.2: Page of the availables benchmark for the Logistic Regression with ℓ_2 penalty. There are three benchmarks available. The last one was run on four different datasets. Only the last two have system information available. Filters are inside the sidebar on the left.

[6] More information at https://benchopt.github.io/publish.html#publish-doc

on the benchmark they want, select the visualization and get an interactive plot using `Plotly` directly available.

Programming side The goal of this static and github-pages supported website is that it can expand easily and with as few maintenance as possible. Typically, if another problem is added, a new *box* does not need to be manually added to the main page. This is made possible by `Mako`[7]. This library lets us use some `Python` programming directly inside an `HTML` template. The syntax is very close to `PHP`, only without the need of a server. `CSS` is loaded to modify the style of the objects (typically the colors, sizes and some animations). And finally `Javascript` performs actions like on-click responses when the user clicks on a button (like with the filters).

Portability Hovering, appearing sidebar, buttons and on-click actions are often well working on computers. However, mobile devices are a non-negligible part of devices from which we browse the internet everyday. So we adapted the website to mobile devices, for example by removing the sidebar and using a filter menu (Figure 5.3) to have a better experience.

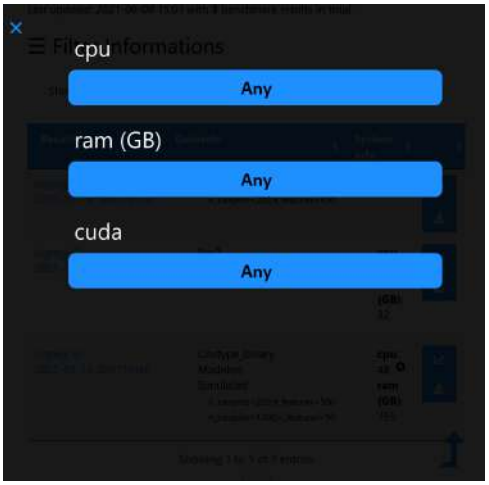


Figure 5.3: Mobile filtering is not made with the sidebar, but in a menu over the page. This allows flexibility for different smaller screen sizes.

Results page After selecting the benchmark of your choice, the page of the results appear. In the system informations, we have besides the number of CPUs, ram, platform and processor that were available before. Added to them are the `Numpy` and `Scipy` versions alongside the `Blas` and `Lapack` libraries. Several dropdown menus allow us to change the datasets, objectives measured and also the kind of the plot (suboptimality curve, histogram,...). One last button was added using `PlotlyJs` to be able to toggle between a plot in *loglog* scale and *semilog-y*.

[7] https://www.makotemplates.org/

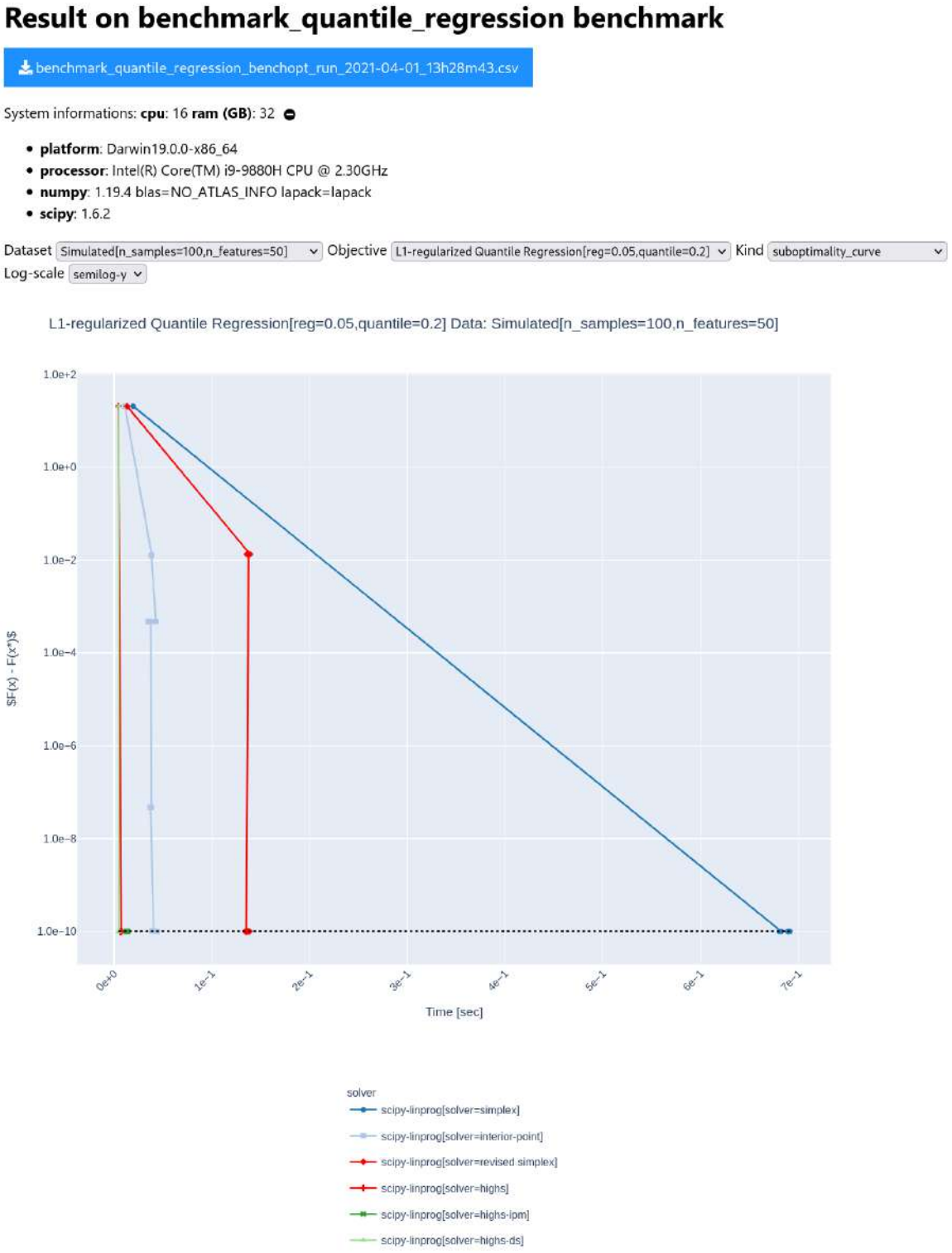


Figure 5.4: Page of the results for the quantile regression.

6 Conclusion

We have presented an implementation of gradient descent methods using the GPU to solve a penalized linear model with first order interactions in high dimensional setting. Different optimization algorithms exploiting the structure of the problem have been compared against vanilla (and warm) coordinate descent. We have applied our algorithms to both simulated and real datasets using the distance to the subdifferential as stopping criterion for our estimator. We have also noticed that the condition number and the values of the penalties lead to different performances. Overall, on a path with highly regularized genomics data, accelerated proximal gradient descent with `CUDA` backend lead to better time performances. We also have presented a way to make easy and reproducible benchmarks with the `BenchOpt` library. Visualizations being a key element, we improved their website for a better-looking experience and device-free access.

Possible future work could be about looking for better bounds for the convergence of the distance to the subdifferential. Improvements on the `BenchOpt` website and library are also still being made. It could also be interesting to look at the performances of the algorithms with `CUDA` against current optimization algorithms into the `R` libraries. And finally, we used the product to generate our interactions. For biologists, it is often interesting to consider min or max functions instead. They lead in practice to different behaviors in time consumption that could also be studied.

Bibliography

Bach, F. (2021). *Learning Theory from First Principles*.

Bascou, F., S. Lèbre, and J. Salmon (2020). "Debiasing the Elastic Net for models with interactions". In: *Journées de Statistique*.

Beck, A. (2017). *First-Order Methods in Optimization*. Vol. 25. SIAM.

Bertrand, Q. and M. Massias (2021). *Anderson acceleration of coordinate descent*. arXiv: 2011.10065 [stat.ML].

Bessière, C. et al. (Jan. 2018). "Probing instructions for expression regulation in gene nucleotide compositions". In: *PLOS Computational Biology* 14.1, pp. 1–28.

Chen, Z., Y. Li, and J. Lu (2021). "On the global convergence of randomized coordinate gradient descent for non-convex optimization". In: *arXiv preprint arXiv:2101.01323*.

Combettes, P. and J-C. Pesquet (2011). "Proximal splitting methods in signal processing". In: *Fixed-point algorithms for inverse problems in science and engineering*. Springer, pp. 185–212.

Fajardo, MD., J. Vicente-Pérez, and MML. Rodríguez (2012). "Infimal convolution, c-subdifferentiability, and Fenchel duality in evenly convex optimization". In: *Top* 20.2, pp. 375–396.

Fercoq, O., A. Gramfort, and J. Salmon (2015). "Mind the duality gap: safer rules for the lasso". In: *ICML*, pp. 333–342.

Feydy, J. et al. (2020). "Fast geometric learning with symbolic matrices". In: *Proc. NeurIPS* 2.4, p. 6.

Golub, GH. and HA. Van der Vorst (2000). "Eigenvalue computation in the 20th century". In: *Journal of Computational and Applied Mathematics* 123.1-2, pp. 35–65.

Hale, E., W. Yin, and Y. Zhang (2008). "Fixed-point continuation for \ell_1-minimization: Methodology and convergence". In: *SIAM Journal on Optimization* 19.3, pp. 1107–1130.

Johnson, T. and C. Guestrin (2015). "Blitz: A principled meta-algorithm for scaling sparse optimization". In: *International Conference on Machine Learning*. PMLR, pp. 1171–1179.

Lam, SK., A. Pitrou, and S. Seibert (2015). "Numba: A llvm-based python jit compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6.

Lánczos, C. (1952). "Solution of systems of linear equations by minimized iterations". In: *J. Res. Nat. Bur. Standards* 49.1, pp. 33–53.

Le Morvan, M. and J-P. Vert (2018). "WHInter: A Working set algorithm for High-dimensional sparse second order Interaction models". In: *International Conference on Machine Learning*. PMLR, pp. 3635–3644.

Massias, M. (2019). "Sparse high dimensional regression in the presence of colored heteroscedastic noise: application to M/EEG source imaging". PhD thesis. Telecom Paristech.

Ndiaye, E. et al. (2017). "Gap safe screening rules for sparsity enforcing penalties". In: *The Journal of Machine Learning Research* 18.1, pp. 4671–4703.

Nesterov, Y. (1983). "A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$". In: *Sov. Math. Dokl.* Vol. 27. 2.

— (2012). "Efficiency of coordinate descent methods on huge-scale optimization problems". In: *SIAM Journal on Optimization* 22.2, pp. 341–362.

Parikh, N. and S. Boyd (2014). "Proximal Algorithms". In: *Found. Trends Optim.* 1.3, pp. 127–239.

Paszke, A. et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Tibshirani, R. (1996). "Regression Shrinkage and Selection via the Lasso". In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 58.1, pp. 267–288.

Tikhonov, AN. (1943). "On the stability of inverse problems". In: *Dokl. Akad. Nauk SSSR* 39, pp. 176–179.

Vandebril, R., M. Van Barel, and N. Mastronardi (2007). *Matrix computations and semiseparable matrices: linear systems*. Vol. 1. JHU Press.

Wu, T. and K. Lange (2008). "Coordinate descent algorithms for lasso penalized regression". In: *The Annals of Applied Statistics* 2.1, pp. 224–244.

Zou, H. and TJ. Hastie (2005). "Regularization and variable selection via the elastic net". In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 67.2, pp. 301–320.

A Convergence rates

As a stopping criterion, we choose the distance from 0 to the subdifferential of the primal objective function taken at the current iterate. Convergence rates can be explicited as it is often done for the objectives convergence.

A.1 A simple case: Ordinary-Least-Squares

Let us start by considering a differentiable case. In particular, we investigate the least-squares case:

$$\mathcal{P}_{OLS}(\beta) = \frac{1}{2n}\left\|X\beta - y\right\|_2^2 \ .$$

We aim at upper bounding

$$\Delta\left(\beta^k, \beta^*\right) = d_{\|\cdot\|_\infty}\left(0, \partial\mathcal{P}_{OLS}(\beta^k)\right) - d_{\|\cdot\|_\infty}\left(0, \partial\mathcal{P}_{OLS}(\beta^*)\right) \ , \tag{A.1}$$

where β^k is the k^{th} iterate generated by gradient descent and β^* is a minimizer[1]. Since β^* is a minimizer, we can simplify Equation (A.1). The subdifferential $\partial\mathcal{P}_{OLS}(\beta^*)$ reduces to the singleton $\{\nabla\mathcal{P}_{OLS}(\beta^*)\} = \{0\}$ by optimality. Hence, to upper bound Equation (A.1) one can simply upper bound $d(0, \nabla\mathcal{P}_{OLS}(\beta^k)) = \left\|\nabla\mathcal{P}_{OLS}(\beta^k)\right\|_\infty$.

Proposition A.1.1. *For $X \in \mathbb{R}^{n\times p}$, $k \geqslant 0$, $\gamma = 1/L$ with L (resp. μ) the largest (resp. smallest) eigenvalue of the Hessian of $\mathcal{P}_{OLS}$ at optimum, and $\kappa = L/\mu$ the condition number of X, we have the following inequality:*

$$\left\|\partial\mathcal{P}_{OLS}(\beta^k)\right\|_\infty \leqslant pL\exp\left(-\frac{k}{\kappa}\right)\left\|\beta^0 - \beta^*\right\|_\infty \ . \tag{A.2}$$

Proof. We follow here the line of the proof showing the convergence of the objectives by (Bach, 2021). Remark that the Hessian matrix $(\frac{\partial^2\mathcal{P}_{OLS}}{\partial^2\beta}(\cdot))$ is constant for all $\beta \in \mathbb{R}^p$, so we write H for this matrix. In particular

$$H = \frac{1}{n}X^\top X \ , \tag{A.3}$$

$$H\beta^* = \frac{1}{n}X^\top y \ . \tag{A.4}$$

It follows that

$$\nabla\mathcal{P}_{OLS}(\beta^k) = H(\beta^k - \beta^*) \ . \tag{A.5}$$

We can iterate the gradient descent update formula (Equation (2.7)) with step size γ, leading to:

$$\begin{aligned}\nabla\mathcal{P}_{OLS}(\beta^k) &= H\left(\beta^{k-1} - \gamma\nabla\mathcal{P}(\beta^{k-1}) - \beta^*\right)\\ &\overset{A.5}{=} H(\mathrm{Id} - \gamma H)(\beta^{k-1} - \beta^*)\\ &= H(\mathrm{Id} - \gamma H)^k(\beta^0 - \beta^*) \ .\end{aligned}$$

[1] we assume it to be unique for simplicity.

Taking the infinite norm on both sides, we get:

$$\begin{aligned}\left\|\nabla\mathcal{P}_{OLS}(\beta^k)\right\|_\infty &= \left\|H(\mathrm{Id}-\gamma H)^k(\beta^0-\beta^*)\right\|_\infty \\ &\leqslant |||H|||_\infty \left|\left|\left|(\mathrm{Id}-\gamma H)^k\right|\right|\right|_\infty \left\|\beta^0-\beta^*\right\|_\infty \ ,\end{aligned}$$

where for a matrix $M \in \mathbb{R}^{n\times p}$, $|||M|||_\infty$ is the induced norm of $\|\cdot\|_\infty$ applied to M, defined as:

$$|||M|||_\infty = \max_{j\in[p]} \sum_{i=1}^{n} \left|m_{ij}\right| \ .$$

Using that $|||M|||_\infty \leqslant \sqrt{p}\,|||M|||_2$,

$$\left\|\nabla\mathcal{P}_{OLS}(\beta^k)\right\|_\infty \leqslant \sqrt{p}\,|||H|||_2\,\sqrt{p}\left|\left|\left|(\mathrm{Id}-\gamma H)^k\right|\right|\right|_2 \left\|\beta^0-\beta^*\right\|_\infty \ .$$

Finally, if we denote $\eta \in \mathbb{R}_+^*$ an eigenvalue of the Hessian H, then (using the spectral theorem), for a continuous function φ, $\varphi(\eta)$ is an eigenvalue of $\varphi(H)$. We can use this property to find an upper bound of $\left|\left|\left|(\mathrm{Id}-\gamma H)^k\right|\right|\right|_2$:

$$\begin{aligned}\max_{\eta\in[\mu,L]}(1-\gamma\eta)^k &= \left(1-\frac{\mu}{L}\right)^k \\ &= \left(1-\kappa^{-1}\right)^k \ .\end{aligned}$$

Combining our results leads to Proposition A.1.1:

$$\begin{aligned}\left\|\partial\mathcal{P}_{OLS}(\beta^k)\right\|_\infty &\leqslant pL\left(1-\kappa^{-1}\right)^k \left\|w^0-w^*\right\|_\infty \\ &\leqslant pL\exp\left(-\frac{k}{\kappa}\right)\left\|\beta^0-\beta^*\right\|_\infty \ .\end{aligned}$$

□

A.2 Ridge regularization

Building up to the Elastic-Net, let us consider the Ridge regularization with $\lambda \geqslant 0$ as ℓ_2 penalty. The primal considered is:

$$\mathcal{P}_{Ridge} = \frac{1}{2n}\left\|X\beta-y\right\|_2^2 + \frac{\lambda}{2}\left\|\beta\right\|_2^2 \ . \tag{A.6}$$

Proposition A.2.1. *For $X \in \mathbb{R}^{n\times p}$, $k \geqslant 0$, $\gamma = 1/L$ with L (resp. μ) is the largest (resp. smallest) eigenvalue of the Hessian of $\mathcal{P}$ at optimum, and $\kappa = \mu/L$ the condition number of X, we have the following inequality:*

$$\left\|\partial\mathcal{P}_{Ridge}(\beta^k)\right\|_\infty \leqslant pL\exp\left(-\frac{k}{\kappa}\right)\left\|\beta^0-\beta^*\right\|_\infty \ . \tag{A.7}$$

Meaning that the convergence rate for Ridge regularization and Ordinary-Least-Squares only differs by the condition number.

Proof. The first order conditions give us:

$$\begin{aligned}\nabla\mathcal{P}_{Ridge}(\beta) &= \frac{1}{n}X^\top(X\beta-y)+\lambda\beta \\ &= \left[\frac{1}{n}X^\top X+\lambda\,\mathrm{Id}\right]\beta-\frac{1}{n}X^\top y \\ &= \left[\frac{1}{n}X^\top X+\lambda\,\mathrm{Id}\right](\beta-\beta^*) \ .\end{aligned}$$

Noticing that $\frac{1}{n}X^\top X + \lambda \mathrm{Id}$ is the Hessian matrix of $\mathcal{P}_{Ridge}$, constant for all $\beta \in \mathbb{R}^p$, we thus recover:

$$\nabla \mathcal{P}_{Ridge}(\beta) = H(\beta - \beta^*) \ . \tag{A.8}$$

Equation (A.8) is exactly the same as the OLS case. Thus with the exact same steps we obtain Proposition A.2.1. □

A visualization of the convergence rate in Proposition A.2.1 is available in Figure 3.4.

A.3 LASSO regularization

In Appendix A.1 and Appendix A.2 we used the closed form of the solution. With the LASSO, we do not have this. Let us consider the primal with $\lambda > 0$ as penalty:

$$\mathcal{P}_{LASSO}(\beta) = \frac{1}{2n}\left\|X\beta - y\right\|_2^2 + \lambda \left\|\beta\right\|_1 \ . \tag{A.9}$$

Proposition A.3.1. *Under the assumptions of Proposition A.1.1:*

$$\Delta(\beta^k, \beta^*) \leqslant pL \exp\left(-\frac{k}{\kappa}\right) \left\|\beta^0 - \beta^*\right\|_\infty \ . \tag{A.10}$$

Proof. The subgradient of $\mathcal{P}_{LASSO}$ at iterate $k \geqslant 0$ writes:

$$\partial \mathcal{P}_{LASSO}(\beta^k) = \frac{1}{n}X^\top(X\beta^k - y) + \lambda \partial_{\|\cdot\|_1}(\beta^k)$$

.

Then

$$\begin{aligned}
\Delta(\beta^k, \beta^*) &= d\left(0, \frac{1}{n}X^\top(y - X\beta^k) + \lambda \partial_{\|\cdot\|_1}(\beta^k)\right) \\
&= \left\|\mathrm{ST}\left(\frac{1}{n}X^\top(y - X\beta^k), \lambda\right)\right\|_\infty \\
&\leqslant \left\|\frac{1}{n}X^\top(y - X\beta^k)\right\|_\infty \qquad (A.11) \\
&\leqslant \left\|\nabla \mathcal{P}_{OLS}(\beta^k)\right\|_\infty \ ,
\end{aligned}$$

with Equation (A.11) obtained using that the soft-thresholding operator is non-expansive coordinate-wise (Hale, Yin, and Zhang, 2008, Lemma 3.2) *i.e.*,

$$\forall x, y \in \mathbb{R}^n \ \forall i \in [n], \ |\mathrm{ST}(x, \lambda)_i - \mathrm{ST}(y, \lambda)_i| \leqslant |x_i - y_i| \ .$$

Using Proposition A.1.1 we obtain our upper bound. □

Notice that Equation (A.11) is in fact not threshold-dependant. So in a weighted LASSO problem with penalties $(\lambda_j)_{j=1}^p$, we would have:

$$\begin{aligned}
\Delta(\beta^k, \beta^*) &= \max_{j \in [p]} \left|\mathrm{ST}\left(\frac{1}{n}X_j^\top(y - X\beta^k), \lambda_j\right)\right| \\
&\leqslant \max_{j \in [p]} \left|\frac{1}{n}X_j^\top(y - X\beta^k)\right| \\
&\leqslant \left\|\nabla \mathcal{P}_{OLS}(\beta^k)\right\|_\infty \ .
\end{aligned}$$

Thus leading to the same upper bound.

A.3.1 Elastic-Net regularization

For the Elastic-Net, we consider the primal:

$$\mathcal{P}_{Enet}(\beta) = \frac{1}{2n}\left\|X\beta - y\right\|_2^2 + \lambda_{\ell_1}\left\|\beta\right\|_1 + \frac{\lambda_{\ell_2}}{2}\left\|\beta\right\|_2^2 \ . \tag{A.12}$$

The Elastic-Net being an augmented LASSO problem (Section 2.5.1), we have the following corollary.

Corollary A.3.1. *For* $\widetilde{y} = [y \,|\, 0_p]^\top$, $\widetilde{X} = \begin{bmatrix} X \\ \sqrt{\lambda_{\ell_2} n}\mathrm{Id}_{p\times p} \end{bmatrix}$, *we get:*

$$\Delta(\beta^k, \beta^*) \leqslant pL_{\widetilde{X}} \exp\left(-\frac{k}{\kappa_{\widetilde{X}}}\right)\left\|\beta^0 - \beta^*\right\|_\infty \ ,$$

with $L_{\widetilde{X}} = \frac{\sigma_{\max}(X)+n\lambda_{\ell_2}}{n}$ *and* $\kappa_{\widetilde{X}} = \frac{\sigma_{\max}(X)+n\lambda_{\ell_2}}{\sigma_{\min}(X)+n\lambda_{\ell_2}}$.

Proof. We only need to notice that the spectrum of $\widetilde{X}^\top\widetilde{X}$ is the same as $X^\top X$ shifted of $n\lambda_{\ell_2}$. Then we apply Proposition A.3.1 to Equation (A.9) with $\widetilde{y}$ and $\widetilde{X}$. □

We can visualize the impact of the ℓ_2 regularization on the convergence. Figure A.1 shows that with larger ℓ_2 regularizations, the proximal descent for the Elastic-Net converges faster. This is a visualization of the dependance of Δ by the condition number from Corollary A.3.1.

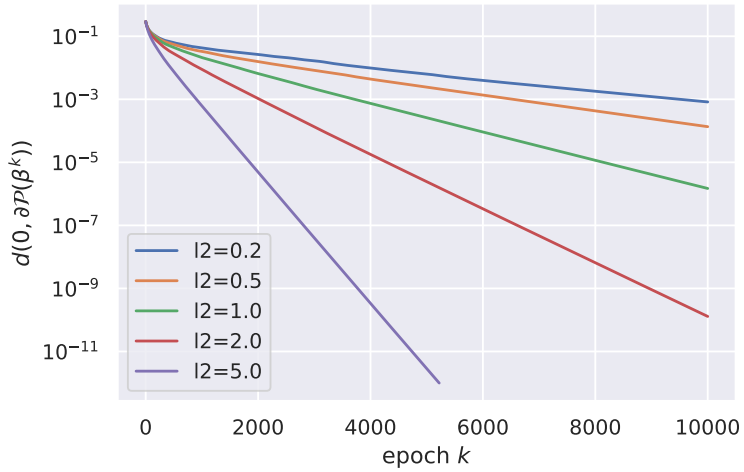


Figure A.1: Elastic-Net with ℓ_1 penalty set to 0.5 and varying ℓ_2 penalties on the leukemia dataset. For information, $\lambda_{\max} \simeq 0.626$ for this dataset.

B Double interactions equivalence

For the interactions, the vector Θ considered was in $\mathbb{R}^{\frac{p(p+1)}{2}}$. This implies that we removed all the feature-interactions which occurred more than once in the matrix Z. However, we could also consider to keep all of the interactions, meaning have $\widetilde{Z} \in \mathbb{R}^{n \times p^2}$ and $\widetilde{\Theta} \in \mathbb{R}^{p^2}$ the full versions, with a supplementary constraint on the symmetry of the coefficients in $\widetilde{\Theta}$. With the notations of Chapter 2, if we exclude redundant interactions, then $K = q = p(p+1)/2$, else $K = \widetilde{q} = p^2$. In the later case, we write $\widetilde{Z}$ the interaction matrix. The problem is then:

$$\widehat{p} \in \min_{\substack{\beta \in \mathbb{R}^p \\ \widetilde{\Theta} \in \mathbb{R}^{p \times p}}} \frac{1}{2n} \left\| y - X\beta - \widetilde{Z}\widetilde{\Theta} \right\|_2^2 + g^{(1)}(\beta) + g^{(2)}(\widetilde{\Theta}) + \iota(\widetilde{\Theta}_{\tau_{\widetilde{q}}(i,j)} = \widetilde{\Theta}_{\tau_{\widetilde{q}}(j,i)}) \ . \tag{$\mathcal{P}_2$}$$

where $\iota(\widetilde{\Theta}_{\tau_{\widetilde{q}}(i,j)} = \widetilde{\Theta}_{\tau_{\widetilde{q}}(j,i)}) = 0$ if the condition is verified and $+\infty$ o.w. First, we study $y - X\beta - \widetilde{Z}\widetilde{\Theta}$, with the symmetry constraint:

$$\begin{aligned} y - X\beta - \widetilde{Z}\widetilde{\Theta} &= y - X\beta - \sum_{(i,j) \in [p]^2} \widetilde{Z}_{ij}\widetilde{\Theta}_{ij} \\ &= y - X\beta - 2\sum_{i \in [p]} \sum_{j > i} \widetilde{Z}_{\tau_q(i,j)}\widetilde{\Theta}_{\tau_q(i,j)} - \sum_{i \in [p]} \widetilde{Z}_{\tau_q(i,i)}\widetilde{\Theta}_{\tau_q(i,i)} \\ &= y - X\beta - \sum_{i \in [p]} \sum_{j > i} \widetilde{Z}_{\tau_q(i,j)} 2\widetilde{\Theta}_{\tau_q(i,j)} - \sum_{i \in [p]} Z_{\tau_q(i,i)}\Theta_{\tau_q(i,i)} \ . \end{aligned}$$

Posing $\Theta_{\tau_q(i,j)} = 2\widetilde{\Theta}_{\tau_q(i,j)}$ for $i \neq j$, we thus get

$$\begin{aligned} \frac{1}{2n} \left\| y - X\beta - \widetilde{Z}\widetilde{\Theta} \right\|_2^2 &= \frac{1}{2n} \left\| y - X\beta - \sum_{i \in [p]} \sum_{j > i} Z_{\tau_q(i,j)}\Theta_{\tau_q(i,j)} - \sum_{i \in [p]} Z_{\tau_q(i,i)}\Theta_{\tau_q(i,i)} \right\|_2^2 \\ &= \frac{1}{2n} \left\| y - X\beta - Z\Theta \right\|_2^2 \ . \end{aligned}$$

We execute the same decomposition on the penalties depending on $\widetilde{\Theta}$. This leads to modifying $g^{(2)}$ as follows:

$$\begin{aligned} g^{(2)}(\widetilde{\Theta}) &= \lambda_{\widetilde{\Theta},\ell_1} \sum_{i \in [p]} \left| \widetilde{\Theta}_{\tau_q(i,i)} \right| + 2\lambda_{\widetilde{\Theta},\ell_1} \sum_{i > j} \left| \widetilde{\Theta}_{\tau_q(i,j)} \right| + \frac{\lambda_{\widetilde{\Theta},\ell_2}}{2} \sum_{i \in [p]} \widetilde{\Theta}^2_{\tau_q(i,i)} + \lambda_{\widetilde{\Theta},\ell_2} \sum_{i > j} \widetilde{\Theta}^2_{\tau_q(i,j)} \\ &= \lambda_{\Theta,\ell_1} \sum_{i \in [p]} \left| \widetilde{\Theta}_{\tau_q(i,i)} \right| + \lambda_{\Theta,\ell_1} \sum_{i > j} \left| \Theta_{\tau_q(i,j)} \right| + \frac{\lambda_{\Theta,\ell_2}}{2} \sum_{i \in [p]} \Theta^2_{\tau_q(i,i)} + \frac{\lambda_{\Theta,\ell_2}}{2} \sum_{i > j} \left(\frac{\Theta_{\tau_q(i,j)}}{\sqrt{2}} \right)^2 \ , \end{aligned}$$

using the same variable change for non-diagonal terms. Considering as reference the problem in dimension $q = p(p+1)/2$, then we only need to multiply instead of dividing in the ℓ_2 penalty. So to resume, solving $(\mathcal{P})$ in dimension q is equivalent to solving:

$$\begin{aligned} \widehat{p} = \min_{\substack{\beta \in \mathbb{R}^p \\ \widetilde{\Theta} \in \mathbb{R}^{p \times p}}} & \frac{1}{2n} \left\| y - X\beta - \widetilde{Z}\widetilde{\Theta} \right\|_2^2 + g^{(1)}(\beta) \\ & + \lambda_{\Theta,\ell_1} \left\| \widetilde{\Theta} \right\|_1 + \frac{\lambda_{\Theta,\ell_2}}{2} \sum_{i \in [p]} \widetilde{\Theta}_{\tau_{\widetilde{q}}(i,i)} + \lambda_{\Theta,\ell_2} \sum_{i \in [p]} \sum_{j > i} \widetilde{\Theta}^2_{\tau_{\widetilde{q}}(i,j)} + \iota(\widetilde{\Theta}_{\tau_{\widetilde{q}}(i,j)} = \widetilde{\Theta}_{\tau_{\widetilde{q}}(j,i)}) \ . \end{aligned} \tag{B.1}$$

We then compute the proximal operator for the $\widetilde{\Theta}$ part with the separability of the components (Beck, 2017, p. 135), which leads to:

- if $i = j$:

$$\operatorname{prox}_{\mu\lambda_{\Theta,\ell_1}\left(|\cdot|+\frac{\lambda_{\Theta,\ell_2}/\lambda_{\Theta,\ell_1}}{2}(\cdot)^2\right)}(t) = \frac{\operatorname{sign}(t)}{1+\mu\lambda_{\Theta,\ell_2}}(|t|-\mu\lambda_{\Theta,\ell_1})_+ \ ,$$

- if $i \neq j$:

$$\operatorname{prox}_{\mu\lambda_{\Theta,\ell_1}\left(|\cdot|+2\frac{\lambda_{\Theta,\ell_2}}{\lambda_{\Theta,\ell_1}}(\cdot)^2\right)}(t) = \frac{\operatorname{sign}(t)}{1+2\mu\lambda_{\Theta,\ell_2}}(|t|-\mu\lambda_{\Theta,\ell_1})_+ \ .$$