Implicit differentiation of Lasso-type models for hyperparameter optimization

Quentin Bertrand, Quentin Klopfenstein, Mathieu Blondel, Samuel Vaiter, Alexandre Gramfort, Joseph Salmon

Implicit differentiation of Lasso-type models for hyperparameter optimization

Quentin Bertrand [*1] **Quentin Klopfenstein** [*2] **Mathieu Blondel** [3] **Samuel Vaiter** [4] **Alexandre Gramfort** [1] **Joseph Salmon** [5]

Abstract

Setting regularization parameters for Lasso-type estimators is notoriously difficult, though crucial in practice. The most popular hyperparameter optimization approach is grid-search using held-out validation data. Grid-search however requires to choose a predefined grid for each parameter, which scales exponentially in the number of parameters. Another approach is to cast hyperparameter optimization as a bi-level optimization problem, one can solve by gradient descent. The key challenge for these methods is the estimation of the gradient *w.r.t.* the hyperparameters. Computing this gradient via forward or backward automatic differentiation is possible yet usually suffers from high memory consumption. Alternatively implicit differentiation typically involves solving a linear system which can be prohibitive and numerically unstable in high dimension. In addition, implicit differentiation usually assumes smooth loss functions, which is not the case for Lasso-type problems. This work introduces an efficient implicit differentiation algorithm, *without* matrix inversion, tailored for Lasso-type problems. Our approach scales to high-dimensional data by leveraging the sparsity of the solutions. Experiments demonstrate that the proposed method outperforms a large number of standard methods to optimize the error on held-out data, or the Stein Unbiased Risk Estimator (SURE).

[*]Equal contribution [1]Université Paris-Saclay, Inria, CEA, Palaiseau, France [2]Institut Mathématique de Bourgogne, Université de Bourgogne, Dijon, France [3]Google Research, Brain team, Paris, France [4]CNRS and Institut Mathématique de Bourgogne, Université de Bourgogne, Dijon, France [5]IMAG, Université de Montpellier, CNRS, Montpellier, France. Correspondence to: Quentin Bertrand <quentin.bertrand@inria.fr>, Quentin Klopfenstein <quentin.klopfenstein@u-bourgogne.fr>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

1. Introduction

In many statistical applications, the number of parameters p is much larger than the number of observations n. In such scenarios, a popular approach to tackle linear regression problems is to consider convex ℓ_1-type penalties, used in Lasso (Tibshirani, 1996), Group-Lasso (Yuan and Lin, 2006), Elastic-Net (Zou and Hastie, 2005) or adaptive Lasso (Zou, 2006). These *Lasso-type* estimators rely on regularization hyperparameters, trading data fidelity against sparsity. Unfortunately, setting these hyperparameters is hard in practice: estimators based on ℓ_1-type penalties are indeed more sensitive to the choice of hyperparameters than ℓ_2 regularized estimators.

To control for overfitting, it is customary to use different datasets for model training (*i.e.,* computing the regression coefficients) and hyperparameter selection (*i.e.,* choosing the best regularization parameters). A *metric*, *e.g., hold-out loss*, is optimized on a validation dataset (Stone and Ramer, 1965). Alternatively one can rely on a statistical criteria that penalizes complex models such as AIC/BIC (Liu et al., 2011) or SURE (Stein Unbiased Risk Estimator, Stein 1981). In all cases, hyperparameters are tuned to optimize a chosen metric.

The canonical hyperparameter optimization method is *grid-search*. It consists in fitting and selecting the best model over a predefined grid of parameter values. The complexity of grid-search is exponential with the number of hyperparameters, making it only competitive when the number of hyperparameters is small. Other hyperparameter selection strategies include *random search* (Bergstra and Bengio, 2012) and Bayesian optimization (Brochu et al., 2010; Snoek et al., 2012) that aims to learn an approximation of the metric over the parameter space and rely on an exploration policy to find the optimum.

Another line of work for hyperparameter optimization (HO) relies on gradient descent in the hyperparameter space. This strategy has been widely explored for smooth objective functions (Larsen et al., 1996; Bengio, 2000; Larsen et al., 2012). The main challenge for this class of methods is estimating the gradient *w.r.t.* the hyperparameters. Gradient estimation techniques are mostly divided in two categories. *Implicit differentiation* requires the exact

solution of the optimization problem and involves the resolution of a linear system (Bengio, 2000). This can be expensive to compute and lead to numerical instabilities, especially when the system is ill-conditioned (Lorraine et al., 2019). Alternatively, *iterative differentiation* computes the gradient using the iterates of an optimization algorithm. Backward iterative differentiation (Domke, 2012) is computationally efficient when the number of hyperparameters is large. However it is memory consuming since it requires storing all intermediate iterates. In contrast, forward iterative differentiation (Deledalle et al., 2014; Franceschi et al., 2017) does not require storing the iterates but can be computationally expensive with a large number of hyperparameters; see Baydin et al. (2018) for a survey.

This article proposes to investigate the use of these methods to set the regularization hyperparameters in an automatic fashion for Lasso-type problems. To cover the cases of both low and high number of hyperparameters, two estimators are investigated, namely the Lasso and the weighted Lasso which have respectively one or as many parameters as features. Our contributions are as follows:

- We show that forward iterative differentiation of block coordinate descent (BCD), a state-of-the-art solver for Lasso-type problems, converges towards the true gradient. Crucially, we show that this scheme converges linearly once the support is identified and that its limit does **not** depend of the initial starting point.
- These results lead to the proposed algorithm (Algorithm 2) where the computation of the Jacobian is **decoupled** from the computation of the regression coefficients. The later can be done with state-of-the-art convex solvers, and interestingly, it does not require solving a linear system, potentially ill-conditioned.
- We show through an extensive benchmark on simulated and real high dimensional data that the proposed method outperforms state-of-the-art HO methods.

Our work is somewhat similar to Gregor and LeCun (2010); Xin et al. (2016); Borgerding et al. (2017); Liu et al. (2018); Wu et al. (2019), where the *solution* is differentiated w.r.t. optimization parameters instead of the regularization parameter. However the goal is very different as they want to accelerate the optimization algorithm whereas we provide an efficient algorithm to compute the gradient.

Notation The design matrix is $X \in \mathbb{R}^{n \times p}$ (corresponding to n samples and p features) and the observation vector is $y \in \mathbb{R}^n$. The regularization parameter, possibly multivariate, is denoted by $\lambda = (\lambda_1, \dots, \lambda_r)^\top \in \mathbb{R}^r$. We denote $\hat{\beta}^{(\lambda)} \in \mathbb{R}^p$ the regression coefficients associated to λ. We denote $\hat{\mathcal{J}}_{(\lambda)} \triangleq (\nabla_\lambda \hat{\beta}_1^{(\lambda)}, \dots, \nabla_\lambda \hat{\beta}_p^{(\lambda)})^\top \in \mathbb{R}^{p \times r}$ the weak Jacobian (Evans and Gariepy, 1992) of $\hat{\beta}^{(\lambda)}$ *w.r.t.* λ. For a function $\psi : \mathbb{R}^p \times \mathbb{R}^r \to \mathbb{R}$ with weak derivatives of order two, we denote by $\nabla_\beta \psi(\beta, \lambda) \in \mathbb{R}^p$ (resp. $\nabla_\lambda(\beta, \lambda) \in \mathbb{R}^r$) its weak gradient *w.r.t.* the first parameter (resp. the second parameter). The weak Hessian $\nabla^2 \psi(\beta, \lambda)$ is a matrix in $\mathbb{R}^{(p+r)\times(p+r)}$ which has a block structure

$$\nabla^2 \psi(\beta, \lambda) = \begin{pmatrix} \nabla^2_\beta \psi(\beta, \lambda) & \nabla^2_{\beta,\lambda} \psi(\beta, \lambda) \\ \nabla^2_{\lambda,\beta} \psi(\beta, \lambda) & \nabla^2_\lambda \psi(\beta, \lambda) \end{pmatrix} .$$

The support of $\hat{\beta}^{(\lambda)}$ (the indices of non-zero coefficients) is denoted by $\hat{S}^{(\lambda)}$, and $\hat{s}^{(\lambda)}$ represents its cardinality (*i.e.,* the number of non-zero coefficients). The sign vector $\operatorname{sign} \hat{\beta}^{(\lambda)} \in \mathbb{R}^p$ is the vector of component-wise signs (with the convention that $\operatorname{sign}(0) = 0$) of $\hat{\beta}^{(\lambda)}$. Note that to ease the reading, we drop λ in the notation when it is clear from the context and use $\hat{\beta}, \hat{\mathcal{J}}, \hat{S}$ and $\hat{s}$. The Mahalanobis distance of a vector $x \in \mathbb{R}^p$ and a matrix $A \succ 0$ is noted $\|x\|_A \triangleq \sqrt{x^\top A^{-1} x}$.

2. Background

2.1. Problem setting

To favor sparse coefficients, we consider Lasso-type estimators based on non-smooth regularization functions. Such problems consist in finding:

$$\hat{\beta}^{(\lambda)} \in \underset{\beta \in \mathbb{R}^p}{\arg\min}\, \psi(\beta, \lambda) \ . \tag{1}$$

The Lasso (Tibshirani, 1996) is recovered, with the number of hyperparameters set to $r = 1$:

$$\psi(\beta, \lambda) = \frac{1}{2n} \|y - X\beta\|_2^2 + e^\lambda \|\beta\|_1 \ , \tag{2}$$

while the weighted Lasso (wLasso, Zou 2006, introduced to reduce the bias of the Lasso) has $r = p$ hyperparameters and reads:

$$\psi(\beta, \lambda) = \frac{1}{2n} \|y - X\beta\|_2^2 + \sum_{j=1}^p e^{\lambda_j} |\beta_j| \ . \tag{3}$$

Note that we adopt the hyperparameter parametrization of Pedregosa (2016), *i.e.,* we write the regularization parameter as e^λ. This avoids working with a positivity constraint in the optimization process and fixes scaling issues in the line search. It is also coherent with the usual choice of a geometric grid for grid-search (Friedman et al., 2010).

Remark 1. Other formulations could be investigated like Elastic-Net or non-convex formulation, *e.g.,* MCP (Zhang, 2010). Our theory does not cover non-convex cases, though we illustrate that it behaves properly numerically. Handling such non-convex cases is left as a question for future work.

The HO problem can be expressed as a nested *bi-level optimization* problem. For a given differentiable criterion $\mathcal{C} : \mathbb{R}^p \mapsto \mathbb{R}$ (*e.g.,* hold-out loss or SURE), it reads:

$$\underset{\lambda \in \mathbb{R}^r}{\arg\min} \left\{ \mathcal{L}(\lambda) \triangleq \mathcal{C}\left(\hat{\beta}^{(\lambda)}\right) \right\} \quad s.t.\ \hat{\beta}^{(\lambda)} \in \underset{\beta \in \mathbb{R}^p}{\arg\min}\, \psi(\beta, \lambda) \ . \tag{4}$$

Note that SURE itself is not necessarily weakly differentiable *w.r.t.* $\hat{\beta}^{(\lambda)}$. However a weakly differentiable approximation can be constructed (Ramani et al., 2008; Deledalle et al., 2014). Under the hypothesis that Problem (1) has a unique solution for every $\lambda \in \mathbb{R}^r$, the function $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is weakly differentiable (Vaiter et al., 2013). Using the chain rule, the gradient of $\mathcal{L}$ *w.r.t.* λ then writes:

$$\nabla_\lambda \mathcal{L}(\lambda) = \hat{\mathcal{J}}_{(\lambda)}^\top \nabla \mathcal{C}\left(\hat{\beta}^{(\lambda)}\right) \ . \tag{5}$$

Computing the weak Jacobian $\hat{\mathcal{J}}_{(\lambda)}$ of the inner problem is the main challenge, as once the *hypergradient* $\nabla_\lambda \mathcal{L}(\lambda)$ has been computed, one can use usual gradient descent, $\lambda^{(t+1)} = \lambda^{(t)} - \rho \nabla_\lambda \mathcal{L}(\lambda^{(t)})$, for a step size $\rho > 0$. Note however that $\mathcal{L}$ is usually non-convex and convergence towards a global minimum is not guaranteed. In this work, we propose an efficient algorithm to compute $\hat{\mathcal{J}}_{(\lambda)}$ for Lasso-type problems, relying on improved forward differentiation.

2.2. Implicit differentiation (smooth case)

Implicit differentiation, which can be traced back to Larsen et al. (1996), is based on the knowledge of $\hat{\beta}$ and requires solving a $p \times p$ linear system (Bengio, 2000, Sec. 4). Since then, it has been extensively applied in various contexts. Chapelle et al. (2002); Seeger (2008) used implicit differentiation to select hyperparameters of kernel-based models. Kunisch and Pock (2013) applied it to image restoration. Pedregosa (2016) showed that each inner optimization problem could be solved only approximately, leveraging noisy gradients. Related to our work, Foo et al. (2008) applied implicit differentiation on a "weighted" Ridge-type estimator (*i.e.,* a Ridge penalty with one λ_j per feature).

Yet, all the aforementioned methods have a common drawback : they are limited to the smooth setting, since they rely on optimality conditions for smooth optimization. They proceed as follows: if $\beta \mapsto \psi(\beta, \lambda)$ is a smooth convex function (for any fixed λ) in Problem (1), then for all λ, the solution $\hat{\beta}^{(\lambda)}$ satisfies the following fixed point equation:

$$\nabla_\beta \psi\left(\hat{\beta}^{(\lambda)}, \lambda\right) = 0 \ . \tag{6}$$

Then, this equation can be differentiated *w.r.t.* λ:

$$\nabla^2_{\beta,\lambda} \psi(\hat{\beta}^{(\lambda)}, \lambda) + \hat{\mathcal{J}}_{(\lambda)}^\top \nabla^2_\beta \psi(\hat{\beta}^{(\lambda)}, \lambda) = 0 \ . \tag{7}$$

Assuming that $\nabla^2_\beta \psi(\hat{\beta}^{(\lambda)}, \lambda)$ is invertible this leads to a closed form solution for the weak Jacobian $\hat{\mathcal{J}}_{(\lambda)}$:

$$\hat{\mathcal{J}}_{(\lambda)}^\top = -\nabla^2_{\beta,\lambda} \psi\left(\hat{\beta}^{(\lambda)}, \lambda\right) \underbrace{\left(\nabla^2_\beta \psi(\beta^{(\lambda)}, \lambda)\right)}_{p \times p}^{-1} \ , \tag{8}$$

which in practice is computed by solving a linear system. Unfortunately this approach cannot be generalized for non-smooth problems since Equation (6) no longer holds.

2.3. Implicit differentiation (non-smooth case)

Related to our work Mairal et al. (2012) used implicit differentiation with respect to the dictionary ($X \in \mathbb{R}^{n \times p}$) on Elastic-Net models to perform dictionary learning. Regarding Lasso problems, the literature is quite scarce, see (Dossal et al., 2013; Zou et al., 2007) and (Vaiter et al., 2013; Tibshirani and Taylor, 2011) for a more generic setting encompassing weighted Lasso. General methods for gradient estimation of non-smooth optimization schemes exist (Vaiter et al., 2017) but are not practical since they depend on a possibly ill-posed linear system to invert. Amos and Kolter (2017) have applied implicit differentiation on estimators based on quadratic objective function with linear constraints, whereas Niculae and Blondel (2017) have used implicit differentiation on a smooth objective function with simplex constraints. However none of these approaches leverages the sparsity of Lasso-type estimators.

3. Hypergradients for Lasso-type problems

To tackle hyperparameter optimization of non-smooth Lasso-type problems, we propose in this section an efficient algorithm for hypergradient estimation. Our algorithm relies on implicit differentiation, thus enjoying low-memory cost, yet does not require to naively solve a (potentially ill-conditioned) linear system of equations. In the sequel, we assume access to a (weighted) Lasso solver, such as ISTA (Daubechies et al., 2004) or Block Coordinate Descent (BCD, Tseng and Yun 2009, see also Algorithm 5).

3.1. Implicit differentiation

Our starting point is the key observation that Lasso-type solvers induce a fixed point iteration that we can leverage to compute a Jacobian. Indeed, proximal BCD algorithms (Tseng and Yun, 2009), consist in a local gradient step composed with a soft-thresholding step (ST), *e.g.,* for the Lasso, for $j \in 1, \dots, p$:

$$\beta_j \leftarrow \mathrm{ST}\left(\beta_j - \frac{X_{:,j}^\top (X\beta - y)}{\|X_{:,j}\|^2}, \frac{ne^\lambda}{\|X_{:,j}\|^2}\right) \tag{9}$$

where $\mathrm{ST}(t, \tau) = \mathrm{sign}(t) \cdot (|t| - \tau)_+$ for any $t \in \mathbb{R}$ and $\tau \geq 0$ (extended for vectors component-wise). The solution of

the optimization problem satisfies, for any $\alpha > 0$, the fixed-point equation (Combettes and Wajs, 2005, Prop. 3.1), for $j \in 1, \dots, p$:

$$\hat{\beta}_j^{(\lambda)} = \mathrm{ST}\left(\hat{\beta}_j^{(\lambda)} - \frac{1}{\alpha} X_{j,:}^\top (X\hat{\beta}^{(\lambda)} - y), \frac{ne^\lambda}{\alpha}\right) \quad . \quad (10)$$

The former can be differentiated *w.r.t.* λ, see Lemma A.1 in Appendix, leading to a closed form solution for the Jacobian $\mathcal{J}_{(\lambda)}$ of the Lasso and the weighted Lasso.

Proposition 1 (Adapting Vaiter et al. 2013, Thm. 1)**.** Let $\hat{S}$ be the support of the vector $\hat{\beta}^{(\lambda)}$. Suppose that $X_{\hat{S}}^\top X_{\hat{S}} \succ 0$, then a weak Jacobian $\hat{\mathcal{J}} = \hat{\mathcal{J}}_{(\lambda)}$ of the Lasso writes:

$$\hat{\mathcal{J}}_{\hat{S}} = -ne^\lambda \left(X_{\hat{S}}^\top X_{\hat{S}}\right)^{-1} \operatorname{sign} \hat{\beta}_{\hat{S}}, \quad (11)$$

$$\hat{\mathcal{J}}_{\hat{S}^c} = 0 \ , \quad (12)$$

and for the weighted Lasso:

$$\hat{\mathcal{J}}_{\hat{S},\hat{S}} = -\left(X_{\hat{S}}^\top X_{\hat{S}}\right)^{-1} \mathrm{diag}\left(ne^{\lambda_{\hat{S}}} \odot \operatorname{sign} \hat{\beta}_{\hat{S}}\right) \quad (13)$$

$$\hat{\mathcal{J}}_{j_1,j_2} = 0 \quad \text{if } j_1 \notin \hat{S} \text{ or if } j_2 \notin \hat{S} \ . \quad (14)$$

The proof of Proposition 1 can be found in Appendix A.1. Note that the positivity condition in Proposition 1 is satisfied if the (weighted) Lasso has a unique solution. Moreover, even for multiple solutions cases, there exists at least one satisfying the positivity condition (Vaiter et al., 2013).

Proposition 1 shows that the Jacobian of the weighted Lasso $\hat{\mathcal{J}}_{(\lambda)} \in \mathbb{R}^{p \times p}$ is row and column sparse. This is key for algorithmic efficiency. Indeed, *a priori*, one has to store a possibly dense $p \times p$ matrix, which is prohibitive when p is large. Proposition 1 leads to a simple algorithm (see Algorithm 1) to compute the Jacobian in a *cheap* way, as it *only* requires storing and inverting an $\hat{s} \times \hat{s}$ matrix. Even if the linear system to solve is of size $\hat{s} \times \hat{s}$, instead of $p \times p$ for smooth objective function, the system to invert can be ill-conditioned, especially when a large support size $\hat{s}$ is encountered. This leads to numerical instabilities and slows down the resolution (see an illustration in Figure 2). Forward (Algorithm 3 in Appendix) and backward (Algorithm 4 in Appendix) iterative differentiation, which do not require solving linear systems, can overcome these issues.

3.2. Link with iterative differentiation

Iterative differentiation in the field of hyperparameter setting can be traced back to Domke (2012) who derived a backward differentiation algorithm for gradient descent, heavy ball and L-BFGS algorithms applied to smooth loss functions. Agrawal et al. (2019) generalized it to a specific subset of convex programs. Maclaurin et al. (2015) derived a backward differentiation for stochastic gradient descent. On the other hand Deledalle et al. (2014) used forward differentiation of (accelerated) proximal gradient descent for hyperparameter optimization with non-smooth penalties. Franceschi et al. (2017) proposed a benchmark of forward mode versus backward mode, varying the number of hyperparameters to learn. Frecon et al. (2018) cast the problem of inferring the groups in a group-Lasso model as a bi-level optimization problem and solved it using backward differentiation.

Algorithm 1 IMPLICIT DIFFERENTIATION

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}, n_{\text{iter}} \in \mathbb{N}$

`// jointly compute coef. and Jacobian`

if *Lasso* **then**

  Get $\hat{\beta} = Lasso(X, y, \lambda, n_{\text{iter}})$ and its support $\hat{S}$.

  $\hat{\mathcal{J}} = 0_p$

  $\hat{\mathcal{J}}_{\hat{S}} = -ne^\lambda (X_{\hat{S}}^\top X_{\hat{S}})^{-1} \operatorname{sign} \hat{\beta}_{\hat{S}}$

if *wLasso* **then**

  Get $\hat{\beta} = wLasso(X, y, \lambda, n_{\text{iter}})$ and its support $\hat{S}$.

  $\hat{\mathcal{J}} = 0_{p \times p}$

  $\hat{\mathcal{J}}_{\hat{S},\hat{S}} = -(X_{\hat{S}}^\top X_{\hat{S}})^{-1} \operatorname{diag}(ne^{\lambda_{\hat{S}}} \odot \operatorname{sign} \hat{\beta}_{\hat{S}})$

return $\hat{\beta}, \hat{\mathcal{J}}$

Forward differentiation consists in differentiating each step of the algorithm (*w.r.t.* λ in our case). For the Lasso solved with BCD it amounts differentiating Equation (9), and leads to the following recursive equation for the Jacobian, for $j \in 1, \dots p$, with $z_j = \beta_j - X_{:,j}^\top (X\beta - y) / \|X_{:,j}\|^2$:

$$\mathcal{J}_j \leftarrow \partial_1 \mathrm{ST}\left(z_j, \tfrac{ne^\lambda}{\|X_{:,j}\|^2}\right)\left(\mathcal{J}_j - \tfrac{1}{\|X_{:,j}\|^2} X_{:,j}^\top X \mathcal{J}\right) + \partial_2 \mathrm{ST}\left(z_j, \tfrac{ne^\lambda}{\|X_{:,j}\|^2}\right) \tfrac{ne^\lambda}{\|X_{:,j}\|^2} \quad , \quad (15)$$

see Algorithm 3 (in Appendix) for full details. Our proposed algorithm uses the fact that after a finite number of epochs $\partial_1 \mathrm{ST}(z_j, ne^\lambda / \|X_{:,j}\|^2)$ and $\partial_2 \mathrm{ST}(z_j, ne^\lambda / \|X_{:,j}\|^2)$ are **constant** (they no no longer depends on the current β). Indeed, the sign of $\hat{\beta}$ is identified after a finite number of iterations thus the partial derivatives are constant. It is then possible to **decouple** the computation of the Jacobian by only solving Problem (1) in a first step and then apply the forward differentiation recursion steps, see Algorithm 2. This can be seen as the forward counterpart in a non-smooth case of the recent paper Lorraine et al. (2019). An additional benefit of such updates is that they can be restricted to the (current) support, which leads to faster Jacobian computation.

We now show that the Jacobian computed using forward differentiation and our method, Algorithm 2, converges toward the true Jacobian.

Proposition 2. Assuming the Lasso solution (Problem (2)) (or weighted Lasso Problem (3)) is unique, then Algorithms 2 and 3 converge toward the Jacobian $\hat{\mathcal{J}}$ defined in Proposition 1. Algorithm 3 computes the Jacobian along with the regression coefficients, once the support has been identified, the Jacobian converges linearly. Algorithm 2 computes first the coefficients $\hat{\beta}$ and then the Jacobian $\hat{\mathcal{J}}$, provided that the support has been identified in the first step, the convergence is linear in the second, with the same rate as Algorithm 3:

$$\|\mathcal{J}_{\hat{S}}^{(k+1)} - \hat{\mathcal{J}}\|_{(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{-1}} \leq C^k \|\mathcal{J}_{\hat{S}}^{(k)} - \hat{\mathcal{J}}\|_{(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{-1}}$$

where $C = \|A^{(j_{\hat{s}})} \dots A^{(j_1)}\|_2 < 1$, $j_1, \dots, j_{\hat{s}}$ are the indices of the support of $\hat{\beta}$ in increasing order and

$$A^{(j_s)} = \mathrm{Id}_{\hat{s}} - \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)_{:,j_s}^{1/2}}{\|X_{:,j_s}\|} \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)_{j_s,:}^{1/2}}{\|X_{:,j_s}\|} \in \mathbb{R}^{\hat{s}\times\hat{s}}.$$

□

Proof of Proposition 2 can be found in Appendix A.2 and A.3.

Remark 3. *Uniqueness.* As proved in Tibshirani (2013, Lem. 3 and 4) the set of (pathological) lambdas where the Lasso solution is not unique is typically empty. Moreover if the Lasso solution is not unique, there could be a non-continuous solution path $\lambda \mapsto \hat{\beta}^{(\lambda)}$, leaving only non-gradient based methods available. Even if Proposition 2 does not provide theoretical guarantees in such a pathological setting, one can still apply Algorithms 2 and 3, see Appendix E.1 for experiments in this settings.

Remark 4. *Rate for the backward differentiation.* The backward and forward differentiation compute the same quantity: $\nabla_\lambda \mathcal{L}(\lambda)$, but the backward differentiation directly computes the product given in Equation (5) leading to the gradient of $\mathcal{L}(\lambda)$. Proposition 2 provides rates for the convergence of the Jacobian $\mathcal{J}$ which leads to rates for the gradient *i.e.,* for the backward algorithm as well.

As an illustration, Figure 1 shows the times of computation of a single gradient $\nabla_\lambda \mathcal{L}(\lambda)$ and the distance to "optimum" of this gradient as a function of the number of iterations in the inner optimization problem for the forward iterative differentiation (Algorithm 3), the backward iterative differentiation (Algorithm 4), and the proposed algorithm (Algorithm 2). The backward iterative differentiation is several order of magnitude slower than the forward and our implicit forward method. Moreover, once the support has been identified (after 20 iterations) the proposed implicit forward method converges faster than other methods. Note also that in Propositions 1 and 2 the Jacobian for the

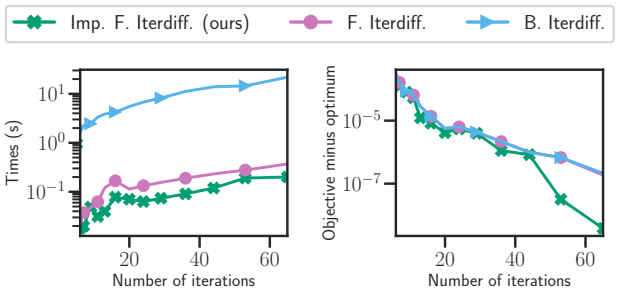

Figure 1. **Time to compute a single gradient** (Synthetic data, Lasso, $n, p = 1000, 2000$). Influence on the number of iterations of BCD (in the inner optimization problem of Problem (4)) on the computation time (left) and the distance to "optimum" of the gradient $\nabla_\lambda \mathcal{L}(\lambda)$(right) for the Lasso estimator. The "optimum" is here the gradient given by implicit differentiation (Algorithm 1).

Lasso only depends on the *support* (*i.e.,* the indices of the non-zero coefficients) of the regression coefficients $\hat{\beta}^{(\lambda)}$. In other words, once the support of $\hat{\beta}^{(\lambda)}$ is correctly identified, even if the value of the non-zeros coefficients are not correctly estimated, the Jacobian is exact, see Sun et al. (2019) for support identification guarantees.

4. Experiments

Our Python code is released as an open source package: https://github.com/QB3/sparse-ho. All the experiments are written in Python using Numba (Lam et al., 2015) for the critical parts such as the BCD loop. We compare our gradient computation technique against other competitors (see the competitors section) on the HO problem (Problem (4)).

Solving the inner optimization problem. Note that our proposed method, implicit forward differentiation, has the appealing property that it can be used with any solver. For instance for the Lasso one can combine the proposed algorithm with state of the art solver such as Massias et al. (2018) which would be tedious to combine with iterative differentiation methods. However for the comparison to be fair, for all methods we have used the same vanilla BCD algorithm (recalled in Algorithm 5). We stop the Lasso-types solver when $\frac{f(\beta^{(k+1)}) - f(\beta^{(k)})}{f(0)} < \epsilon^{\text{tol}}$, where f is the cost function of the Lasso or wLasso and ϵ^{tol} a given tolerance. The tolerance is fixed at $\epsilon^{\text{tol}} = 10^{-5}$ for all methods throughout the different benchmarks.

Line search. For each hypergradient-based method, the gradient step is combined with a line-search strategy following the work of Pedregosa (2016)[1].

Initialization. Since the function to optimize $\mathcal{L}$ is not con-

[1] see https://github.com/fabianp/hoag for details

Table 1. Summary of cost in time and space for each method

Mode	Computed quantity	Space (Lasso)	Time (Lasso)	Space (wLasso)	Time (wLasso)
F. Iterdiff.	$\mathcal{J}$	$\mathcal{O}(p)$	$\mathcal{O}(2npn_{\text{iter}})$	$\mathcal{O}(p^2)$	$\mathcal{O}(np^2n_{\text{iter}})$
B. Iterdiff.	$\mathcal{J}^\top v$	$\mathcal{O}(2pn_{\text{iter}})$	$\mathcal{O}(npn_{\text{iter}} + np^2n_{\text{iter}})$	$\mathcal{O}(p^2n_{\text{iter}})$	$\mathcal{O}(npn_{\text{iter}} + np^2n_{\text{iter}})$
Implicit	$\mathcal{J}^\top v$	$\mathcal{O}(p)$	$\mathcal{O}(npn_{\text{iter}} + \hat{s}^3)$	$\mathcal{O}(p + \hat{s}^2)$	$\mathcal{O}(npn_{\text{iter}} + \hat{s}^3)$
Imp. F. Iterdiff.	$\mathcal{J}$	$\mathcal{O}(p)$	$\mathcal{O}(npn_{\text{iter}} + n\hat{s}n_{\text{iter\_jac}})$	$\mathcal{O}(p + \hat{s}^2)$	$\mathcal{O}(npn_{\text{iter}} + n\hat{s}^2n_{\text{it\_jac}})$

Algorithm 2 IMP. F. ITERDIFF. (proposed)

input : $X \in \mathbb{R}^{n\times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}, n_{\text{iter}}, n_{\text{iter\_jac}} \in \mathbb{N}$
init : $\mathcal{J} = 0$
// sequentially compute coef. & Jacobian
if *Lasso* **then**
  Get $\hat{\beta} = Lasso(X, y, \lambda, n_{\text{iter}})$ and its support $\hat{S}$.
  $dr = -X_{:,\hat{S}}\mathcal{J}_{\hat{S}}$ // trick for cheap updates
if *wLasso* **then**
  Get $\hat{\beta} = wLasso(X, y, \lambda, n_{\text{iter}})$ and its support $\hat{S}$.
  $dr = -X_{:,\hat{S}}\mathcal{J}_{\hat{S},\hat{S}}$
for $k = 0, \dots, n_{iter\_jac} - 1$ **do**
  for $j \in \hat{S}$ **do**
    if *Lasso* **then**
      $\mathcal{J}_{old} = \mathcal{J}_j$ // trick for cheap update
      // diff. Equation (9) w.r.t. λ
      $\mathcal{J}_j \mathrel{+}= \frac{X_{:,j}^\top dr}{\|X_{:,j}\|^2} - \frac{ne^\lambda}{\|X_{:,j}\|^2}\operatorname{sign}\hat{\beta}_j$ // $\mathcal{O}(n)$
      $dr \mathrel{-}= X_{:,j}(\mathcal{J}_{j,:} - \mathcal{J}_{old})$ // $\mathcal{O}(n)$
    if *wLasso* **then**
      $\mathcal{J}_{old} = \mathcal{J}_{j,:}$ // trick for cheap update
      // diff. Equation (9) w.r.t. λ
      $\mathcal{J}_{j,\hat{S}} \mathrel{+}= \frac{1}{\|X_{:,j}\|^2}X_{:,j}^\top dr$ // $\mathcal{O}(n \times \hat{s})$
      $\mathcal{J}_{j,j} \mathrel{-}= \frac{ne^{\lambda_j}}{\|X_{:,j}\|^2}\operatorname{sign}\hat{\beta}_j$ // $\mathcal{O}(1)$
      $dr \mathrel{-}= X_{:,j} \otimes (\mathcal{J}_{j,:} - \mathcal{J}_{old})$ // $\mathcal{O}(n \times \hat{s})$
return $\hat{\beta}, \mathcal{J}$

vex, initialization plays a crucial role in the final solution as well as the convergence of the algorithm. For instance, initializing $\lambda = \lambda_{\text{init}}$ in a flat zone of $\mathcal{L}(\lambda)$ could lead to slow convergence. In the numerical experiments, the Lasso is initialized with $\lambda_{\text{init}} = \lambda_{\max} - \log(10)$, where $\lambda_{\max}$ is the smallest λ such that 0 is a solution of Problem (2).

Competitors. In this section we compare the empirical performance of implicit forward differentiation algorithm to different competitors. Competitors are divided in two categories. Firstly, the ones relying on hyperparameter gradient:

- **Imp. F. Iterdiff.**: implicit forward differentiation (proposed) described in Algorithm 2.
- **Implicit**: implicit differentiation, which requires solving a $\hat{s} \times \hat{s}$ linear system as described in Algorithm 1.
- **F. Iterdiff.**: forward differentiation (Deledalle et al., 2014; Franceschi et al., 2017) which jointly computes the regression coefficients $\hat{\beta}$ as well as the Jacobian $\hat{\mathcal{J}}$ as shown in Algorithm 3.

Secondly, the ones not based on hyperparameter gradient:

- **Grid-search**: as recommended by Friedman et al. (2010), we use 100 values on a uniformly-spaced grid from $\lambda_{\max}$ to $\lambda_{\max} - 4\log(10)$.
- **Random-search**: we sample uniformly at random 100 values taken on the same interval as for the Grid-search $[\lambda_{\max} - 4\log(10); \lambda_{\max}]$, as suggested by Bergstra et al. (2013).
- **Lattice Hyp.**: lattice hypercube sampling (Bousquet et al., 2017), combines the idea of grid-search and random-search. We used the sampling scheme of Bouhlel et al. (2019) and their code [2] to sample the points to evaluate the function on.
- **Bayesian**: sequential model based optimization (SMBO) using a Gaussian process to model the objective function. We used the implementation of Bergstra et al. (2013).[3] The constraints space for the hyperparameter search was set in $[\lambda_{\max} - 4\log(10); \lambda_{\max}]$, and the expected improvement (EI) was used as aquisition function.

The cost and the quantity computed by each algorithm can be found in Table 1. The backward differentiation (Domke, 2012) is not included in the benchmark in Figure 2 since it was several orders of magnitude slower than the other techniques (see Figure 1). This is due to the high cost of the BCD algorithm in backward mode, see Table 1.

4.1. Application to held-out loss

When using the held-out loss, each dataset (X, y) is split in 3 equal parts: the training set $(X^{\text{train}}, y^{\text{train}})$, the validation set $(X^{\text{val}}, y^{\text{val}})$ and the test set $(X^{\text{test}}, y^{\text{test}})$.

[2] https://github.com/SMTorg/smt
[3] https://github.com/hyperopt/hyperopt

(*Lasso, held-out criterion*). For the Lasso and the held-out loss, the bilevel optimization Problem (4) reads:

$$\arg\min_{\lambda \in \mathbb{R}} \|y^{\text{val}} - X^{\text{val}} \hat{\beta}^{(\lambda)}\|^2 \quad (16)$$
$$s.t.\ \hat{\beta}^{(\lambda)} \in \arg\min_{\beta \in \mathbb{R}^p} \tfrac{1}{2n} \|y^{\text{train}} - X^{\text{train}} \beta\|_2^2 + e^{\lambda} \|\beta\|_1 \ .$$

Figure 2 (top) shows on 3 datasets (see Appendix D for dataset details) the distance to the "optimum" of $\|y^{\text{val}} - X^{\text{val}} \hat{\beta}^{(\lambda)}\|^2$ as a function of time. Here the goal is to find λ solution of Problem (16). The "optimum" is chosen as the minimum of $\|y^{\text{val}} - X^{\text{val}} \hat{\beta}^{(\lambda)}\|^2$ among all the methods. Figure 2 (bottom) shows the loss $\|y^{\text{test}} - X^{\text{test}} \hat{\beta}^{(\lambda)}\|^2$ on the test set (independent from the training set and the validation set). This illustrates how well the estimator generalizes. Firstly, it can be seen that on all datasets the proposed implicit forward differentiation outperforms forward differentiation which illustrates Proposition 2 and corroborates the cost of each algorithm in Table 1. Secondly, it can be seen that on the *20news* dataset (Figure 2, top) the implicit differentiation (Algorithm 1) convergence is slower than implicit forward differentiation, forward differentiation, and even slower than the grid-search. In this case, this is due to the very slow convergence of the conjugate gradient algorithm (Nocedal and Wright, 2006) when solving the ill-conditioned linear system in Algorithm 1.

(*MCP, held-out criterion*). We also applied our algorithm on an estimator based on a non-convex penalty: the MCP (Zhang, 2010) with 2 hyperparameters. Since the penalty is non-convex the estimator may not be continuous *w.r.t.* hyperparameters and the theory developed above does not hold. However experimentally implicit forward differentiation outperforms forward differentiation for the HO, see Appendix C for full details.

4.2. Application to another criterion: SURE

Evaluating models on held-out data makes sense if the design is formed from random samples as it is often considered in supervised learning. However, this assumption does not hold for certain kinds of applications in signal or image processing. For these applications, the held-out loss cannot be used as the criterion for optimizing the hyperparameters of a given model. In this case, one may use a proxy of the prediction risk, like the Stein Unbiased Risk Estimation (SURE, Stein (1981)). The SURE is an unbiased estimator of the prediction risk under weak differentiable conditions. The drawback of this criterion is that it requires the knowledge of the variance of the noise. The SURE is defined as follows: $\text{SURE}(\lambda) = \|y - X\hat{\beta}^{(\lambda)}\|^2 - n\sigma^2 + 2\sigma^2 \text{dof}(\hat{\beta}^{(\lambda)})$, where the degrees of freedom (dof Efron 1986) is defined as $\text{dof}(\hat{\beta}^{(\lambda)}) = \sum_{i=1}^{n} \text{cov}(y_i, (X\hat{\beta}^{(\lambda)})_i)/\sigma^2$. The dof can be seen a measure of the complexity of the model, for instance for the Lasso $\text{dof}(\hat{\beta}^{(\lambda)}) = \hat{s}$, see Zou et al. (2007). The SURE can thus be seen as a criterion trading data-fidelity against model complexity. However, the dof is not differentiable (not even continuous in the Lasso case), yet it is possible to construct a weakly differentiable approximation of it based on Finite Differences Monte-Carlo (see Deledalle et al. 2014 for full details), with $\epsilon > 0$ and $\delta \sim \mathcal{N}(0, \text{Id}_n)$:

$$\text{dof}_{\text{FDMC}}(y, \lambda, \delta, \epsilon) = \tfrac{1}{\epsilon} \langle X\hat{\beta}^{(\lambda)}(y + \epsilon\delta) - X\hat{\beta}^{(\lambda)}(y), \delta \rangle \ .$$

We use this smooth approximation in the bi-level optimization problem to find the best hyperparameter. The bi-level optimization problem then reads:

$$\arg\min_{\lambda \in \mathbb{R}} \|y - X\hat{\beta}^{(\lambda)}\|^2 + 2\sigma^2 \text{dof}_{\text{FDMC}}(y, \lambda, \delta, \epsilon) \quad (17)$$
$$s.t.\ \hat{\beta}^{(\lambda)}(y) \in \arg\min_{\beta \in \mathbb{R}^p} \tfrac{1}{2n} \|y - X\beta\|_2^2 + e^{\lambda} \|\beta\|_1$$
$$\hat{\beta}^{(\lambda)}(y + \epsilon\delta) \in \arg\min_{\beta \in \mathbb{R}^p} \tfrac{1}{2n} \|y + \epsilon\delta - X\beta\|_2^2 + e^{\lambda} \|\beta\|_1$$

Note that solving this problem requires the computation of two (instead of one for the held-out loss) Jacobians *w.r.t.* λ of the solution $\hat{\beta}^{(\lambda)}$ at the points y and $y + \epsilon\delta$.

(*Lasso, SURE criterion*). To investigate the estimation performance of the implicit forward differentiation in comparison to the competitors described above, we used as metric the (normalized) Mean Squared Error (MSE) defined as $\text{MSE} \triangleq \|\hat{\beta} - \beta^*\|^2 / \|\beta^*\|^2$. The entries of the design matrix $X \in \mathbb{R}^{n \times p}$ are i.i.d. random Gaussian variables $\mathcal{N}(0, 1)$. The number of rows is fixed to $n = 100$. Then, we generated β^* with 5 non-zero coefficients equals to 1. The vector y was computed by adding to $X\beta^*$ additive Gaussian noise controlled by the Signal-to-Noise Ratio: $\text{SNR} \triangleq \|X\beta^*\| / \|y - X\beta^*\|$ (here $\text{SNR} = 3$). Following Deledalle et al. (2014), we set $\epsilon = 2\sigma / n^{0.3}$. We varied the number of features p between 200 and 10,000 on a linear grid of size 10. For a fixed number of features, we performed 50 repetitions and each point of the curves represents the mean of these repetitions. Comparing efficiency in time between methods is difficult since they are not directly comparable. Indeed, grid-search and random-search discretize the HO space whereas others methods work in the continuous space which is already an advantage. However, to be able to compare the hypergradient methods and possibly compare them to the others, we computed the total amount of time for a method to return its optimal value of λ. In order to have a *fair* comparison, we compared 50 evaluations of the line-search for each hypergradient methods, 50 evaluations of the Bayesian methods and finally 50 evaluations on fixed or random grid. We are aware that the cost of each of these evaluations is not the same but it allows to see that our method stays competitive in time with optimizing one parameter. Moreover we will also see that our method scales better with a large number of hyperparameters to optimize.

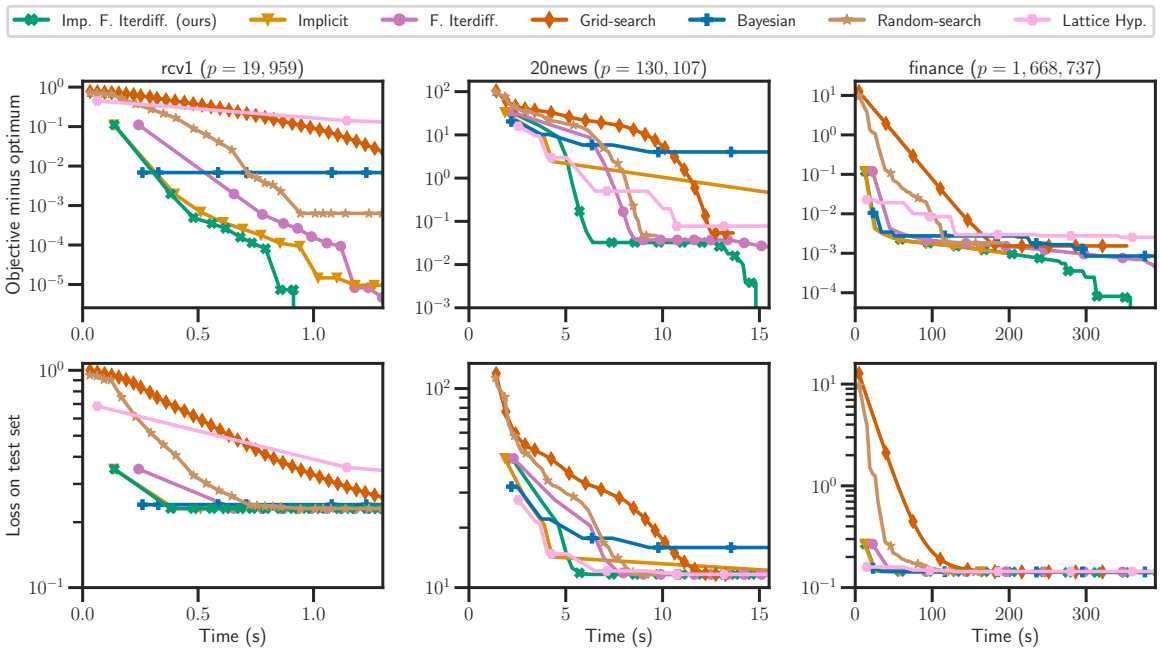

Figure 2. **Computation time for the HO of the Lasso on real data.** Distance to "optimum" (top) and performance (bottom) on the test set for the Lasso for 3 different datasets: *rcv1*, *20news* and *finance*.

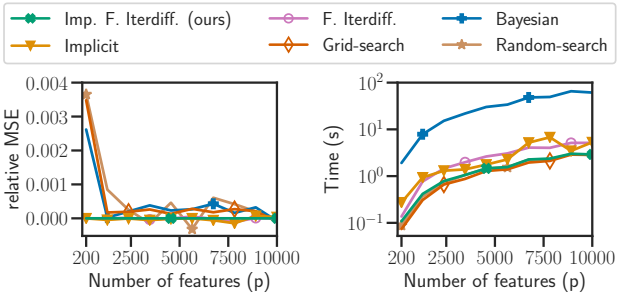

Figure 3. **Lasso: estimation performance.** Estimation relative Mean Squared Error (left) and running time (right) as a function of the number of features for the Lasso model.

Figure 3 shows the influence of the number of features on the relative MSE (ie. MSE of a method minus the MSE of our implicit forward method) and the computation time. First, MSE of all gradient based methods is lower than the other methods which means that $\hat{\beta}^{(\lambda)}$ leads to a better estimation when λ is chosen via the gradient based methods. This illustrates that continuous optimization for hyperparameter selection leads to better estimation performance than discrete or Bayesian optimization. Yet, the running time of our proposed method is the lowest of all hypergradient-based strategies and competes with the grid-search and the random-search.

(*Weighted Lasso vs Lasso, SURE criterion*). As our method leverages the sparsity of the solution, it can be used for HO with a large number of hyperparameters, contrary to classical forward differentiation. The weighted Lasso (wLasso, Zou 2006) has p hyperparameters and was introduced to reduce the bias of the Lasso. However setting the p hyperparameters is impossible with grid-search.

Figure 4 shows the estimation MSE and the running time of the different methods to obtain the hyperparameter values as a function of the number of features used to simulate the data. The simulation setting is here the same as for the Lasso problems investigated in Figure 3 ($n = 100$, SNR $= 3$). We compared the classical Lasso estimator and the weighted Lasso estimator where the regularization hyperparameter was chosen using implicit forward differentiation and the forward iterative differentiation as described in Algorithm 3. Problem (4) is not convex for the weighted Lasso and a descent algorithm like ours can be trapped in local minima, crucially depending on the starting point λ_{init}. To alleviate this problem, we introduced a regularized version of Problem (4):

$$\underset{\lambda \in \mathbb{R}}{\arg\min} \quad \mathcal{C}\left(\hat{\beta}^{(\lambda)}\right) + \gamma \sum_{j}^{p} \lambda_j^2$$
$$s.t.\ \hat{\beta}^{(\lambda)} \in \underset{\beta \in \mathbb{R}^p}{\arg\min} \triangleq \psi(\beta, \lambda) \ . \qquad (18)$$

The solution obtained by solving Equation (18) is then used as the initialization $\lambda^{(0)}$ for our algorithm. In this experiment the regularization term is constant $\gamma =$

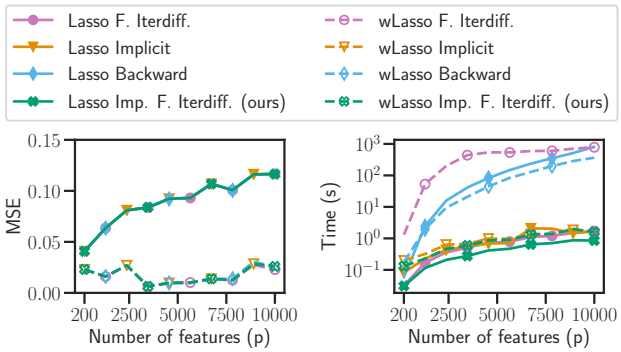

Figure 4. **Lasso vs wLasso.** Estimation Mean Squared Error (left) and running (right) of competitors as a function of the number of features for the weighted Lasso and Lasso models.

$C(\beta^{(\lambda_{\max})})/10$. We see in Figure 4 that the weighted Lasso gives a lower MSE than the Lasso and allows for a better recovery of β^*. This experiment shows that the amount of time needed to obtain the vector of hyperparameters of the weighted Lasso via our algorithm is in the same range as for obtaining the unique hyperparameter of the Lasso problem. It also shows that our proposed method is much faster than the *naive* way of computing the Jacobian using forward or backward iterative differentiation. The implicit differentiation method stays competitive for the wLasso due to the small support of the solution and hence a small matrix to inverse. A maximum running time threshold was used for this experiment checking the running time at each line-search iteration, explaining why the forward differentiation and backward differentiation of the wLasso does not explode in time on Figure 4.

Conclusion

In this work we studied the performance of several methods to select hyperparameters of Lasso-type estimators showing results for the Lasso and the weighted Lasso, which have respectively one or p hyperparameters. We exploited the sparsity of the solutions and the specific structure of the iterates of forward differentiation, leading to our implicit forward differentiation algorithm that computes efficiently the full Jacobian of these estimators *w.r.t.* the hyperparameters. This allowed us to select them through a standard gradient descent and have an approach that scales to a high number of hyperparameters. Importantly, contrary to a classical implicit differentiation approach, the proposed algorithm does not require solving a linear system. Finally, thanks to its two steps nature, it is possible to leverage in the first step the availability of state-of-the-art Lasso solvers that make use of techniques such as active sets or screening rules. Such algorithms, that involve calls to inner solvers run on subsets of features, are discontinuous *w.r.t.* hyperparameters which would significantly challenge a single step approach based on automatic differentiation.

Acknowledgments This work was funded by ERC Starting Grant SLAB ERC-StG-676943 and ANR GraVa ANR-18-CE40-0005.

References

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In *Advances in neural information processing systems*, pages 9558–9570, 2019.

B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, volume 70, pages 136–145, 2017.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18(153):1–43, 2018.

Y. Bengio. Gradient-based optimization of hyperparameters. *Neural computation*, 12(8):1889–1900, 2000.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 2012.

J. Bergstra, D. Yamins, and D. D. Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, pages 13–20, 2013.

M. Borgerding, P. Schniter, and S. Rangan. Amp-inspired deep networks for sparse linear inverse problems. *IEEE Transactions on Signal Processing*, 65(16):4293–4308, 2017.

M. A. Bouhlel, J. T. Hwang, N. Bartoli, R. Lafage, J. Morlier, and J. R. R. A. Martins. A python surrogate modeling framework with derivatives. *Advances in Engineering Software*, page 102662, 2019. ISSN 0965-9978. doi: https://doi.org/10.1016/j.advengsoft.2019.03.005.

O. Bousquet, S. Gelly, K. Kurach, O. Teytaud, and D. Vincent. Critical hyper-parameters: No random, no cry. *arXiv preprint arXiv:1706.03200*, 2017.

P. Breheny and J. Huang. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Ann. Appl. Stat.*, 5(1):232, 2011.

E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine learning*, 46(1-3):131–159, 2002.

P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Comm. Pure Appl. Math.*, 57(11): 1413–1457, 2004.

C.-A. Deledalle, S. Vaiter, J. Fadili, and G. Peyré. Stein Unbiased GrAdient estimator of the Risk (SUGAR) for multiple parameter selection. *SIAM J. Imaging Sci.*, 7 (4):2448–2487, 2014.

J. Domke. Generic methods for optimization-based modeling. In *AISTATS*, volume 22, pages 318–326, 2012.

C. Dossal, M. Kachour, M.J. Fadili, G. Peyré, and C. Chesneau. The degrees of freedom of the lasso for general design matrix. *Statistica Sinica*, 23(2):809–828, 2013.

B. Efron. How biased is the apparent error rate of a prediction rule? *J. Amer. Statist. Assoc.*, 81(394):461–470, 1986.

L. C. Evans and R. F. Gariepy. *Measure theory and fine properties of functions*. CRC Press, 1992.

C. S. Foo, C. B. Do, and A. Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in neural information processing systems*, pages 377–384, 2008.

L. Franceschi, M. Donini, P. Frasconi, and M. Pontil. Forward and reverse gradient-based hyperparameter optimization. In *ICML*, pages 1165–1173, 2017.

J. Frecon, S. Salzo, and M. Pontil. Bilevel learning of the group lasso structure. In *Advances in Neural Information Processing Systems*, pages 8301–8311, 2018.

J. Friedman, T. J. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *J. Stat. Softw.*, 33(1):1–22, 2010.

K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *ICML*, pages 399–406, 2010.

E. Hale, W. Yin, and Y. Zhang. Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. *SIAM J. Optim.*, 19(3):1107–1130, 2008.

K. Kunisch and T. Pock. A bilevel optimization approach for parameter learning in variational models. *SIAM J. Imaging Sci.*, 6(2):938–983, 2013.

S. K. Lam, A. Pitrou, and S. Seibert. Numba: A LLVM-based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6. ACM, 2015.

J. Larsen, L. K. Hansen, C. Svarer, and M. Ohlsson. Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop*, 1996.

J. Larsen, C. Svarer, L. N. Andersen, and L. K. Hansen. Adaptive regularization in neural network modeling. In *Neural Networks: Tricks of the Trade - Second Edition*, pages 111–130. Springer, 2012.

J. Liu, X. Chen, Z. Wang, and W. Yin. Alista: Analytic weights are as good as learned weights in lista. In *International Conference on Learning Representations*, 2018.

W. Liu, Y. Yang, et al. Parametric or nonparametric? a parametricness index for model selection. *Ann. Statist.*, 39(4):2074–2102, 2011.

J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. *arXiv preprint arXiv:1911.02590*, 2019.

D. Maclaurin, D. Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, volume 37, pages 2113–2122, 2015.

J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4): 791–804, 2012.

M. Massias, A. Gramfort, and J. Salmon. Celer: a Fast Solver for the Lasso with Dual Extrapolation. In *ICML*, volume 80, pages 3315–3324, 2018.

M. Massias, S. Vaiter, A. Gramfort, and J. Salmon. Dual extrapolation for sparse generalized linear models. *arXiv preprint arXiv:1907.05830*, 2019.

V. Niculae and M. Blondel. A regularized framework for sparse and structured neural attention. In *Advances in neural information processing systems*, pages 3338–3348, 2017.

J. Nocedal and S. J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

F. Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, 2016.

S. Ramani, T. Blu, and M. Unser. Monte-Carlo SURE: a black-box optimization of regularization parameters for general denoising algorithms. *IEEE Trans. Image Process.*, 17(9):1540–1554, 2008.

M. W. Seeger. Cross-validation optimization for large scale structured classification kernel methods. *J. Mach. Learn. Res.*, 9:1147–1178, 2008.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2012.

E. Soubies, L. Blanc-Féraud, and G. Aubert. A unified view of exact continuous penalties for ℓ_2-ℓ_0 minimization. *SIAM J. Optim.*, 27(3):2034–2060, 2017.

C. M. Stein. Estimation of the mean of a multivariate normal distribution. *Ann. Statist.*, 9(6):1135–1151, 1981.

L. R. A. Stone and J.C. Ramer. Estimating WAIS IQ from Shipley Scale scores: Another cross-validation. *Journal of clinical psychology*, 21(3):297–297, 1965.

Y. Sun, H. Jeong, J. Nutini, and M. Schmidt. Are we there yet? manifold identification of gradient-related proximal methods. In *AISTATS*, volume 89, pages 1110–1119, 2019.

R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 58(1):267–288, 1996.

R. J. Tibshirani. The lasso problem and uniqueness. *Electron. J. Stat.*, 7:1456–1490, 2013.

R. J. Tibshirani and J. Taylor. The solution path of the generalized lasso. *Ann. Statist.*, 39(3):1335–1371, 2011.

P. Tseng and S. Yun. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *J. Optim. Theory Appl.*, 140(3):513, 2009.

S. Vaiter, C.-A. Deledalle, G. Peyré, C. Dossal, and J. Fadili. Local behavior of sparse analysis regularization: Applications to risk estimation. *Appl. Comput. Harmon. Anal.*, 35(3):433–451, 2013.

S. Vaiter, C.-A. Deledalle, G. Peyré, J. M. Fadili, and C. Dossal. The degrees of freedom of partly smooth regularizers. *Ann. Inst. Stat. Math.*, 69(4):791–832, 2017.

K. Wu, Y. Guo, Z. Li, and C. Zhang. Sparse coding with gated learned ista. In *International Conference on Learning Representations*, 2019.

B. Xin, Y. Wang, W. Gao, D. Wipf, and B. Wang. Maximal sparsity with deep networks? In *Advances in Neural Information Processing Systems*, pages 4340–4348, 2016.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 68(1):49–67, 2006.

C.-H. Zhang. Nearly unbiased variable selection under minimax concave penalty. *Ann. Statist.*, 38(2):894–942, 2010.

H. Zou. The adaptive lasso and its oracle properties. *J. Amer. Statist. Assoc.*, 101(476):1418–1429, 2006.

H. Zou and T. J. Hastie. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 67(2):301–320, 2005.

H. Zou, T. J. Hastie, and R. Tibshirani. On the "degrees of freedom" of the lasso. *Ann. Statist.*, 35(5):2173–2192, 2007.

A. Proofs

A.1. Proof of Proposition 1

We start by a lemma on the weak derivative of the soft-thresholding.

Lemma A.1. The soft-thresholding $\mathrm{ST} : \mathbb{R} \times \mathbb{R}^+ \mapsto \mathbb{R}$ defined by $\mathrm{ST}(t, \tau) = \mathrm{sign}(t) \cdot (|t| - \tau)_+$ is weakly differentiable with weak derivatives

$$\partial_1 \mathrm{ST}(t, \tau) = \mathbb{1}_{\{|t|>\tau\}} \ , \tag{19}$$

and

$$\partial_2 \mathrm{ST}(t, \tau) = -\mathrm{sign}(t) \cdot \mathbb{1}_{\{|t|>\tau\}} \ , \tag{20}$$

where

$$\mathbb{1}_{\{|t|>\tau\}} = \begin{cases} 1, & \text{if } |t| > \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{21}$$

Proof. See (Deledalle et al., 2014, Proposition 1) □

Proof. (Proposition 1, Lasso ISTA) The soft-thresholding is differentiable almost everywhere (a.e.), thus Equation (10) can be differentiated a.e. thanks to the previous lemma, and for any $\alpha > 0$

$$\hat{\mathcal{J}}_= \begin{pmatrix} \mathbb{1}_{\{|\hat{\beta}_1|>0\}} \\ \vdots \\ \mathbb{1}_{\{|\hat{\beta}_p|>0\}} \end{pmatrix} \odot \left(\mathrm{Id}_p - \frac{1}{\alpha} X^\top X \right) \hat{\mathcal{J}}_- \frac{ne^\lambda}{\alpha} \begin{pmatrix} \mathrm{sign}(\hat{\beta}_1)\mathbb{1}_{\{|\hat{\beta}_1|>0\}} \\ \vdots \\ \mathrm{sign}(\hat{\beta}_p)\mathbb{1}_{\{|\hat{\beta}_p|>0\}} \end{pmatrix} \ .$$

Inspecting coordinates inside and outside the support of $\hat{\beta}$ leads to:

$$\begin{cases} \hat{\mathcal{J}}_{\hat{S}^c} &= 0 \\ \hat{\mathcal{J}}_{\hat{S}} &= \hat{\mathcal{J}}_{\hat{S}} - \frac{1}{\alpha} X_{:,\hat{S}}^\top X_{:,\hat{S}} \hat{\mathcal{J}}_{\hat{S}} - \frac{ne^\lambda}{\alpha} \mathrm{sign}\, \hat{\beta}_{\hat{S}} \ . \end{cases} \tag{22}$$

Rearranging the term of Equation (22) it yields:

$$X_{:,\hat{S}}^\top X_{:,\hat{S}} \hat{\mathcal{J}}_{\hat{S}} = -ne^\lambda \,\mathrm{sign}\, \hat{\beta}_{\hat{S}} \tag{23}$$

$$\hat{\mathcal{J}}_{\hat{S}} = -ne^\lambda \left(X_{:,\hat{S}}^\top X_{:,\hat{S}} \right)^{-1} \mathrm{sign}\, \hat{\beta}_{\hat{S}} \ . \tag{24}$$

(Proposition 1, Lasso BCD)

The fixed point equations for the BCD case is

$$\hat{\beta}_j = \mathrm{ST}\left(\hat{\beta}_j - \frac{1}{\|X_{:j}\|_2^2} X_{:j}^\top (X\hat{\beta}_j - y), \frac{ne^\lambda}{\|X_{:j}\|_2^2} \right) \ . \tag{25}$$

As before we can differentiate this fixed point equation Equation (25)

$$\hat{\mathcal{J}}_j = \mathbb{1}_{\{|\hat{\beta}_j|>\tau\}} \cdot \left(\hat{\mathcal{J}}_j - \frac{1}{\|X_{:j}\|_2^2} X_{:j}^\top X \hat{\mathcal{J}} \right) - \frac{ne^\lambda}{\|X_{:j}\|_2^2} \mathrm{sign}\,(\hat{\beta}_j)\mathbb{1}_{\{|\hat{\beta}_j|>\tau\}} \ , \tag{26}$$

leading to the same result. □

A.2. Proof of Proposition 2 in the ISTA case

Proof. (Lasso case, ISTA) In Algorithm 3, $\beta^{(k)}$ follows ISTA steps, thus $(\beta^{(k)})_{l\in\mathbb{N}}$ converges toward the solution of the Lasso $\hat{\beta}$. Let $\hat{S}$ be the support of the Lasso estimator $\hat{\beta}$, and $\nu^{(\hat{S})} > 0$ the smallest eigenvalue of $X_{:,\hat{S}}^\top X_{:,\hat{S}}$. Under uniqueness assumption proximal gradient descent (*a.k.a.* ISTA) achieves sign identification (Hale et al., 2008), *i.e.,* there exists $k_0 \in \mathbb{N}$ such that for all $k \geq k_0 - 1$:

$$\operatorname{sign} \beta^{(k+1)} = \operatorname{sign} \hat{\beta} \ . \tag{27}$$

Recalling the update of the Jacobian $\mathcal{J}$ for the Lasso solved with ISTA is the following:

$$\mathcal{J}^{(k+1)} = \left|\operatorname{sign} \beta^{(k+1)}\right| \odot \left(\operatorname{Id} - \frac{1}{\|X\|_2^2} X^\top X\right) \mathcal{J}^{(k)} - \frac{ne^\lambda}{\|X\|_2^2} \operatorname{sign} \beta^{(k+1)} \ ,$$

it is clear that $\mathcal{J}^{(k)}$ is sparse with the sparsity pattern $\beta^{(k)}$ for all $k \geq k_0$. Thus we have that for all $k \geq k_0$:

$$\begin{aligned}
\mathcal{J}_{\hat{S}}^{(k+1)} &= \mathcal{J}_{\hat{S}}^{(k)} - \frac{1}{\|X\|_2^2} X_{:,\hat{S}}^\top X \mathcal{J}^{(k)} - \frac{ne^\lambda}{\|X\|_2^2} \operatorname{sign} \hat{\beta}_{\hat{S}} \\
&= \mathcal{J}_{\hat{S}}^{(k)} - \frac{1}{\|X\|_2^2} X_{:,\hat{S}}^\top X_{:,\hat{S}} \mathcal{J}_{\hat{S}}^{(k)} - \frac{ne^\lambda}{\|X\|_2^2} \operatorname{sign} \hat{\beta}_{\hat{S}} \\
&= \left(\operatorname{Id}_{\hat{S}} - \frac{1}{\|X\|_2^2} X_{:,\hat{S}}^\top X_{:,\hat{S}}\right) \mathcal{J}_{\hat{S}}^{(k)} - \frac{ne^\lambda}{\|X\|_2^2} \operatorname{sign} \hat{\beta}_{\hat{S}}.
\end{aligned} \tag{28}$$

One can remark that $\hat{\mathcal{J}}$ defined in Equation (11), satisfies the following:

$$\hat{\mathcal{J}}_{\hat{S}} = \left(\operatorname{Id}_{\hat{S}} - \frac{1}{\|X\|_2^2} X_{:,\hat{S}}^\top X_{:,\hat{S}}\right) \hat{\mathcal{J}}_{\hat{S}} - \frac{ne^\lambda}{\|X\|_2^2} \operatorname{sign} \hat{\beta}_{\hat{S}} \ . \tag{29}$$

Combining Equations (28) and (29) and denoting $\nu^{(\hat{S})} > 0$ the smallest eigenvalue of $X_{\hat{S}}^\top X_{\hat{S}}$, we have for all $k \geq k_0$:

$$\begin{aligned}
\mathcal{J}_{\hat{S}}^{(k+1)} - \hat{\mathcal{J}}_{\hat{S}} &= \left(\operatorname{Id}_{\hat{S}} - \frac{1}{\|X\|_2^2} X_{:,\hat{S}}^\top X_{:,\hat{S}}\right) \left(\mathcal{J}_{\hat{S}}^{(k)} - \hat{\mathcal{J}}_{\hat{S}}\right) \\
\|\mathcal{J}_{\hat{S}}^{(k+1)} - \hat{\mathcal{J}}_{\hat{S}}\|_2 &\leq \left(1 - \frac{\nu^{(\hat{S})}}{\|X\|_2^2}\right) \|\mathcal{J}_{\hat{S}}^{(k)} - \hat{\mathcal{J}}_{\hat{S}}\|_2 \\
\|\mathcal{J}_{\hat{S}}^{(k)} - \hat{\mathcal{J}}_{\hat{S}}\|_2 &\leq \left(1 - \frac{\nu^{(\hat{S})}}{\|X\|_2^2}\right)^{k-k_0} \|\mathcal{J}_{\hat{S}}^{(k_0)} - \hat{\mathcal{J}}_{\hat{S}}\|_2 \ .
\end{aligned}$$

Thus the sequence of Jacobian $\left(\mathcal{J}^{(k)}\right)_{k\in\mathbb{N}}$ converges linearly to $\hat{\mathcal{J}}$ once the support is identified. □

Proof. (wLasso case, ISTA) Recalling the update of the Jacobian $\mathcal{J} \in \mathbb{R}^{p\times p}$ for the wLasso solved with ISTA is the following:

$$\begin{aligned}
\mathcal{J}^{(k+1)} = &\left|\operatorname{sign} \beta^{(k+1)}\right| \odot \left(\operatorname{Id} - \frac{1}{\|X\|_2^2} X^\top X\right) \mathcal{J}^{(k)} \\
&- \frac{ne^\lambda}{\|X\|_2^2} \operatorname{diag}\left(\operatorname{sign} \beta^{(k+1)}\right) \ ,
\end{aligned} \tag{30}$$

The proof follows exactly the same steps as the ISTA Lasso case to show convergence in spectral norm of the sequence $(\mathcal{J}^{(k)})_{k\in\mathbb{N}}$ toward $\hat{\mathcal{J}}$.

□

A.3. Proof of Proposition 2 in the BCD case

The goal of the proof is to show that iterations of the Jacobian sequence $(\mathcal{J}^{(k)})_{k\in\mathbb{N}}$ generated by the Block Coordinate Descent algorithm (Algorithm 3) converges toward the true Jacobian $\hat{\mathcal{J}}$. The main difficulty of the proof is to show that the Jacobian sequence follows a Vector AutoRegressive (VAR, see Massias et al. (2019, Thm. 10) for more detail), *i.e.,* the main difficulty is to show that there exists k_0 such that for all $k \geq k_0$:

$$\mathcal{J}^{(k+1)} = A\mathcal{J}^{(k)} + B \ , \tag{31}$$

with $A \in \mathbb{R}^{p\times p}$ a contracting operator and $B \in \mathbb{R}^p$. We follow exactly the proof of Massias et al. (2019, Thm. 10).

Proof. (Lasso, BCD, forward differentiation (Algorithm 3))

Let $j_1, \dots, j_S$ be the indices of the support of $\hat{\beta}$, in increasing order. As the sign is identified, coefficients outside the support are 0 and remain 0. We decompose the k-th epoch of coordinate descent into individual coordinate updates: Let $\tilde{\beta}^{(0)} \in \mathbb{R}^p$ denote the initialization (i.e., the beginning of the epoch,), $\tilde{\beta}^{(1)} = \beta^{(k)}$ the iterate after coordinate j_1 has been updated, etc., up to $\tilde{\beta}^{(S)}$ after coordinate j_S has been updated, i.e., at the end of the epoch ($\tilde{\beta}^{(S)} = \beta^{(k+1)}$). Let $s \in S$, then $\tilde{\beta}^{(s)}$ and $\tilde{\beta}^{(s-1)}$ are equal everywhere, except at coordinate j_s:

$$\begin{aligned}
\tilde{\mathcal{J}}^{(s)}_{j_s} &= \tilde{\mathcal{J}}^{(s-1)}_{j_s} - \frac{1}{\|X_{:,j_s}\|^2} X_{:,j_s}^\top X \tilde{\mathcal{J}}^{(s-1)} - \frac{1}{\|X_{j_s}\|^2} \operatorname{sign} \beta_{j_s} \quad \text{after sign identification we have:} \\
&= \tilde{\mathcal{J}}^{(s-1)}_{j_s} - \frac{1}{\|X_{:,j_s}\|^2} X_{:,j_s}^\top X_{:,\hat{S}} \tilde{\mathcal{J}}^{(s-1)}_{\hat{S}} - \frac{1}{\|X_{:,j_s}\|^2} \operatorname{sign} \hat{\beta}_{j_s} \\
\tilde{\mathcal{J}}^{(s)}_{\hat{S}} &= \underbrace{\left(\mathrm{Id}_{\hat{s}} - \frac{1}{\|X_{:,j_s}\|^2} e_{j_s} e_{j_s}^\top X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)}_{A_s} \tilde{\mathcal{J}}^{(s-1)}_{\hat{S}} - \frac{1}{\|X_{:,j_s}\|^2} \operatorname{sign} \hat{\beta}_{j_s}
\end{aligned}$$

$$\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \tilde{\mathcal{J}}^{(s)}_{\hat{S}} = \underbrace{\left(\mathrm{Id}_{\hat{s}} - \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s} e_{j_s}^\top \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|}\right)}_{A^{(s)}} \left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \tilde{\mathcal{J}}^{(s-1)}_{\hat{S}} - \underbrace{\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|^2} \operatorname{sign} \hat{\beta}_{j_s}}_{b^{(s)}}$$

We thus have:

$$\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \tilde{\mathcal{J}}^{(\hat{s})}_{\hat{S}} = \underbrace{A^{(\hat{s})} \dots A^{(1)}}_{A\in\mathbb{R}^{\hat{s}\times\hat{s}}} \left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \mathcal{J}^{(1)}_{\hat{S}} + \underbrace{A_S \dots A_2 b_1 + \dots + A_S b_{S-1} + b_S}_{b\in\mathbb{R}^{\hat{s}}} \ .$$

After sign identification and a full update of coordinate descent we thus have:

$$\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \mathcal{J}^{(t+1)}_{\hat{S}} = A \left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2} \mathcal{J}^{(t)}_{\hat{S}} + b \ . \tag{32}$$

Lemma A.2.

$$\|A_s\|_2 \leq 1 \ ,$$

Moreover if $\left\|A^{(s)}x\right\| = \|x\|$ then

$$x \in \operatorname{vect}\left(\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s}\right)^\top \tag{33}$$

Proof.

$$\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s} e_{j_s}^\top \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|}$$

is a symmetric rank 1 matrix, its non-zero eigenvalue is

$$e_{j_s}^\top \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} \frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s} = e_{j_s}^\top \frac{X_{:,\hat{S}}^\top X_{:,\hat{S}}}{\|X_{:,j_s}\|^2} e_{j_s} = 1 \ .$$

An eigenvector associated to this non-zeros eigenvalue is

$$\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s} \ .$$

A_s is symmetric and real, is diagonalisable in an orthogonal basis, it has eigenvalue 1 with multiplicity $\hat{s}-1$ and eigenvalue 0 with multiplicity 1. Moreover if $\|Ax\| = \|x\|$, then $x \in \text{vect}\left(\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s}\right)^\top$. □

Lemma A.3.

$$\|A\|_2 < 1 \ .$$

Proof. $A = A^{(\hat{s})} \dots A^{(1)}$ We have

$$\|A\| \leq \underbrace{\|A^{(\hat{s})}\|}_{\leq 1} \dots \underbrace{\|A^{(1)}\|}_{\leq 1} \leq 1 \ .$$

Let $x \in \mathbb{R}^{\hat{s}}$ such that $\|Ax\| = \|x\|$, we thus have for all $s \in 1, \dots, \hat{s}$, $\left\|A^{(s)}x\right\| = \|x\|$. Using Lemma A.3 we have that for all $s \in 1, \dots, \hat{s}$ $x \in \text{vect}\left(\frac{\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}}{\|X_{:,j_s}\|} e_{j_s}\right)^\top$, *i.e.,* $x \in \text{vect}\left(\left(X_{:,\hat{S}}^\top X_{:,\hat{S}}\right)^{1/2}\right)^\top = \{0\}$ because $X_{:,\hat{S}}^\top X_{:,\hat{S}} \succ 0$ □

Using Equation (32) we have:

$$\|\mathcal{J}_{\hat{S}}^{(t+1)} - \hat{\mathcal{J}}\|_{(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{-1}} \leq \|A\|_2 \|\mathcal{J}_{\hat{S}}^{(t)} - \hat{\mathcal{J}}\|_{(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{-1}} \ , \tag{34}$$

with $\|A\|_2 < 1$, which leads to the desire result. Since the recursion of the Jacobian sequences of Algorithm 2 and Algorithm 2 are the same once the support is identified, the proof of convergence of Algorithm 2 is the same (provided that support identification has been achieved). □

Proof. (wLasso case, BCD) As for the Lasso case:

$$\tilde{\mathcal{J}}_{j_s,:}^{(s)} = \tilde{\mathcal{J}}_{j_s,:}^{(s-1)} - \frac{1}{\|X_{:,j_s}\|^2} X_{:,j_s}^\top X \tilde{\mathcal{J}}^{(s-1)} - \frac{1}{\|X_{j_s}\|^2} \operatorname{sign} \beta_{j_s} e_{j_s} e_{j_s}^\top \quad \text{after sign identification we have:}$$

$$\tilde{\mathcal{J}}_{j_s,\hat{S}}^{(s)} = \tilde{\mathcal{J}}_{j_s,\hat{S}}^{(s-1)} - \frac{1}{\|X_{:,j_s}\|^2} X_{:,j_s}^\top X_{:,\hat{S}} \tilde{\mathcal{J}}_{\hat{S},\hat{S}}^{(s-1)} - \frac{1}{\|X_{:,j_s}\|^2} \operatorname{sign} \hat{\beta}_{j_s} e_{j_s} e_{j_s}^\top$$

$$(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} \tilde{\mathcal{J}}_{\hat{S},\hat{S}}^{(s)} = \underbrace{\left(\mathrm{Id}_n - \frac{(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} e_{j_s} e_{j_s}^\top (X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2}}{\|X_{:,j_s}\|^2}\right)}_{A^{(s)}} (X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} \tilde{\mathcal{J}}_{\hat{S},\hat{S}}^{(s-1)}$$

$$- \underbrace{\frac{\operatorname{sign} \hat{\beta}_{j_s}}{\|X_{:,j_s}\|^2} (X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} e_{j_s} e_{j_s}^\top}_{B^{(s)}}$$

$$(X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} \tilde{\mathcal{J}}_{\hat{S},\hat{S}}^{(\hat{s})} = \underbrace{A^{(\hat{s})} \dots A^{(1)}}_{A \in \mathbb{R}^{\hat{s} \times \hat{s}}} (X_{:,\hat{S}}^\top X_{:,\hat{S}})^{1/2} \tilde{\mathcal{J}}_{\hat{S},\hat{S}}^{(0)} + \underbrace{A^{(\hat{s})} \dots A^{(2)} B^{(1)} e_{j_1} e_{j_1}^\top + \dots + B^{(\hat{s})} e_{j_{\hat{s}}} e_{j_{\hat{s}}}^\top}_{D \in \mathbb{R}^{\hat{s} \times \hat{s}}} \ . \tag{35}$$

As in the Lasso case, Equation (35) leads to linear convergence once the support is identified for Algorithms 2 and 3. □

B. Block coordinate descent algorithms

Algorithm 3 presents the forward iteration scheme which computes iteratively the solution of the Lasso or wLasso jointly with the Jacobian computation. This is the *naive* way of computing the Jacobian without taking advantage of its sparsity. Eventually, it requires to differentiate every lines of code *w.r.t.* to λ and take advantage of the BCD updates for cheap updates on the Jacobian as well.

Algorithm 3 FORWARD ITERDIFF (Deledalle et al., 2014; Franceschi et al., 2017)

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}, n_{\text{iter}} \in \mathbb{N}$
// jointly compute coef. & Jacobian
$\beta = 0$ // potentially warm started
$\mathcal{J} = 0$ // potentially warm started
$r = y - X\beta$
$dr = -X\mathcal{J}$
for $k = 0, \dots, n_{iter} - 1$ **do**
  for $j = 0, \dots, p - 1$ **do**
    // update the regression coefficients
    $\beta_{\text{old}} = \beta_j$
    $z_j = \beta_j + \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top r$ // gradient step
    $\beta_j = \mathrm{ST}(z_j, ne^\lambda / \|X_{:,j}\|^2)$ // proximal step
    $r \mathrel{-}= X_{:,j}(\beta_j - \beta_{\text{old}})$
    // update the Jacobian
    if *Lasso* **then**
      $\mathcal{J}_{old} = \mathcal{J}_j$
      $\mathcal{J}_j = |\operatorname{sign} \beta_j| \left(\mathcal{J}_j + \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top dr \right)$ // diff. w.r.t. λ
      $\mathcal{J}_j \mathrel{-}= \frac{ne^\lambda}{\|X_{:,j}\|^2} \operatorname{sign} \beta_j$ // diff. w.r.t. λ
      $dr_j \mathrel{-}= X_{:,j}(\mathcal{J}_j - \mathcal{J}_{old})$
    if *wLasso* **then**
      $\mathcal{J}_{old} = \mathcal{J}_{j,:}$
      $\mathcal{J}_{j,:} = |\operatorname{sign} \beta_j| \left(\mathcal{J}_{j,:} + \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top dr \right)$ // diff. w.r.t. $\lambda_1, \dots, \lambda_p$
      $\mathcal{J}_{j,j} \mathrel{-}= \frac{ne^{\lambda_j}}{\|X_{:,j}\|^2} \operatorname{sign} \beta_j$ // diff. w.r.t. $\lambda_1, \dots, \lambda_p$
      $dr \mathrel{-}= X_{:,j}(\mathcal{J}_j - \mathcal{J}_{old})$
return $\beta^{n_{iter}}, \mathcal{J}_{(\lambda)}^{n_{iter}}$

Algorithm 4 describes the backward iterative differentiation algorithm used for benchmark. Backward differentiation requires the storage of every updates on β. As Figure 1 shows, this algorithm is not efficient for our case because the function to differentiate $f : \mathbb{R} \to \mathbb{R}^p$ ($f : \mathbb{R}^p \to \mathbb{R}^p$, for the wLasso) has a higher dimension output space than the input space. The storage is also an issue mainly for the wLasso case which makes this algorithm difficult to use in practice in our context.

Algorithm 5 presents the classical BCD iterative scheme for solving the Lasso problem using the composition of a gradient step with the soft-thresholding operator.

Algorithm 4 BACKWARD ITERDIFF (Domke, 2012)

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}, n_{\text{iter}} \in \mathbb{N}$
// backward computation of $\hat{\beta}$ and $\hat{\mathcal{J}}_{(\lambda)}^\top \alpha$
$\beta = 0$ // potentially warm started
// compute the regression coefficients and store the iterates
for $k = 0, \dots, n_{iter} - 1$ **do**
  for $j = 0, \dots, p - 1$ **do**
    $\beta_{\text{old}} = \beta_j$
    $z_j = \beta_j + \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top r$ // gradient step
    $\beta_j = \mathrm{ST}(z_j, ne^\lambda / \|X_{:,j}\|^2)$ // proximal step
    $r \mathrel{-}= X_{:,j}(\beta_j - \beta_{\text{old}})$
// Init. backward differentiation
$g = 0$ // g stores $\hat{\mathcal{J}}_\lambda^\top \alpha$
// compute the Jacobian
for $k = n_{iter}$ ***down to*** 1 **do**
  for $j = 0, \dots, p - 1$ **do**
    if *Lasso* **then**
      $g \mathrel{-}= \frac{ne^\lambda}{\|X_{:,j}\|^2} \alpha_j \operatorname{sign} \beta_j^{(k)}$
      $\alpha_j \mathrel{*}= |\operatorname{sign} \beta_j^{(k)}|$
      $\alpha \mathrel{-}= \frac{1}{\|X_{:,j}\|^2} \alpha_j X_{:,j}^\top X$ // $\mathcal{O}(np)$
    if *wLasso* **then**
      $g_j \mathrel{-}= \frac{ne^{\lambda_j}}{\|X_{:,j}\|^2} \alpha_j \operatorname{sign} \beta_j^{(k)}$
      $\alpha_j \mathrel{*}= |\operatorname{sign} \beta_j^{(k)}|$
      $\alpha \mathrel{-}= \frac{1}{\|X_{:,j}\|^2} \alpha_j X_{:,j}^\top X$
return $\beta^{n_{iter}}, g^{(1)}$

Algorithm 5 BCD FOR THE LASSO (Friedman et al., 2010)

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}, \beta^{(0)} \in \mathbb{R}^p, n_{\text{iter}} \in \mathbb{N}$
$\beta = \beta^{(0)}$ // warm start
for $k = 0, \dots, n_{iter} - 1$ **do**
  for $j = 0, \dots, p - 1$ **do**
    $\beta_{\text{old}} = \beta_j$
    $z_j = \beta_j + \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top r$ // gradient step
    $\beta_j = \mathrm{ST}(z_j, ne^\lambda / \|X_{:,j}\|^2)$ // proximal step
    $r \mathrel{-}= X_{:,j}(\beta_j - \beta_{\text{old}})$
return $\beta^{n_{iter}}$

C. Derivations for MCP

Let us remind the definition of the Minimax Concave Penalty (MCP) estimator introduced by Zhang (2010), also analyzed under the name CELE0 by Soubies et al. (2017). First of all, for any $t \in \mathbb{R}$:

$$p^{\text{MCP}}_{\lambda,\gamma}(t) = \begin{cases} \lambda|t| - \frac{t^2}{2\gamma}, & \text{if } |t| \leq \gamma\lambda \\ \frac{1}{2}\gamma\lambda^2, & \text{if } |t| > \gamma\lambda \ . \end{cases} \tag{36}$$

The proximity operator of $p_{\lambda,\gamma}$ for parameters $\lambda > 0$ and $\gamma > 1$ is defined as follow (see Breheny and Huang 2011, Sec. 2.1):

$$\text{prox}^{\text{MCP}}_{\lambda,\gamma}(t) = \begin{cases} \frac{\text{ST}(t,\lambda)}{1-\frac{1}{\gamma}} & \text{if } |t| \leq \gamma\lambda \\ t & \text{if } |t| > \gamma\lambda \ . \end{cases} \tag{37}$$

For ourselves we choose as for the Lasso an exponential parametrization of the coefficients, for $\lambda \in \mathbb{R}$ and $\gamma > 0$:

$$\hat{\beta}^{(\lambda,\gamma)}(y) \triangleq \underset{\beta\in\mathbb{R}^p}{\arg\min} \frac{1}{2n}\|y - X\beta\|_2^2 + \sum_{j=1}^{p} p^{\text{MCP}}_{e^\lambda,e^\gamma}(|\beta_j|) \ . \tag{38}$$

Update rule for Coordinate Descent Below, we provide equation to update the coefficient in the coordinate descent algorithm of the MCP:

$$\begin{aligned}
\beta_j &\leftarrow \underset{\beta_j\in\mathbb{R}}{\arg\min} \frac{1}{2n}\|y - \beta_j X_{:,j} - \sum_{j'\neq j} \beta_{j'} X_{:,j'}\|_2^2 + \sum_{j'\neq j}^{p} p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_{j'}) + p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_j) \\
&= \underset{\beta_j\in\mathbb{R}}{\arg\min} \frac{1}{2n}\|y - \beta_j X_{:,j} - \sum_{j'\neq j} \beta_{j'} X_{:,j'}\|_2^2 + p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_j) \\
&= \underset{\beta_j\in\mathbb{R}}{\arg\min} \|X_{:,j}\|_2^2 \left(\frac{1}{2n} \left[\beta_j - \frac{1}{\|X_{:,j}\|_2^2} \left\langle y - \sum_{j'\neq j} \beta_{j'} X_{:,j'}, X_{:,j} \right\rangle \right]^2 + \frac{1}{\|X_{:,j}\|_2^2} p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_j) \right) \\
&= \underset{\beta_j\in\mathbb{R}}{\arg\min} \left(\frac{1}{2n} \left[\beta_j - \frac{1}{\|X_{:,j}\|_2^2} \left\langle y - \sum_{j'\neq j} \beta_{j'} X_{:,j'}, X_{:,j} \right\rangle \right]^2 + \frac{1}{\|X_{:,j}\|_2^2} p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_j) \right) \\
&= \underset{\beta_j\in\mathbb{R}}{\arg\min} \left(\frac{1}{2L_j} \left[\beta_j - \frac{1}{\|X_{:,j}\|_2^2} \left\langle y - \sum_{j'\neq j} \beta_{j'} X_{:,j'}, X_{:,j} \right\rangle \right]^2 + p^{\text{MCP}}_{e^\lambda,e^\gamma}(\beta_j) \right), \text{with } L_j \triangleq \frac{n}{\|X_{:,j}\|_2^2} \\
&= \text{prox}^{\text{MCP}}_{e^\lambda/L_j,e^\gamma L_j} \left(\beta_j - \frac{1}{\left\|X_{:,j}^2\right\|} X_{:,j}^\top (X\beta - y), \lambda \right) \ .
\end{aligned} \tag{39}$$

One can write the following fixed point equation satisfied by the estimator $\hat{\beta}$, with $L_j = \|X_{:,j}\|^2 / n$:

$$\begin{aligned}
\hat{\beta}_j &= \text{prox}^{\text{MCP}}_{e^\lambda/L_j,e^\gamma L_j} \left(\left\langle y - \sum_{k\neq j} \hat{\beta}_k X_{:,k}, \frac{X_{:,j}}{\|X_{:,j}\|^2} \right\rangle \right) \\
&= \text{prox}^{\text{MCP}}_{e^\lambda/L_j,e^\gamma L_j} \left(\hat{\beta}_j - \frac{1}{\|X_{:,j}\|^2} X_{:,j}^\top \left(X\hat{\beta} - y \right) \right) \ .
\end{aligned} \tag{40}$$

Since the MCP penalty is non-convex, the estimator may not be continuous *w.r.t.* hyperparameters and gradient based hyperparameter optimization may not be theoretically justified. However we can differentiate the fixed point equation

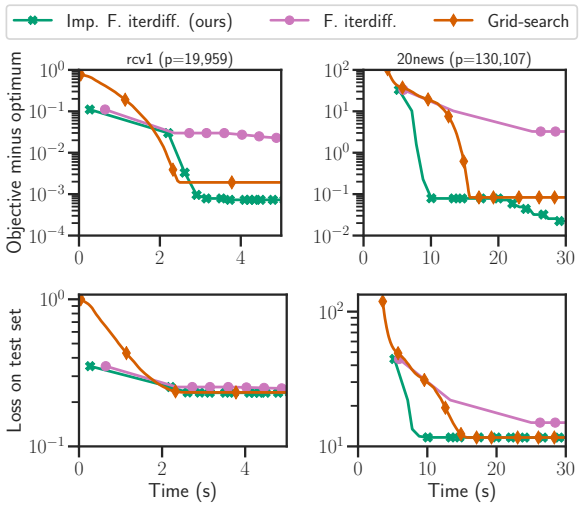

Figure 5. **Computation time for the HO of the MCP on real data** Distance to "optimum" (top) and performance (bottom) on the test set for the MCP.

Equation (40) almost everywhere:

$$\begin{aligned}\hat{\mathcal{J}}_j = \left(\hat{\mathcal{J}}_j - \frac{1}{\|X_{:j}\|_2^2} X_{:j}^\top X \hat{\mathcal{J}}\right) \cdot \frac{\partial \operatorname{prox}^{\text{MCP}}_{e^\lambda/L_j, e^\gamma L_j}}{\partial t} \left(\hat{\beta}_j - \frac{1}{\|X_{:,j}^2\|} X_{:,j}^\top (X\beta - y)\right) \\ + \frac{e^\lambda}{L_j} \frac{\partial \operatorname{prox}^{\text{MCP}}_{e^\lambda/L_j, e^\gamma L_j}}{\partial \lambda} \left(\hat{\beta}_j - \frac{1}{\|X_{:,j}^2\|} X_{:,j}^\top (X\beta - y)\right) \\ + e^\gamma L_j \frac{\partial \operatorname{prox}^{\text{MCP}}_{e^\lambda/L_j, e^\gamma L_j}}{\partial \gamma} \left(\hat{\beta}_j - \frac{1}{\|X_{:,j}^2\|} X_{:,j}^\top (X\beta - y)\right) \quad . \end{aligned} \tag{41}$$

where

$$\frac{\partial \operatorname{prox}^{\text{MCP}}_{\lambda,\gamma}}{\partial t}(t) = \begin{cases} \frac{|\operatorname{sign} t|}{1-\frac{1}{\gamma}}, & \text{if } |t| \leq \lambda\gamma \\ 1, & \text{otherwise} \end{cases} , \tag{42}$$

$$\frac{\partial \operatorname{prox}^{\text{MCP}}_{\lambda,\gamma}}{\partial \lambda}(t) = \begin{cases} 0, & \text{if } |t| \leq \lambda \\ -\frac{\operatorname{sign} t}{1-\frac{1}{\gamma}}, & \text{if } \lambda \leq |t| \leq \lambda\gamma \\ 0, & \text{if } |t| > \lambda\gamma \end{cases} , \tag{43}$$

$$\frac{\partial \operatorname{prox}^{\text{MCP}}_{\lambda,\gamma}}{\partial \gamma}(t) = \begin{cases} -\frac{\operatorname{ST}(t,\lambda)}{(\gamma-1)^2} & \text{if } |t| \leq \lambda\gamma \\ 0 & \text{if } |t| > \lambda\gamma \end{cases} . \tag{44}$$

Contrary to other methods, HO based algorithms do not scale exponentially in the number of hyperparameters. Here we propose experiments on the held-out loss with the MCP estimator (Zhang, 2010), which has 2 hyperparameters λ and γ. Our algorithm can generalize to such non-smooth proximity-based estimator.

Comments on Figure 5 (MCP, held-out criterion). Figure 5 (top) shows the convergence of the optimum on 2 datasets (*rcv1* and *20news*) for the MCP estimator. As before implicit forward differentiation outperforms forward differentiation illustrating Proposition 2 and Table 1.

D. Datasets and implementation details

The code used to produce all the figures as well as the implementation details can be found in the supplementary material in the $forward\_implicit/expes$ folder. In particular in all experiments, for our algorithm, implicit forward differentiation, the size of the loop computing the Jacobian is fixed: n_iter_jac $= 100$. Reminding that the goal is to compute the gradient:

$$\hat{\mathcal{J}}_{(\lambda)}^\top \nabla \mathcal{C}\left(\hat{\beta}^{(\lambda)}\right) \ , \tag{45}$$

we break the loop if

$$\|(\mathcal{J}^{(k+1)} - \mathcal{J}^{(k)})\nabla \mathcal{C}(\hat{\beta}^{(\lambda)})\| \leq \|\nabla \mathcal{C}(\hat{\beta}^{(\lambda)})\| \times \epsilon^{\text{jac}} \ , \tag{46}$$

with $\epsilon^{\text{jac}} = 10^{-3}$. All methods benefit from warm start.

D.1. Details on Figure 1

Figure 1 is done using synthetic data. As described in Section 4.2, $X \in \mathbb{R}^{n \times p}$ is a Toeplitz correlated matrix, with correlation coefficient $\rho = 0.9$, $(n, p) = (1000, 2000)$. $\beta \in \mathbb{R}^p$ is chosen with 5 non-zero coefficients chosen at random. Then $y \in \mathbb{R}^n$ is chosen to be equal to $X\beta$ contaminated by some i.i.d. random Gaussian noise, we chose SNR $= 3$. For Figure 1 all the implementation details can be found in the joint code in the $forward\_implicit/examples/plot\_time\_to\_compute\_single\_gradient.py$ file. Figure 1 shows the time of computation of one gradient and the distance to "optimum". For this figure we evaluated the gradient in $\lambda = \lambda_{\text{max}} - \ln(10)$. The "optimum" is the gradient obtained using the implicit differentiation method.

D.2. Details on Figure 2

Let us first begin by a description of all the datasets and where they can be downloaded.

rcv1. The *rcv1* dataset can be downloaded here: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html#rcv1v2%20(topics;%20subsets). The dataset contains $n = 20,242$ samples and $p = 19,959$ features.

20news. The *20news* dataset can be downloaded here https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html#news20. The dataset contains $n = 11,314$ samples and $p = 130,107$ features.

finance. The *finance* (*E2006-log1p* on libsvm) dataset can be downloaded here: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#E2006-log1p. The dataset contains $n = 16,087$ samples and $p = 1,668,737$ features.

All the implementation details can be found in the code: $forward\_implicit/expes/main\_lasso\_pred.py$.

D.3. Details on Figure 3

Figure 3 was performed using simulated data. The matrix $X \in \mathbb{R}^{n \times p}$ was obtained by simulated $n \times p$ i.i.d. Gaussian variables $\mathcal{N}(0, 1)$. The number of rows was fixed at $n = 100$ and we changed the number of columns p from 200 to 10,000 on a linear grid of size 10. Then , we generated β^* with 5 coefficients equal to 1 and the rest equals to 0. The vector y is equal to $X\beta^*$ contaminated by some i.i.d. random Gaussian noise controlled by a SNR value of 3. We performed 50 repetitions for each value of p and computed the average MSE on these repetitions. The initial value for the line-search algorithm was set at $\lambda_{\text{max}} + \ln(0.7)$ and the number of iterations for the Jacobian at 500 for the whole experiment. All the implementation details can be found in the code : $forward\_implicit/expes/main\_lasso\_est.py$.

D.4. Details on Figure 4

Figure 4 was performed using the same simulating process as described above only this time we performed only 25 repetitions for each value of p. We had to deal with the fact that Problem (4) is not convex for the weighted Lasso which means that our line-search algorithm could get stuck in local minima. In order to alleviate this problem, we introduced Equation (18) to obtain an initial point for the line-search algorithm. We chose the regularization term to be constant and equals

to $C(\beta^{(\lambda_{\max})})/10$. We used a time treshold of 500 seconds which was hit only by the forward differentiation algorithm for the wLasso. The details about this experiment can be found in the code : $forward\_implicit/expes/main\_wLasso.py$.

E. Supplementary experiments

E.1. Experiments with a non-unique solution to the inner problem

We recall here that the bi-level optimization Problem (4) is solved using gradient descent. We recall also that gradient descent may not converge toward a global minima since the optimized function $\lambda \mapsto \mathcal{L}(\lambda)$ may not be convex. It may be even worse: if the inner optimization problem has not a unique solution, the function $\lambda \mapsto \mathcal{L}(\lambda)$ may not be continuous. However our algorithm can still be applied to compute the hypergradient. Figure 6 shows the time to compute a single (hyper)gradient when the solution to the inner problem is not unique.

As proved for instance in Tibshirani (2013, Lemma 3 and 4), the set of parameters where the Lasso solution is not unique is typically $\emptyset$ or a set whose Lebesgue measure is zero. Moreover, there exist settings such that the solution path (as a multivalued mapping) could be non-continuous, which leaves only non-gradient based methods available. Thus, we decided to not investigate the theory in such pathological settings. The authors are not aware of a classical dataset where non-uniqueness arises. Nevertheless, in the case where there exists λ such that the solution set is not reduced to a singleton, our proposed algorithm can still be applied to any solution without theoretical guarantees.

Experimental setting for non-uniqueness. For completeness, we run our methods on the following toy example Tibshirani (2013): we consider a design X such that $n = 100$, $p = 10000$ and X_1, X_2, X_3 are generated iid following a standard normal distribution, $X_4 = (X_2 + X_3)/2$ and $X_5, \dots, X_p$ are generated i.i.d. following a standard normal distribution, then orthogonalized $X_5, \dots, X_p$ *w.r.t.* $\mathrm{Span}(X_1, \dots, X_4)$. We let $y = -X1 + X2 + X3$. We let the reader observe that when $\lambda = 1/n$, the solution set is not reduced to a singleton. In this case, similar conclusions are observed in Figure 6 as for other datasets: Imp. F. Iterdiff (ours) still outperforms its competitors.

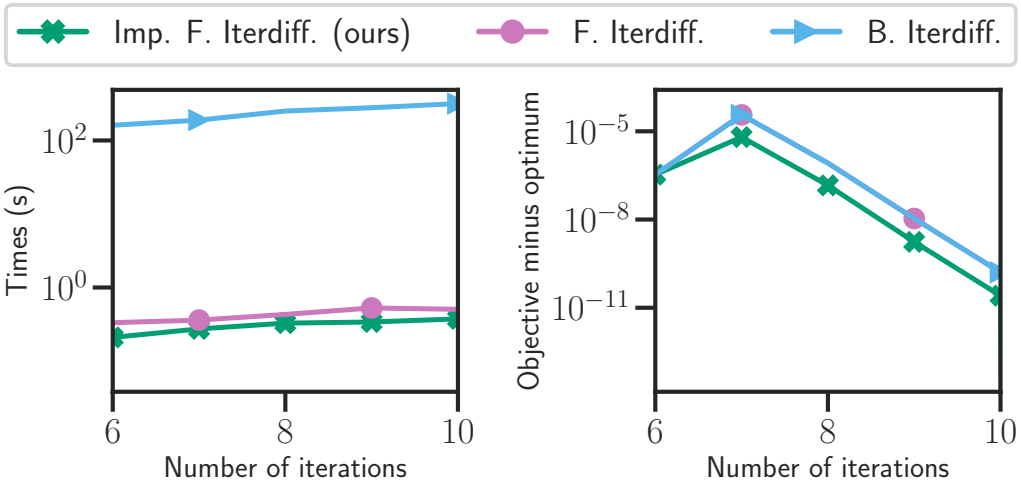

Figure 6. **Time to compute a single gradient with non-unique solution** (Synthetic data, Lasso, $n, p = 1000, 10000$). Influence on the number of iterations of BCD (in the inner optimization problem of Problem (4)) on the computation time (left) and the distance to "optimum" of the gradient $\nabla_\lambda \mathcal{L}(\lambda)$(right) for the Lasso estimator.