Implicit differentiation for fast hyperparameter selection in non-smooth convex learning

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

Implicit differentiation for fast hyperparameter selection in non-smooth convex learning

Quentin Bertrand* QUENTIN.BERTRAND@INRIA.FR
Université Paris-Saclay, Inria, CEA, Palaiseau, France

Quentin Klopfenstein* QUENTIN.KLOPFENSTEIN@U-BOURGOGNE.FR
Institut Mathématique de Bourgogne, Université de Bourgogne, Dijon, France

Mathurin Massias MATHURIN.MASSIAS@GMAIL.COM
MaLGa, DIBRIS, Università degli Studi di Genova, Genova, Italy

Mathieu Blondel MBLONDEL@GOOGLE.COM
Google Research, Brain team, Paris, France

Samuel Vaiter SAMUEL.VAITER@MATH.CNRS.FR
CNRS and Institut Mathématique de Bourgogne, Université de Bourgogne, Dijon, France

Alexandre Gramfort ALEXANDRE.GRAMFORT@INRIA.FR
Université Paris-Saclay, Inria, CEA, Palaiseau, France

Joseph Salmon JOSEPH.SALMON@UMONTPELLIER.FR
IMAG, Université de Montpellier, CNRS, Montpellier, France

Editor: ???

Abstract

Finding the optimal hyperparameters of a model can be cast as a bilevel optimization problem, typically solved using zero-order techniques. In this work we study first-order methods when the inner optimization problem is convex but non-smooth. We show that the forward-mode differentiation of proximal gradient descent and proximal coordinate descent yield sequences of Jacobians converging toward the exact Jacobian. Using implicit differentiation, we show it is possible to leverage the non-smoothness of the inner problem to speed up the computation. Finally, we provide a bound on the error made on the hypergradient when the inner optimization problem is solved approximately. Results on regression and classification problems reveal computational benefits for hyperparameter optimization, especially when multiple hyperparameters are required.

1. Introduction

Almost all models in machine learning require at least one hyperparameter, the tuning of which drastically affects accuracy. This is the case for many popular estimators, where the regularization hyperparameter controls the trade-off between a data fidelity term and a regularization term. Such estimators, including Ridge regression (Hoerl and Kennard,

Table 1: Examples of non-smooth inner problems as in (1).

Inner problem, Φ	$f(\beta)$	$g_j(\beta_j, \lambda)$	$e^{\lambda_{\max}}$
Lasso	$\frac{1}{2n}\|y - X\beta\|^2$	$e^{\lambda}\|\beta_j\|$	$\frac{1}{n}\|X^\top y\|_\infty$
elastic net	$\frac{1}{2n}\|y - X\beta\|^2$	$e^{\lambda_1}\|\beta_j\| + \frac{1}{2}e^{\lambda_2}\beta_j^2$	$\frac{1}{n}\|X^\top y\|_\infty$
sparse log. reg.	$\frac{1}{n}\sum_{i=1}^n \ln(1 + e^{-y_i X_{i:}\beta})$	$e^{\lambda}\|\beta_j\|$	$\frac{1}{2n}\|X^\top y\|_\infty$
dual SVM	$\frac{1}{2}\|(y \odot X)^\top \beta\|^2 - \sum_{j=1}^p \beta_j$	$\iota_{[0, e^\lambda]}(\beta_j)$	–

1970), Lasso (Tibshirani, 1996; Chen et al., 1998), elastic net (Zou and Hastie, 2005), sparse logistic regression (Koh et al., 2007), support-vector machine/SVM (Boser et al., 1992; Platt, 1999) are often cast as an optimization problem (Table 1)

$$\hat{\beta}^{(\lambda)} \in \arg\min_{\beta \in \mathbb{R}^p} \Phi(\beta, \lambda) \triangleq f(\beta) + \underbrace{\sum_{j=1}^p g_j(\beta_j, \lambda)}_{\triangleq g(\beta, \lambda)} \ , \tag{1}$$

with smooth $f : \mathbb{R}^p \to \mathbb{R}$ (*i.e.,* with Lipschitz gradient), proper closed convex (possibly non-smooth) functions $g_j(\cdot, \lambda)$, and a regularization hyperparameter $\lambda \in \mathbb{R}^r$. In the examples of Table 1, the computation of f involves a design matrix $X \in \mathbb{R}^{n \times p}$; and the cost of computing $\nabla f(\beta)$ is $\mathcal{O}(np)$. In the SVM example, since we consider the dual problem, we chose to reverse the roles of n and p to enforce $\beta \in \mathbb{R}^p$. We often drop the λ dependency and write $\hat{\beta}$ instead of $\hat{\beta}^{(\lambda)}$ when it is clear from context.

For a fixed λ, the issue of solving efficiently Problem (1) has been largely explored. If the functions g_j are smooth, one can use solvers such as L-BFGS (Liu and Nocedal, 1989), SVRG (Johnson and Zhang, 2013; Zhang et al., 2013), or SAGA (Defazio et al., 2014). When the functions g_j are non-smooth, Problem (1) can be tackled efficiently with stochastic algorithms (Pedregosa et al., 2017) or using working set methods (Fan and Lv, 2008; Tibshirani et al., 2012) combined with coordinate descent (Tseng and Yun, 2009), see overview by Massias et al. (2020). The question of *model selection*, *i.e.,* how to select the hyperparameter $\lambda \in \mathbb{R}^r$ (potentially multidimensional), is more open, especially when the dimension r of the regularization hyperparameter λ is large.

For the Lasso, a broad literature has been devoted to parameter tuning. Under strong hypothesis on the design matrix X, it is possible to derive guidelines for the setting of the regularization parameter λ (Lounici, 2008; Bickel et al., 2009; Belloni et al., 2011). Unfortunately, these guidelines rely on quantities which are typically unknown in practice, and Lasso users still have to resort to other techniques to select the hyperparameter λ.

A popular approach for hyperparameter selection is *hyperparameter optimization* (Kohavi and John, 1995; Hutter et al., 2015; Feurer and Hutter, 2019): one selects the hyperparameter λ such that the regression coefficients $\hat{\beta}^{(\lambda)}$ minimize a given criterion $\mathcal{C} : \mathbb{R}^p \to \mathbb{R}$.

Table 2: Examples of outer criteria used for hyperparameter selection.

Criterion	Problem type	Criterion $\mathcal{C}(\beta)$
Hold-out mean squared error	Regression	$\frac{1}{n}\|y^{\text{val}} - X^{\text{val}}\beta\|^2$
Stein unbiased risk estimate (SURE)[1]	Regression	$\|y - X\beta\|^2 - n\sigma^2 + 2\sigma^2\text{dof}(\beta)$
Hold-out logistic loss	Classification	$\frac{1}{n}\sum_{i=1}^{n} \ln(1 + e^{-y_i^{\text{val}} X_{i:}^{\text{val}}\beta})$
Hold-out smoothed Hinge loss[2]	Classification	$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i^{\text{val}}, X_{i:}^{\text{val}}\beta)$

Here $\mathcal{C}$ should ensure good generalization, or avoid overcomplex models. Common examples (see Table 2) include the hold-out loss (Devroye and Wagner, 1979), the cross-validation loss (CV, Stone and Ramer 1965, see Arlot and Celisse 2010 for a survey), the AIC (Akaike, 1974), BIC (Schwarz, 1978) or SURE (Stein, 1981) criteria. Formally, the hyperparameter optimization problem is a bilevel optimization problem (Colson et al., 2007):

$$\begin{aligned} &\underset{\lambda \in \mathbb{R}^r}{\arg\min} \left\{ \mathcal{L}(\lambda) \triangleq \mathcal{C}\left(\hat{\beta}^{(\lambda)}\right) \right\} \\ &s.t.\ \hat{\beta}^{(\lambda)} \in \underset{\beta \in \mathbb{R}^p}{\arg\min}\, \Phi(\beta, \lambda) \ . \end{aligned} \tag{2}$$

Popular approaches to solve (the generally non-convex) Problem (2) include zero-order optimization (gradient-free) techniques such as grid-search, random-search (Rastrigin, 1963; Bergstra and Bengio, 2012; Bergstra et al., 2013) or Sequential Model-Based Global Optimization (SMBO), often referred to as Bayesian optimization (Mockus, 1989; Jones et al., 1998; Forrester et al., 2008; Brochu et al., 2010; Snoek et al., 2012). Grid-search is a naive discretization of Problem (2). It consists in evaluating the outer function $\mathcal{L}$ on a grid of hyperparameters, solving one inner optimization Problem (1) for each λ in the grid (see Figure 1). For each inner problem solution $\hat{\beta}^{(\lambda)}$, the criterion $\mathcal{C}(\hat{\beta}^{(\lambda)})$ is evaluated, and the model achieving the lowest value is selected. Random-search has a similar flavor, but one randomly selects where the criterion must be evaluated. Finally, SMBO models the objective function $\mathcal{L}$ via a function amenable to uncertainty estimates on its predictions such as a Gaussian process. Hyperparameter values are chosen iteratively to maximize a function such as the expected improvement as described, *e.g.,* by Bergstra et al. (2011). However, these zero-order methods share a common drawback: they scale exponentially with the dimension of the search space (Nesterov, 2004, Sec. 1.1.2).

When the hyperparameter space is continuous and the regularization path $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is well-defined and almost everywhere differentiable, first-order optimization methods are well suited to solve the bilevel optimization Problem (2). Using the chain rule, the gradient of

1. For a linear model $y = X\beta + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, the degree of freedom (dof, Efron 1986) is defined as $\text{dof}(\beta) = \sum_{i=1}^{n} \text{cov}(y_i, (X\beta)_i)/\sigma^2$.
2. The smoothed Hinge loss is given by $\ell(x) = \frac{1}{2} - x$ if $x \leq 0$, $\frac{1}{2}(1-x)^2$ if $0 \leq x \leq 1$ and 0 else.

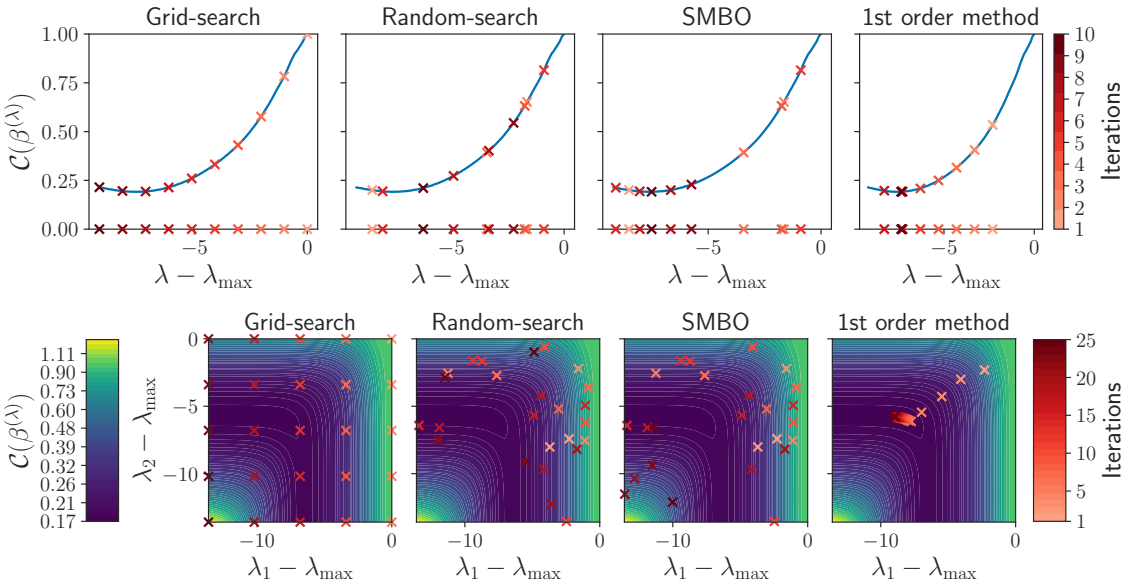

Figure 1: **5-fold cross-validation error** $\mathcal{C}(\beta^{(\lambda)})$: (top) Lasso CV error with respect to λ for multiple hyperparameter optimization methods on the *real-sim* dataset, and (bottom) elastic net CV error with respect to λ_1 and λ_2 on the *rcv1* dataset. Crosses represent the 10 (top) or 25 (bottom) first error evaluations for each method.

$\mathcal{L}$ with respect to λ, also referred to as the *hypergradient*, evaluates to

$$\nabla_\lambda \mathcal{L}(\lambda) = \hat{\mathcal{J}}_{(\lambda)}^\top \nabla \mathcal{C}(\hat{\beta}^{(\lambda)}) \ , \tag{3}$$

with $\hat{\mathcal{J}}_{(\lambda)} \in \mathbb{R}^{p \times r}$ the *Jacobian* of the function $\lambda \mapsto \hat{\beta}^{(\lambda)}$,

$$\hat{\mathcal{J}}_{(\lambda)} \triangleq \begin{pmatrix} \frac{\partial \hat{\beta}_1^{(\lambda)}}{\partial \lambda_1} & \dots & \frac{\partial \hat{\beta}_1^{(\lambda)}}{\partial \lambda_r} \\ \vdots & \dots & \vdots \\ \frac{\partial \hat{\beta}_p^{(\lambda)}}{\partial \lambda_1} & \dots & \frac{\partial \hat{\beta}_p^{(\lambda)}}{\partial \lambda_r} \end{pmatrix} . \tag{4}$$

An important challenge of applying first-order methods to solve Problem (2) is evaluating the hypergradient in Equation (3). There are three main algorithms to compute the hypergradient $\nabla_\lambda \mathcal{L}(\lambda)$: implicit differentiation (Larsen et al., 1996; Bengio, 2000) and automatic differentiation using the reverse-mode (Linnainmaa, 1970; LeCun et al., 1998) or the forward-mode (Wengert, 1964; Deledalle et al., 2014; Franceschi et al., 2017). As illustrated in Figure 1, once the hypergradient in Equation (3) has been computed, one can solve Problem (2) with first-order schemes, *e.g.,* gradient descent.

Contributions. We are interested in tackling the bilevel optimization Problem (2), with a non-smooth inner optimization Problem (1). More precisely,

- We show that classical algorithms used to compute hypergradients for smooth inner problem have theoretically grounded non-smooth counterparts. We provide in Theorem 9 an implicit differentiation formula for non-smooth optimization problems. We obtain in Theorem 13, for the first time in the non-smooth case, error bounds with respect to the hypergradient when the inner problem and the linear system involved are only solved approximately. We obtain in Theorem 12 convergence rates on the hypergradient for iterative differentiation of non-smooth optimization problems.

- Based on the former contributions we propose an algorithm to tackle Problem (2). We develop an efficient implicit differentiation algorithm to compute the hypergradient in Equation (3), leveraging the sparsity of the Jacobian and enabling the use of state-of-the-art solvers (Algorithm 5). We combine in Algorithm 6 this fast hypergradient computation with a gradient descent scheme to solve Problem (2).

- We provide extensive experiments on diverse datasets and estimators (Section 4). We first show that implicit differentiation significantly outperforms other hypergradient methods (Section 4.1). Then, leveraging sparsity, we illustrate computational benefits of first-order optimization with respect to zero-order techniques for solving Problem (2) on Lasso, elastic net and multiclass logistic regression (Section 4.2).

- We release our implementation as a high-quality, documented and tested Python package: `https://github.com/qb3/sparse-ho`.

General notation. We write $\|\cdot\|$ the Euclidean norm on vectors. For a set S, we denote by S^c its complement. We denote $[p] = \{1, \dots, p\}$. We denote by $(e_j)_{j=1}^p$ the vectors of the canonical basis of $\mathbb{R}^p$. We denote the coordinate-wise multiplication of two vectors u and v by $u \odot v$, and by $u \odot M$ the row-wise multiplication between a vector and a matrix. The i-th line of the matrix M is $M_{i:}$ and its j-th column is $M_{:j}$. The spectral radius of a matrix $M \in \mathbb{R}^{n \times n}$ is denoted $\rho(M) = \max_i |s_i|$ where $s_1, \dots, s_n$ are the eigenvalues of M. For a matrix M, we write that $M \succ 0$ if M is positive definite. The regularization parameter, possibly multivariate, is denoted by $\lambda = (\lambda_1, \dots, \lambda_r)^\top \in \mathbb{R}^r$. 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 $\hat{\beta}^{(\lambda)}$ with respect to λ.

Convex analysis. For a convex function $h : \mathbb{R}^p \to \mathbb{R}$, the proximal operator of h is defined, for any $x \in \mathbb{R}^p$, as: $\text{prox}_h(x) = \arg\min_{y \in \mathbb{R}^p} \frac{1}{2}\|x - y\|^2 + h(y)$. The subdifferential of h at x is denoted $\partial h(x) = \left\{u \in \mathbb{R}^p : \forall z \in \mathbb{R}^p, h(z) \geq h(x) + u^\top (z - x)\right\}$. A function is said to be *smooth* if it has Lipschitz gradients. Let f be a L-smooth function. Lipschitz constants of the functions $\nabla_j f$ are denoted by L_j; hence for all $x \in \mathbb{R}^p$, $h \in \mathbb{R}$:

$$|\nabla_j f(x + h e_j) - \nabla_j f(x)| \leq L_j |h| \ .$$

Table 3: Partial derivatives of proximal operators used.

$g_j(\beta_j, \lambda)$	$\operatorname{prox}_{g_j(\cdot,\lambda)}(z_j)$	$\partial_z \operatorname{prox}_{g_j(\cdot,\lambda)}(z_j)$	$\partial_\lambda \operatorname{prox}_{g_j(\cdot,\lambda)}(z_j)$
$e^\lambda \beta_j^2/2$	$z_j/(1+e^\lambda)$	$1/(1+e^\lambda)$	$-z_j e^\lambda/(1+e^\lambda)^2$
$e^\lambda \lvert\beta_j\rvert$	$\operatorname{ST}(z_j, e^\lambda)$	$\lvert\operatorname{sign}(\operatorname{ST}(z_j, e^\lambda))\rvert$	$-e^\lambda \operatorname{sign}(\operatorname{ST}(z_j, e^\lambda))$
$e^{\lambda_1}\lvert\beta_j\rvert + \frac{1}{2}e^{\lambda_2}\beta_j^2$	$\frac{\operatorname{ST}(z_j, e^{\lambda_1})}{1+e^{\lambda_2}}$	$\frac{\lvert\operatorname{sign}(\operatorname{ST}(z_j, e^{\lambda_1}))\rvert}{1+e^{\lambda_2}}$	$\left(\frac{-e^{\lambda_1}\operatorname{sign}(\operatorname{ST}(z_j, e^{\lambda_1}))}{1+e^{\lambda_2}}, \frac{-\operatorname{ST}(z_j, e^{\lambda_1})e^{\lambda_2}}{(1+e^{\lambda_2})^2}\right)$
$\iota_{[0,e^\lambda]}(\beta_j)$	$\max(0, \min(z_j, e^\lambda))$	$\mathbb{1}_{]0,e^\lambda[}(z_j)$	$e^\lambda \mathbb{1}_{z_j > e^\lambda}$

For a function f, its gradient restricted to the indices in a set S is denoted $\nabla_S f$. For a set $\Xi \subset \mathbb{R}^p$, its relative interior is noted $\operatorname{ri}(\Xi)$, and its indicator function is defined for any $x \in \mathbb{R}^p$ by $\iota_\Xi(x) = 0$ if $x \in \Xi$ and $+\infty$ otherwise. A function $h : \mathbb{R} \to \mathbb{R} \cup \{+\infty\}$ is said to be proper if $\operatorname{dom}(h) = \{x \in \mathbb{R} : h(x) < +\infty\} \neq \emptyset)$, and closed if for any $\alpha \in \mathbb{R}$, the sublevel set $\{x \in \operatorname{dom}(h) : h(x) \leq \alpha\}$ is a closed set.

For a function $\psi : \mathbb{R}^p \times \mathbb{R}^r \mapsto \mathbb{R}^p$, we denote $\partial_z \psi$ the weak Jacobian with respect to the first variable and $\partial_\lambda \psi$ the weak Jacobian with respect to the second variable. The proximal operator of $g(\cdot, \lambda)$ can be seen as such a function ψ of β and λ (see Table 1 for examples):

$$\begin{aligned} \mathbb{R}^p \times \mathbb{R}^r &\to \mathbb{R}^p \\ (z, \lambda) &\mapsto \operatorname{prox}_{g(\cdot,\lambda)}(z) = \psi(z, \lambda) \ . \end{aligned}$$

In this case we denote $\partial_z \operatorname{prox}_{g(\cdot,\lambda)} \triangleq \partial_z \psi$ and $\partial_\lambda \operatorname{prox}_{g(\cdot,\lambda)} \triangleq \partial_\lambda \psi$. Since we consider only separable penalties $g(\cdot, \lambda)$, $\partial_z \operatorname{prox}_{g(\cdot,\lambda)}$ is a diagonal matrix, so to make notation lighter, we write $\partial_z \operatorname{prox}_{g(\cdot,\lambda)}$ for its diagonal. We thus have

$$\begin{aligned} \partial_z \operatorname{prox}_{g(\cdot,\lambda)} = (\partial_z \operatorname{prox}_{g_j(\cdot,\lambda)})_{j \in [p]} \in \mathbb{R}^p \quad \text{(by separability of } g) \\ \partial_\lambda \operatorname{prox}_{g(\cdot,\lambda)} \in \mathbb{R}^{p \times r} \ . \end{aligned}$$

Explicit partial derivatives formulas for usual proximal operators can be found in Table 3.

2. Related work

The main challenge to evaluate the hypergradient $\nabla_\lambda \mathcal{L}(\lambda)$ is the computation of the Jacobian $\mathcal{J}_{(\lambda)}$. We first focus on the case where $\Phi(\cdot, \lambda)$ is convex and smooth for any λ.

Implicit differentiation. We recall how the *implicit differentiation*[3] formula of the gradient $\nabla_\lambda \mathcal{L}(\lambda)$ is obtained for smooth inner optimization problems. We will provide a generalization to non-smooth optimization problems in Section 3.2.

3. Note that *implicit* refers to the implicit function theorem, but leads to an *explicit* formula for the gradient.

Theorem 1 (Bengio 2000) *Let $\hat{\beta}^{(\lambda)} \in \arg\min_{\beta \in \mathbb{R}^p} \Phi(\beta, \lambda)$ be a solution of Problem (1). Assume that for all $\lambda > 0$, $\Phi(\cdot, \lambda)$ is a convex smooth function, $\nabla^2_\beta \Phi(\hat{\beta}^{(\lambda)}, \lambda) \succ 0$, and that for all $\beta \in \mathbb{R}^p$, $\Phi(\beta, \cdot)$ is differentiable over $]0, +\infty[$. Then the hypergradient $\nabla_\lambda \mathcal{L}(\lambda)$ reads:*

$$\underbrace{\nabla_\lambda \mathcal{L}(\lambda)}_{\in \mathbb{R}^r} = \underbrace{-\nabla^2_{\beta,\lambda} \Phi(\hat{\beta}^{(\lambda)}, \lambda)}_{\in \mathbb{R}^{r \times p}} \underbrace{\left(\nabla^2_\beta \Phi(\hat{\beta}^{(\lambda)}, \lambda)\right)^{-1}}_{\in \mathbb{R}^{p \times p}} \underbrace{\nabla \mathcal{C}(\hat{\beta}^{(\lambda)})}_{\in \mathbb{R}^p} \ . \tag{5}$$

Proof For a smooth convex function $\beta \mapsto \Phi(\beta, \lambda)$ the first-order condition writes:

$$\nabla_\beta \Phi(\hat{\beta}^{(\lambda)}, \lambda) = 0 \ , \tag{6}$$

for any $\hat{\beta}^{(\lambda)}$ solution of the inner problem. Moreover, if $\lambda \mapsto \nabla_\beta \Phi(\hat{\beta}^{(\lambda)}, \lambda)$ is differentiable, differentiating Equation (6) with respect to λ leads to:

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

The Jacobian $\hat{\mathcal{J}}^\top_{(\lambda)}$ is computed by solving the following linear system:

$$\hat{\mathcal{J}}^\top_{(\lambda)} = - \underbrace{\nabla^2_{\beta,\lambda} \Phi(\hat{\beta}^{(\lambda)}, \lambda)}_{\in \mathbb{R}^{r \times p}} \underbrace{\left(\nabla^2_\beta \Phi(\hat{\beta}^{(\lambda)}, \lambda)\right)^{-1}}_{\in \mathbb{R}^{p \times p}} \ . \tag{8}$$

Plugging Equation (8) into Equation (3) yields the desired result. ■

The computation of the gradient via implicit differentiation (Equation (5)) involves the resolution of a $p \times p$ linear system (Bengio, 2000, Sec. 4). This potentially large linear system can be solved using different algorithms such as conjugate gradient (Hestenes and Stiefel 1952, as in Pedregosa 2016) or fixed point methods (Lions and Mercier 1979; Tseng and Yun 2009, as in Grazzi et al. 2020). Implicit differentiation has been used for model selection of multiple estimators with smooth regularization term: kernel-based models (Chapelle et al., 2002; Seeger, 2008), weighted Ridge estimator (Foo et al., 2008), neural networks (Lorraine et al., 2019) or meta-learning (Franceschi et al., 2018; Rajeswaran et al., 2019). In addition to hyperparameter selection, it has been applied successfully in natural language processing (Bai et al., 2019) and computer vision (Bai et al., 2020).

Problem (1) is typically solved using iterative solvers. In practice, the number of iterations is limited to reduce computation time, and also since very precise solutions are generally not necessary for machine learning tasks. Thus, Equation (6) is not exactly satisfied at machine precision, and consequently the linear system to solve Equation (5) does not lead to the exact gradient $\nabla_\lambda \mathcal{L}(\lambda)$, see Ablin et al. (2020) for quantitative convergence results. However, Pedregosa (2016) showed that one can resort to *approximate gradients* when the inner problem is smooth, justifying that implicit differentiation can be applied using an approximation of $\hat{\beta}$. Interestingly, this approximation scheme was shown to yield significant practical speedups when solving Problem (2), while preserving theoretical properties of convergence toward the optimum.

Iterative differentiation. Iterative differentiation computes the gradient $\nabla_\lambda \mathcal{L}(\lambda)$ by differentiating through the iterates of the algorithm used to solve Problem (1). Iterative differentiation can be applied using the forward-mode (Wengert 1964; Deledalle et al. 2014; Franceschi et al. 2017) or the reverse-mode (Linnainmaa 1970; LeCun et al. 1998; Domke 2012). Both rely on the chain rule, the gradient being decomposed as a large product of matrices, computed either in a forward or backward way. Note that forward and reverse modes are algorithm-dependent: in this section we illustrate iterative differentiation for proximal gradient descent (PGD, Lions and Mercier 1979; Combettes and Wajs 2005), using the forward-mode (Algorithm 1), and the reverse-mode (Algorithm 2).

The most popular method in automatic differentiation is the reverse-mode, a cornerstone of deep learning (Goodfellow et al., 2016, Chap. 8). Iterative differentiation for hyperparameter optimization can be traced back to Domke (2012), who derived (for smooth loss functions) a reverse-mode with gradient descent, heavy ball and L-BFGS algorithms. It first computes the solution of the optimization Problem (1) using an iterative solver, but requires storing the iterates along the computation for a backward evaluation of the hypergradient (Algorithm 2). Alternatively, the forward-mode computes jointly the solution along with the gradient $\nabla_\lambda \mathcal{L}(\lambda)$. It is memory efficient (no iterates storage) but more computationally expensive when the number of hyperparameters (r) is large; see Baydin et al. (2018) for a survey.

Resolution of the bilevel Problem (2). From a theoretical point of view, solving Problem (2) using gradient-based methods is also challenging, and results in the literature are quite scarce. Kunisch and Pock (2013) studied the convergence of a semi-Newton algorithm where both the outer and inner problems are smooth. Franceschi et al. (2018) gave similar results with weaker assumptions to unify hyperparameter optimization and meta-learning with a bilevel point of view. They required the inner problem to have a unique solution for all $\lambda > 0$ but do not have second-order assumptions on Φ. Recent results (Ghadimi and Wang, 2018; Ji et al., 2020; Mehmood and Ochs, 2021) have provided quantitative convergence toward a global solution of Problem (2), but under global joint convexity assumption and exact knowledge of the gradient Lipschitz constant.

3. Bilevel optimization with non-smooth inner problems

We recalled above how to compute hypergradients when the inner optimization problem is smooth. In this section we tackle the bilevel optimization Problem (2) with non-smooth inner optimization Problem (1). Handling non-smooth inner problems requires specific tools detailed in Section 3.1. We then show how to compute gradients with non-smooth inner problems using implicit differentiation (Section 3.2) or iterative differentiation (Section 3.3). In Section 3.4 we tackle the problem of approximate gradient for a non-smooth inner optimization problem. Finally, we propose in Section 3.6 an algorithm to solve the bilevel optimization Problem (2).

Algorithm 1 FORWARD-MODE PGD

input : $\lambda \in \mathbb{R}^r, \gamma > 0, n_{\text{iter}} \in \mathbb{N}, \beta^{(0)} \in \mathbb{R}^p, \mathcal{J}^{(0)} \in \mathbb{R}^{p \times r}$

// jointly compute coef. & Jacobian

for $k = 1, \dots, n_{\text{iter}}$ **do**

  // update the regression coefficients

  $z^{(k)} = \beta^{(k-1)} - \gamma \nabla f(\beta^{(k-1)})$ // GD step

  $\mathrm{d}z^{(k)} = \mathcal{J}^{(k-1)} - \gamma \nabla^2 f(\beta^{(k-1)}) \mathcal{J}^{(k-1)}$

  $\beta^{(k)} = \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)})$ // prox. step

  // update the Jacobian

  $\mathcal{J}^{(k)} = \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)}) \odot \mathrm{d}z^{(k)}$

  $\mathcal{J}^{(k)} \mathrel{+}= \partial_\lambda \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)})$ // $\mathcal{O}(pr)$

$v = \nabla \mathcal{C}(\beta^{n_{\text{iter}}})$

return $\beta^{n_{\text{iter}}}, {\mathcal{J}^{n_{\text{iter}}}}^\top v$

Algorithm 2 REVERSE-MODE PGD

input : $\lambda \in \mathbb{R}^r, \gamma > 0, n_{\text{iter}} \in \mathbb{N}, \beta^{(0)} \in \mathbb{R}^p$

// computation of $\hat{\beta}$

for $k = 1, \dots, n_{\text{iter}}$ **do**

  $z^{(k)} = \beta^{(k-1)} - \gamma \nabla f(\beta^{(k-1)})$ // GD step

  $\beta^{(k)} = \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)})$ // prox. step

// backward computation of the gradient g

$v = \nabla \mathcal{C}(\beta^{(n_{\text{iter}})})$, $h = 0_{\mathbb{R}^r}$

for $k = n_{\text{iter}}, n_{\text{iter}} - 1, \dots, 1$ **do**

  $h \mathrel{+}= v^\top \partial_\lambda \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)})$ // $\mathcal{O}(pr)$

  $v \leftarrow \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)}(z^{(k)}) \odot v$ // $\mathcal{O}(p)$

  $v \leftarrow (\mathrm{Id} - \gamma \nabla^2 f(\beta^{(k)}))v$ // $\mathcal{O}(np)$

return $\beta^{n_{\text{iter}}}, h$

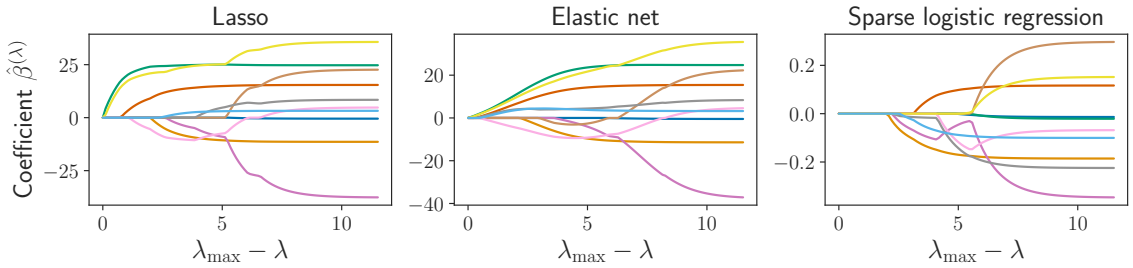

Figure 2: **Regularization paths** (coefficient values as a function of λ), on the *diabetes* and *breast cancer* datasets for the Lasso, the elastic net and sparse logistic regression. This illustrates the weak differentiability of the paths. We used *diabetes* for the Lasso and the elastic net, and the 10 first features of *breast cancer* for the sparse logistic regression.

3.1 Theoretical framework

Differentiability of the regularization path. Before applying first-order methods to tackle Problem (2), one must ensure that the regularization path $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is almost everywhere differentiable (as in Figure 2). This is the case for the Lasso (Mairal and Yu, 2012) and the SVM (Hastie et al., 2004; Rosset and Zhu, 2007) since solution paths are piecewise differentiable (see Figure 2). Results for nonquadratic datafitting terms are scarcer: Friedman et al. (2010) address the practical resolution of sparse logistic regression, but stay evasive regarding the differentiability of the regularization path. In the general case for problems of the form Problem (1), we believe it is an open question and leave it for future work.

Differentiability of proximal operators. The key point to obtain an implicit differentiation formula for non-smooth inner problems is to differentiate the fixed point equation of proximal gradient descent. From a theoretical point of view, ensuring this differentiability at the optimum is non-trivial: Poliquin and Rockafellar (1996, Thm. 3.8) showed that under a *twice epi-differentiability* condition the proximal operator is differentiable at optimum. For the convergence of forward and reverse modes in the non-smooth case, one has to ensure that, after enough iterations, the updates of the algorithms become differentiable. Deledalle et al. (2014) justified (weak) differentiability of proximal operators as they are non-expansive. However this may not be a sufficient condition, see Bolte and Pauwels (2020a,b). In our case, we show differentiability after *support identification* of the algorithms: active constraints are identified after a finite number of iterations by proximal gradient descent (Liang et al., 2014; Vaiter et al., 2018) and proximal coordinate descent, see Nutini (2018, Sec. 6.2) or Klopfenstein et al. (2020). Once these constraints have been identified convergence is linear towards the Jacobian (see Theorem 12 and Figures 3, 10 and 11).

For the rest of this paper, we consider the bilevel optimization Problem (2) with the following assumptions on the inner Problem (1).

Assumption 2 (Smoothness) *The function $f : \mathbb{R}^p \to \mathbb{R}$ is a convex, differentiable function, with a L-Lipschitz gradient.*

Assumption 3 (Proper, closed, convex) *For all $\lambda \in \mathbb{R}^r$, for any $j \in [p]$, the function $g_j(\cdot, \lambda) : \mathbb{R} \to \mathbb{R}$ is proper, closed and convex.*

Assumption 4 (Non-degeneracy) *The problem admits at least one solution:*

$$\arg\min_{\beta \in \mathbb{R}^p} \Phi(\beta, \lambda) \neq \emptyset \ ,$$

and, for any $\hat{\beta}$ solution of Problem (1), we have

$$-\nabla f(\hat{\beta}) \in \operatorname{ri}\left(\partial_\beta g(\hat{\beta}, \lambda)\right) \ .$$

To be able to extend iterative and implicit differentiation to the non-smooth case, we need to introduce the notion of generalized support.

Definition 5 (Generalized support, Nutini et al. 2019, Def. 1) *For a solution $\hat{\beta} \in \arg\min_{\beta \in \mathbb{R}^p} \Phi(\beta, \lambda)$, its* generalized support *$\hat{S} \subseteq [p]$ is the set of indices $j \in [p]$ such that g_j is differentiable at $\hat{\beta}_j$:*

$$\hat{S} \triangleq \{j \in [p] : \partial_\beta g_j(\hat{\beta}_j, \lambda) \text{ is a singleton}\} \ .$$

*An iterative algorithm is said to achieve **finite support identification** if its iterates $\beta^{(k)}$ converge to $\hat{\beta}$, and there exists $K \geq 0$ such that for all $j \notin \hat{S}$, for all $k \geq K, \beta_j^{(k)} = \hat{\beta}_j$.*

Examples. For the ℓ_1 norm (promoting sparsity), $g_j(\hat{\beta}_j, \lambda) = e^\lambda |\hat{\beta}_j|$, the generalized support is $\hat{S} \triangleq \{j \in [p] : \hat{\beta}_j \neq 0\}$. This set corresponds to the indices of the non-zero coefficients, which is the usual support definition. For the SVM estimator, $g_j(\hat{\beta}_j, \lambda) = \iota_{[0,e^\lambda]}(\hat{\beta}_j)$. This function is non-differentiable at 0 and at e^λ. The generalized support for the SVM estimator then corresponds to the set of indices such that $\hat{\beta}_j \in]0, e^\lambda[$.

Finally, to prove local linear convergence of the Jacobian we assume regularity and strong convexity on the generalized support.

Assumption 6 (Locally $\mathcal{C}^2$ and $\mathcal{C}^3$) *The map $\beta \mapsto f(\beta)$ is locally $\mathcal{C}^3$ around $\hat{\beta}$. For all $\lambda \in \mathbb{R}^r$, for all $j \in \hat{S}$ the map $g_j(\cdot, \lambda)$ is locally $\mathcal{C}^2$ around $\hat{\beta}_j$.*

Assumption 7 (Restricted injectivity) *Let $\hat{\beta}$ be a solution of Problem (1) and $\hat{S}$ its generalized support. The solution $\hat{\beta}$ satisfies the following restricted injectivity condition:*

$$\nabla^2_{\hat{S},\hat{S}} f(\hat{\beta}) \succ 0 \ .$$

Assumptions 2 and 3 are classical to ensure inner problems can be solved using proximal algorithms. Assumption 4 can be seen as a generalization of constraint qualifications (Hare and Lewis, 2007, Sec. 1) and is crucial to ensure *support identification*. Assumptions 6 and 7 are classical for the analysis (Liang et al., 2017) and sufficient to derive rates of convergence for the Jacobian of the inner problem once the generalized support has been identified.

The next lemma guarantees uniqueness of Problem (1) under Assumptions 4 and 7.

Lemma 8 (Liang et al. 2017, Prop. 4.1) *Assume that there exists a neighborhood Λ of λ such that Assumptions 4 and 7 are satisfied for every $\lambda \in \Lambda$. Then for every $\lambda \in \Lambda$, Problem (1) has a unique solution, and the map $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is well-defined on Λ.*

We first show how implicit and iterative differentiation can be used with a non-smooth inner problem. Peyré and Fadili (2011) proposed to smooth the inner optimization problem, Ochs et al. (2015); Frecon et al. (2018) relied on the forward-mode combined with Bregman iterations to get differentiable steps. For non-smooth optimization problems, implicit differentiation has been considered for (constrained) convex optimization problems (Gould et al., 2016; Amos and Kolter, 2017; Agrawal et al., 2019), Lasso-type problems (Mairal et al., 2012; Bertrand et al., 2020), total variation penalties (Cherkaoui et al., 2020) and generalized to strongly monotone operators (Winston and Kolter, 2020).

3.2 Hypergradient computation: implicit differentiation

The exact proof of Theorem 1 cannot be applied when $\beta \mapsto \Phi(\beta, \lambda)$ is non-smooth, as Equations (6) and (7) no longer hold. Nevertheless, instead of the optimality condition of smooth optimization, Equation (6), one can leverage the fixed point iteration of proximal gradient descent, which we will see in Equation (11). The main theoretical challenge is

to show the differentiability of the function $\beta \mapsto \operatorname{prox}_{\gamma g}(\beta - \gamma \nabla f(\beta))$. Besides, taking advantage of the generalized sparsity of the regression coefficients $\hat{\beta}^{(\lambda)}$, one can show that the Jacobian $\hat{\mathcal{J}}$ is row-sparse, leading to substantial computational benefits when computing the hypergradient $\nabla_\lambda \mathcal{L}(\lambda))$ for Problem (1),

Theorem 9 (Non-smooth implicit formula) *Suppose Assumptions 2, 3 and 6 hold. Let $0 < \gamma \leq 1/L$, where L is the Lipschitz constant of ∇f. Let $\lambda \in \mathbb{R}^r$, Λ be a neighborhood of λ, and $\Gamma^\Lambda \triangleq \left\{ \hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) \,:\, \lambda \in \Lambda \right\}$. In addition,*

(H1) Suppose Assumptions 4 and 7 hold on Λ.

(H2) Suppose $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is continuously differentiable on Λ.

(H3) Suppose for all $z \in \Gamma^\Lambda$, $\lambda \mapsto \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z)$ is continuously differentiable on Λ.

(H4) Suppose $\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}$ and $\partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}$ are Lipschitz continuous on $\Gamma^\Lambda \times \Lambda$.

Let $\hat{\beta} \triangleq \hat{\beta}^{(\lambda)}$ be the solution of Problem (1), $\hat{S}$ its generalized support of cardinality $\hat{s}$. Then the Jacobian $\hat{\mathcal{J}}$ of the inner Problem (1) is given by the following formula,

$\hat{z} = \hat{\beta} - \gamma \nabla f(\hat{\beta})$, *and* $A \triangleq \operatorname{Id}_{\hat{s}} - \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}} \odot \left(\operatorname{Id}_{\hat{s}} - \gamma \nabla^2_{\hat{S},\hat{S}} f(\hat{\beta}) \right)$*:*

$$\hat{\mathcal{J}}_{\hat{S}^c:} = \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}^c} \quad , \tag{9}$$

$$\hat{\mathcal{J}}_{\hat{S}:} = A^{-1} \left(\partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}} - \gamma \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}} \odot \nabla^2_{\hat{S},\hat{S}^c} f(\hat{\beta}) \hat{\mathcal{J}}_{\hat{S}^c} \right) \quad . \tag{10}$$

Proof According to Lemma 8, Assumptions 4 and 7 ensure Problem (1) has a unique minimizer and $\lambda \mapsto \hat{\beta}^{(\lambda)}$ is well-defined on Λ. We consider the proximal gradient descent fixed point equation:

$$\hat{\beta}^{(\lambda)} = \operatorname{prox}_{\gamma g_{(\cdot,\lambda)}} \left(\hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) \right) \quad . \tag{11}$$

Together with the conclusion of Lemma 8, Assumptions 2 and 6, and given (H2), (H3) and (H4), we have that $\lambda \mapsto \psi \left(\beta^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}), \lambda \right) \triangleq \operatorname{prox}_{\gamma g(\cdot,\lambda)} \left(\hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) \right)$ is differentiable at λ. One can thus differentiate Equation (11) with respect to λ, which leads to:

$$\hat{\mathcal{J}} = \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z}) \odot \left(\operatorname{Id} - \gamma \nabla^2 f(\hat{\beta}) \right) \hat{\mathcal{J}} + \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z}) \quad , \tag{12}$$

with $\hat{z} = \hat{\beta} - \gamma \nabla f(\hat{\beta})$. In addition to $0 < \gamma < 1/L \leq 1/L_j$, the separability of g and Assumptions 2 to 4 and 6 ensure (see Lemma 18) that for any $j \in \hat{S}^c$,

$$\partial_z \operatorname{prox}_{\gamma g_j(\cdot,\lambda)} \left(\hat{\beta}_j - \gamma \nabla_j f(\hat{\beta}) \right) = 0 \quad . \tag{13}$$

Plugging Equation (13) into Equation (12) ensures Equation (9) for all $j \in \hat{S}^c$:

$$\hat{\mathcal{J}}_{j:} = \partial_\lambda \operatorname{prox}_{\gamma g_j(\cdot, \lambda)} \left(\hat{\beta}_j - \gamma \nabla_j f(\hat{\beta}) \right) \quad . \tag{14}$$

Plugging Equations (13) and (14) into Equation (12) shows that the Jacobian restricted on the generalized support $\hat{S}$ satisfies the following linear system:

$$\left(\mathrm{Id}_{\hat{s}} - \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)} (\hat{z})_{\hat{S}} \odot \left(\mathrm{Id}_{\hat{s}} - \gamma \nabla^2_{\hat{S}, \hat{S}} f(\hat{\beta}) \right) \right) \hat{\mathcal{J}}_{\hat{S}:} = \\ -\gamma \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)} (\hat{z})_{\hat{S}} \odot \nabla^2_{\hat{S}, \hat{S}^c} f(\hat{\beta}) \hat{\mathcal{J}}_{\hat{S}^c:} + \partial_\lambda \operatorname{prox}_g (\hat{z})_{\hat{S}:} \quad .$$

Since $0 < \gamma \leq 1/L$,

$$\| \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)} (\hat{z})_{\hat{S}} \odot (\mathrm{Id}_{\hat{s}} - \gamma \nabla^2_{\hat{S}, \hat{S}} f(\hat{\beta})) \|_2 \leq \| \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)} (\hat{z})_{\hat{S}} \| \cdot \| \mathrm{Id}_{\hat{s}} - \gamma \nabla^2_{\hat{S}, \hat{S}} f(\hat{\beta}) \|_2 \\ < 1 \quad . \tag{15}$$

Since Equation (15) holds, $A \triangleq \mathrm{Id}_{\hat{s}} - \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)} (\hat{z})_{\hat{S}} \odot (\mathrm{Id}_{\hat{s}} - \gamma \nabla^2_{\hat{S}, \hat{S}} f(\hat{\beta}))$ is invertible, which leads to Equation (10). ■

Remark 10 *In the smooth case a $p \times p$ linear system is needed to compute the Jacobian in Equation (8). For non-smooth problems this is reduced to an $\hat{s} \times \hat{s}$ linear system ($\hat{s} \leq p$ being the size of the generalized support,* e.g., *the number of non-zero coefficients for the Lasso). This leads to significant speedups in practice, especially for very sparse vector $\hat{\beta}^{(\lambda)}$.*

Remark 11 *To obtain Theorem 9 we differentiated the fixed point equation of proximal gradient descent, though one could differentiate other fixed point equations (such as the one from proximal coordinate descent). The value of the Jacobian $\hat{\mathcal{J}}$ obtained with different fixed point equations would be the same, yet the associated systems could have different numerical stability properties. We leave this analysis to future work.*

3.3 Hypergradient computation: iterative differentiation

Instead of implicit differentiation, it is also possible to use iterative differentiation on proximal solvers. In section Section 2 we presented forward and reverse modes differentiation of proximal gradient descent (Algorithms 1 and 2). In this section we study the iterative differentiation of proximal coordinate descent (Algorithms 3 and 4). To instantiate algorithms easily on problems such as the Lasso, partial derivatives of usual proximal operators can be found in Table 3.

Algorithm 3 FORWARD-MODE PCD

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}^r, n_{\text{iter}} \in \mathbb{N}$, $\beta \in \mathbb{R}^p$, $\mathcal{J} \in \mathbb{R}^{p \times r}, \gamma_1, \dots, \gamma_p$

// jointly compute coef. & Jacobian

for $k = 1, \dots, n_{\text{iter}}$ **do**

  for $j = 1, \dots, p$ **do**

    // update the regression coefficients

    $z_j \leftarrow \beta_j - \gamma_j \nabla_j f(\beta)$ // CD step

    $\mathrm{d}z_j \leftarrow \mathcal{J}_{j:} - \gamma_j \nabla^2_{j:} f(\beta) \mathcal{J}$

    $\beta_j \leftarrow \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j)$ // prox. step

    // update the Jacobian

    // diff. with respect to λ

    $\mathcal{J}_{j:} \leftarrow \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j) \mathrm{d}z_j$

    $\mathcal{J}_{j:} \mathrel{+}= \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j)$

  $\beta^{(k)} = \beta$

  $\mathcal{J}^{(k)} = \mathcal{J}$

$v = \nabla C(\beta)$

return $\beta^{n_{\text{iter}}}, \mathcal{J}^\top v$

Algorithm 4 REVERSE-MODE PCD

input : $X \in \mathbb{R}^{n \times p}, y \in \mathbb{R}^n, \lambda \in \mathbb{R}^r, n_{\text{iter}} \in \mathbb{N}$, $\beta \in \mathbb{R}^p, \gamma_1, \dots, \gamma_p$

// compute coef.

for $k = 1, \dots, n_{\text{iter}}$ **do**

  for $j = 1, \dots, p$ **do**

    // update the regression coefficients

    $z_j \leftarrow \beta_j - \gamma_j \nabla_j f(\beta)$ // CD step

    $\beta_j \leftarrow \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j)$ // prox. step

    $\beta^{(k,j)} = \beta; z_j^{(k)} = z_j$ // store iterates

// compute gradient g in a backward way

$v = \nabla C(\beta^{n_{\text{iter}}})$, $h = 0_{\mathbb{R}^r}$

for $k = n_{\text{iter}}, n_{\text{iter}} - 1, \dots, 1$ **do**

  for $j = p, \dots, 1$ **do**

    $h \mathrel{-}= \gamma_j v_j \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}\left(z_j^{(k)}\right)$

    $v_j \mathrel{*}= \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}\left(z_j^{(k)}\right)$

    $v \mathrel{-}= \gamma_j v_j \nabla^2_{j:} f(\beta^{(k,j)})$ // $\mathcal{O}(np)$

return $\beta^{n_{\text{iter}}}, h$

For coordinate descent, the computation of the iterative Jacobian in a forward way involves differentiating the following update:

$$
\begin{aligned}
z_j &\leftarrow \beta_j - \gamma_j \nabla_j f(\beta) \\
\beta_j &\leftarrow \operatorname{prox}_{\gamma_j g_j}\left(\beta_j - \gamma_j \nabla_j f(\beta)\right) \\
\mathcal{J}_{j:} &\leftarrow \underbrace{\partial_z \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j)}_{\in \mathbb{R}} \underbrace{\left(\mathcal{J}_{j:} - \gamma_j \nabla^2_{j:} f(\beta) \mathcal{J}\right)}_{\in \mathbb{R}^p} + \underbrace{\partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot, \lambda)}(z_j)}_{\in \mathbb{R}^p} \quad .
\end{aligned}
$$

We address now the convergence of the iterative Jacobian scheme, a question which remained open in Deledalle et al. (2014, Section 4.1). We show next that the forward-mode converges to the Jacobian in the non-smooth separable setting of this paper. Moreover, we prove that the iterative Jacobian convergence is locally linear after support identification.

Theorem 12 (Local linear convergence of the Jacobian) *Let $0 < \gamma \leq 1/L$. Suppose Assumptions 2, 3 and 6 hold. Let $\lambda \in \mathbb{R}^r$, Λ be a neighborhood of λ, and $\Gamma^\Lambda \triangleq \left\{\hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) : \lambda \in \Lambda\right\}$. In addition, suppose hypotheses (H1) to (H4) from Theorem 9 are satisfied and the sequence $(\beta^{(k)})_{k \in \mathbb{N}}$ generated by Algorithm 1 (respectively by Algorithm 3) converges toward $\hat{\beta}$.*

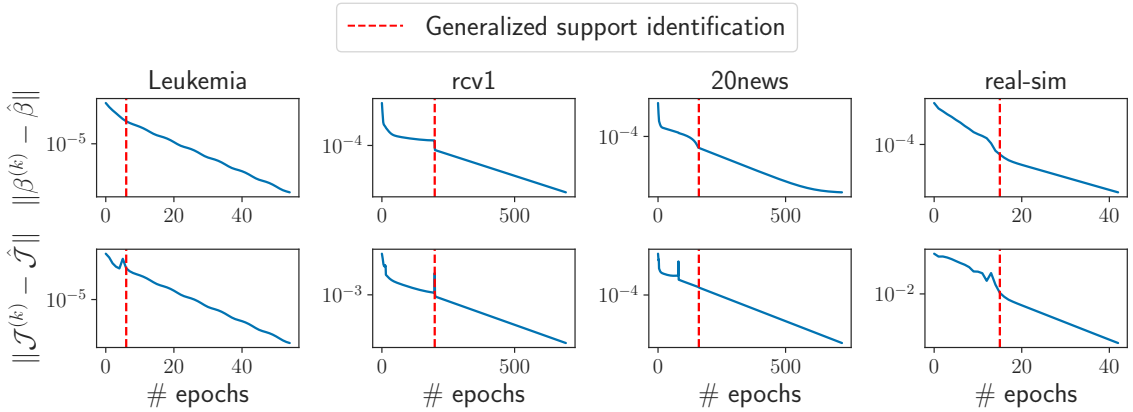

Figure 3: **Local linear convergence of the Jacobian for the SVM.** Distance to optimum for the coefficients β (top) and the Jacobian $\mathcal{J}$ (bottom) of the forward-mode differentiation of proximal coordinate descent (Algorithm 3) on multiple datasets. One epoch corresponds to one pass over the data, *i.e.,* one iteration with proximal gradient descent.

Then, the sequence of Jacobians $(\mathcal{J}^{(k)})_{k\geq 0}$ *generated by the forward-mode differentiation of proximal gradient descent (Algorithm 1) (respectively by forward-mode differentiation of proximal coordinate descent, Algorithm 3) converges locally linearly towards* $\hat{\mathcal{J}}$.

Proof of Theorem 12 can be found in Appendix B.

Comments on Figure 3. We illustrate the results of Theorem 12 on SVM (for the Lasso and sparse logistic regression, see Figures 10 and 11 in Appendix C) for multiple datasets (*leukemia*, *rcv1*, *news20* and *real-sim*[4]). The values of the hyperparameters λ are summarized in Table 6. Regression coefficients $\hat{\beta}^{(\lambda)}$ were computed to machine precision (up to duality gap smaller than 10^{-16}) using a state-of-the-art coordinate descent solver implemented in `Lightning` (Blondel and Pedregosa, 2016). The exact Jacobian was computed via implicit differentiation (Equation (10)). Once these quantities were obtained, we used the forward-mode differentiation of proximal coordinate descent (Algorithm 3) and monitored the distance between the iterates of the regression coefficients $\beta^{(k)}$ and the exact solution $\hat{\beta}$. We also monitored the distance between the iterates of the Jacobian $\mathcal{J}^{(k)}$ and the exact Jacobian $\hat{\mathcal{J}}$. The red vertical dashed line represents the iteration number where support identification happens. Once the support is identified, Figures 3, 10 and 11 illustrate the linear convergence of the Jacobian. However, the behavior of the iterative Jacobian before support identification is more erratic and not even monotone.

4. Data available on the *libsvm* website: `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

3.4 Hypergradient computation with approximate gradients

As mentioned in Section 2, relying on iterative algorithms to solve Problem (1), one only has access to an approximation of $\hat{\beta}^{(\lambda)}$: this may lead to numerical errors when computing the gradient in Theorem 9. Extending the result of Pedregosa (2016, Thm. 1), which states that hypergradients can be computed approximately, we give a stability result for the computation of approximate hypergradients in the case of non-smooth inner problems. For this purpose we need to add several assumptions to the previous framework.

Theorem 13 (Bound on the error of approximate hypergradient) *For $\lambda \in \mathbb{R}^r$, let $\hat{\beta}^{(\lambda)} \in \mathbb{R}^p$ be the exact solution of the inner Problem (1), and $\hat{S}$ its generalized support. Suppose Assumptions 2, 3 and 6 hold. Let Λ be a neighborhood of λ, and $\Gamma^{\Lambda} \triangleq \left\{\hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) \,:\, \lambda \in \Lambda\right\}$. Suppose hypotheses (H1) to (H4) from Theorem 9 are satisfied. In addition suppose*

(H5) The application $\beta \mapsto \nabla^2 f(\beta)$ is Lipschitz continuous.

(H6) The criterion $\beta \mapsto \nabla \mathcal{C}(\beta)$ is Lipschitz continuous.

(H7) Both optimization problems in Algorithm 5 are solved up to precision ϵ with support identification: $\|\beta^{(\lambda)} - \hat{\beta}^{(\lambda)}\| \leq \epsilon$, $A^{\top}$ is invertible, and $\|A^{-1\top} \nabla_{\hat{S}} \mathcal{C}(\beta^{(\lambda)}) - v\| \leq \epsilon$.

Then the error on the approximate hypergradient h returned by Algorithm 5 is of the order of magnitude of the error ϵ on $\beta^{(\lambda)}$ and v:

$$\|\nabla \mathcal{L}(\lambda) - h\| = \mathcal{O}(\epsilon) \ .$$

Proof of Theorem 13 can be found in Appendix B.1. Following the analysis of Pedregosa (2016), two sources of approximation errors arise when computing the hypergradient: one from the inexact computation of $\hat{\beta}$, and another from the approximate resolution of the linear system. Theorem 13 states that if the inner optimization problem and the linear system are solved up to precision ϵ, *i.e.*, $\|\hat{\beta}^{(\lambda)} - \beta^{(\lambda)}\| \leq \epsilon$ and $\|A^{-1\top} \nabla_S \mathcal{C}(\beta^{(\lambda)}) - v\| \leq \epsilon$, then the approximation on the hypergradient is also of the order of ϵ.

Remark 14 *The Lipschitz continuity of the proximity operator with respect to λ (H4) is satisfied for usual proximal operators, in particular all the operators in Table 3. The Lipschitz continuity of the Hessian and the criterion, hypotheses (H5) and (H6), are satisfied for usual machine learning loss functions and criteria, such as the least squares and the logistic loss.*

Remark 15 *To simplify the analysis, we used the same tolerance for the resolution of the inner Problem (1) and the resolution of the linear system. Theorem 13 gives intuition on the fact that the inner problem does not need to be solved at high precision to lead to good hypergradients estimation. Note that in practice one does not easily control the distance between the approximate solution and the exact one $\|\beta^{(k)} - \hat{\beta}\|$: most softwares provide a solution up to a given duality gap (sometimes even other criteria), not $\|\beta^{(k)} - \hat{\beta}\|$.*

3.5 Proposed method for hypergradient computation

We now describe our proposed method to compute the hypergradient of Problem (2). In order to take advantage of the sparsity induced by the generalized support, we propose an implicit differentiation algorithm for non-smooth inner problem that can be found in Algorithm 5. First, we compute a solution of the inner Problem (1) using a solver identifying the generalized support (Liang et al., 2014; Klopfenstein et al., 2020). Then, the hypergradient is computed by solving the linear system in Equation (10). This linear system, as mentioned in Section 2, can be solved using multiple algorithms, including conjugate gradient or fixed point methods. Table 4 summarizes the computational complexity in space and time of the described algorithms.

Table 4: Cost in time and space for each method: p is the number of features, n the number of samples, r the number of hyperparameters, and $\hat{s}$ is the size of the generalized support (Definition 5, $\hat{s} \leq p$ and usually $\hat{s} \ll p$). The number of iterations of the inner solver is noted n_{iter}, the number of iterations of the solver of the linear system is noted n_{sys}.

Differentiation	Algorithm	Space	Time
Forward-mode PGD	Algorithm 1	$\mathcal{O}(p\,r)$	$\mathcal{O}(n\,p\,r\,n_{\text{iter}})$
Reverse-mode PGD	Algorithm 2	$\mathcal{O}(p\,n_{\text{iter}})$	$\mathcal{O}(n\,p\,n_{\text{iter}} + n\,p\,n_{\text{iter}})$
Forward-mode PCD	Algorithm 3	$\mathcal{O}(p\,r)$	$\mathcal{O}(n\,p\,r\,n_{\text{iter}})$
Reverse-mode PCD	Algorithm 4	$\mathcal{O}(p\,n_{\text{iter}})$	$\mathcal{O}(n\,p\,n_{\text{iter}} + n\,p^2\,n_{\text{iter}})$
Implicit differentiation	Algorithm 5	$\mathcal{O}(p + \hat{s})$	$\mathcal{O}(n\,p\,n_{\text{iter}} + n\,\hat{s}\,n_{\text{sys}})$

3.6 Resolution of the bilevel optimization Problem (2)

From a practical point of view, once the hypergradient has been computed, first-order methods require the definition of a step size to solve the non-convex Problem (2). As the Lipschitz constant is not available for the outer problem, first-order methods need to rely on other strategies, such as:

- Gradient descent with manually adjusted fixed step sizes (Frecon et al., 2018; Ji et al., 2020). The main disadvantage of this technique is that it requires a careful tuning of the step size for each experiment. In addition to being potentially tedious, it does not lead to an automatic procedure.

- L-BFGS (as in Deledalle et al. 2014). L-BFGS is a quasi-Newton algorithm that exploits past iterates to approximate the Hessian and propose a better descent direction, which is combined with some line search (Nocedal and Wright, 2006). Yet, due to the approximate gradient computation, we observed that L-BFGS did not always converge.

- ADAM (Kingma and Ba, 2014). It turned out to be inappropriate to the present setting. ADAM was very sensitive to the initial step size and required a careful tuning for each experiment.

- Iteration specific step sizes obtained by line search (Pedregosa, 2016). While the approach from Pedregosa (2016) requires no tuning, we observed that it could diverge when close to the optimum. The adaptive step size strategy proposed in Algorithm 6, used in all the experiments, turned out to be robust and efficient across problems and datasets.

Remark 16 (Uniqueness) *The solution of Problem (1) may be non-unique, leading to a multi-valued regularization path* $\lambda \mapsto \hat{\beta}^{(\lambda)}$ *(Liu et al., 2020) and requiring tools such as* optimistic gradient *(Dempe et al., 2015, Chap. 3.8). Though it is not possible to ensure uniqueness in practice, we did not face experimental issues due to potential non-uniqueness. For the Lasso, this experimental observation can be theoretically justified (Tibshirani, 2013): when the design matrix is sampled from a continuous distribution, the solution of the Lasso is almost surely unique.*

Remark 17 (Initialization) *One advantage of the non-smooth case with the* ℓ_1 *norm is that one can find a good initialization point: there exists a value* $\lambda_{\max}$ *(see Table 1) such that the solution of Problem (1) vanishes for* $\lambda \geq \lambda_{\max}$*. Hence, a convenient and robust initialization value can be chosen as* $e^{\lambda} = e^{\lambda_{\max}}/100$*. This is in contrast with the smooth case, where finding a good initialization heuristic is hard: starting in flat zones can lead to poor performance for gradient-based methods (Pedregosa, 2016).*

4. Experiments

In this section, we illustrate the benefits of our proposed Algorithm 5 to compute hypergradients and Algorithm 6 to solve Problem (2). Our package, `sparse-ho`, is implemented in `Python`. It relies on `Numpy` (Harris et al., 2020), `Numba` (Lam et al., 2015) and `SciPy` (Virtanen et al., 2020). Figures were plotted using `matplotlib` (Hunter, 2007). The package is available under BSD3 license at `https://github.com/qb3/sparse-ho`, with documentation and examples available at `https://qb3.github.io/sparse-ho/`. Online code includes scripts to reproduce all figures and experiments of the paper.

4.1 Hypergradient computation

Comparison with alternative approaches (Figure 4). First, we compare different methods to compute the hypergradient:

- Forward-mode differentiation of proximal coordinate descent (Algorithm 3).
- Reverse-mode differentiation of proximal coordinate descent (Algorithm 4).

Algorithm 5 IMPLICIT DIFFERENTIATION

input : $\lambda \in \mathbb{R}, \epsilon > 0$
init : $\gamma > 0$
`// compute the solution of inner problem`
Find β such that: $\Phi(\beta, \lambda) - \Phi(\hat{\beta}, \lambda) \leq \epsilon$
`// compute the gradient`
Compute the generalized support S of β,
$z = \beta - \gamma \nabla f(\beta)$
$\mathcal{J}_{S^c:} = \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z)_{S^c}$
$s = |S|$
$A = \mathrm{Id}_s - \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z)_S \odot (\mathrm{Id}_s - \gamma \nabla^2_{S,S} f(\beta))$
Find $v \in \mathbb{R}^s$ s.t. $\|A^{-1\top} \nabla_S \mathcal{C}(\beta) - v\| \leq \epsilon$
$B = \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z)_S$
$\quad - \gamma \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z)_S \odot \nabla^2_{S,S^c} f(\beta) \mathcal{J}_{S^c}$
$\nabla \mathcal{L}(\lambda) = \mathcal{J}^\top_{S^c:} \nabla_{S^c} \mathcal{C}(\beta) + v^\top B$
return $\mathcal{L}(\lambda) \triangleq \mathcal{C}(\beta), \nabla \mathcal{L}(\lambda)$

Algorithm 6 GRADIENT DESCENT WITH APPROXIMATE GRADIENT

input : $\lambda \in \mathbb{R}^r, (\epsilon_i)$
init : use_adaptive_step_size = True
for $i = 1, \ldots,$ iter **do**
  $\lambda^{\text{old}} \leftarrow \lambda$
  `// compute the value and the gradient`
  $\mathcal{L}(\lambda), \nabla \mathcal{L}(\lambda) \leftarrow$ Algorithm 5$(X, y, \lambda, \epsilon_i)$
  if use_adaptive_step_size **then**
    $\alpha = 1/\|\nabla \mathcal{L}(\lambda)\|$
  $\lambda \mathrel{-}= \alpha \nabla \mathcal{L}(\lambda)$ `// gradient step`
  if $\mathcal{L}(\lambda) > \mathcal{L}(\lambda^{\text{old}})$ **then**
    use_adaptive_step_size = False
    $\alpha \mathrel{/}= 10$
return λ

Table 5: Characteristics of the datasets used for the experiments.

name	# samples n	# features p	# classes q	density
breast cancer	569	30	–	1
diabetes	442	10	–	1
leukemia	72	7,129	–	1
gina agnostic	3,468	970	–	1
rcv1	20,242	19,960	–	3.7×10^{-3}
real-sim	72,309	20,958	–	2.4×10^{-3}
news20	19,996	632,983	–	6.1×10^{-4}
mnist	60.000	683	10	2.2×10^{-1}
usps	7,291	256	10	1
rcv1 (multiclass)	15,564	16,245	53	4.0×10^{-3}
aloi	108,000	128	1,000	2.4×10^{-1}

- `cvxpylayers` (Agrawal et al., 2019), a software based on `cvxpy` (Diamond and Boyd, 2016), solving *disciplined parametrized programming* and providing derivatives with respect to the parameters of the program. It is thus possible to use `cvxpylayers` to compute gradients with respect to the regularization parameters.

Figure 4 compares the time taken by multiple methods to compute a single hypergradient $\nabla \mathcal{L}(\lambda)$ for the Lasso (see Table 1), for multiple values of λ. It shows the time taken to compute the regression coefficients and the hypergradient, as a function of the number of

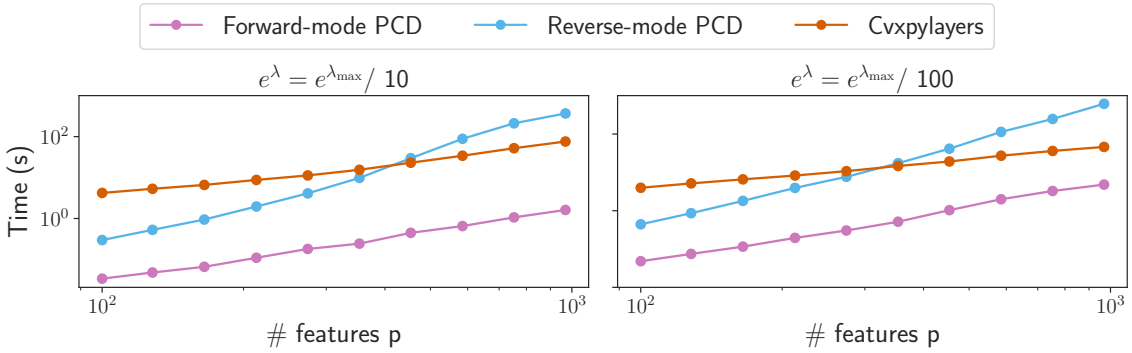

Figure 4: **Lasso with hold-out criterion:** time comparison on the gina dataset to compute a single hypergradient as a function of the number of features, for two values of λ, $e^\lambda = e^{\lambda_{\max}}/10$ (left) and $e^\lambda = e^{\lambda_{\max}}/100$ (right).

columns, sampled from the design matrix from the *gina* dataset. The columns were selected at random and 10 repetitions were performed for each point of the curves. In order to aim for good numerical precision, problems were solved up to a duality gap of 10^{-6} for the forward-mode and the reverse-mode. `cvxpylayers` relies on `cvxpy`, solving Problem (1) using a splitting conic solver (O'Donoghue et al., 2019). Since the termination criterion of the splitting conic solver is not exactly the duality gap (O'Donoghue et al., 2016, Sec. 3.5), we used the default tolerance of 10^{-4}. The hypergradient $\nabla\mathcal{L}(\lambda)$ was computed for hold-out mean squared error (see Table 2).

The forward-mode differentiation of proximal coordinate descent is one order of magnitude faster than `cvxpylayers` and two orders of magnitude faster than the reverse-mode differentiation of proximal coordinate descent. The larger the value of λ, the sparser the coefficients β are, leading to significant speedups in this regime. This performance is in accordance with the lower time cost of the forward mode in Table 4.

Combining implicit differentiation with state-of-the art solvers (Figures 5 and 6). We now compare the different approaches described in Section 3:

- Forward-mode differentiation of proximal coordinate descent (Algorithm 3).
- Implicit differentiation (Algorithm 5) with proximal coordinate descent to solve the inner problem. For efficiency, this solver was coded in `Numba` (Lam et al., 2015).
- Implicit differentiation (Algorithm 5) with state-of-the-art algorithm to solve the inner problem: we used `Celer` (Massias et al., 2020) for the Lasso, and `Lightning` (Blondel and Pedregosa, 2016) for the SVM.

Figure 5 shows for three datasets and two values of regularization parameters the absolute difference between the exact hypergradient and the approximate hypergradient obtained via multiple algorithms as a function of time. Figure 6 reports similar results for the SVM,

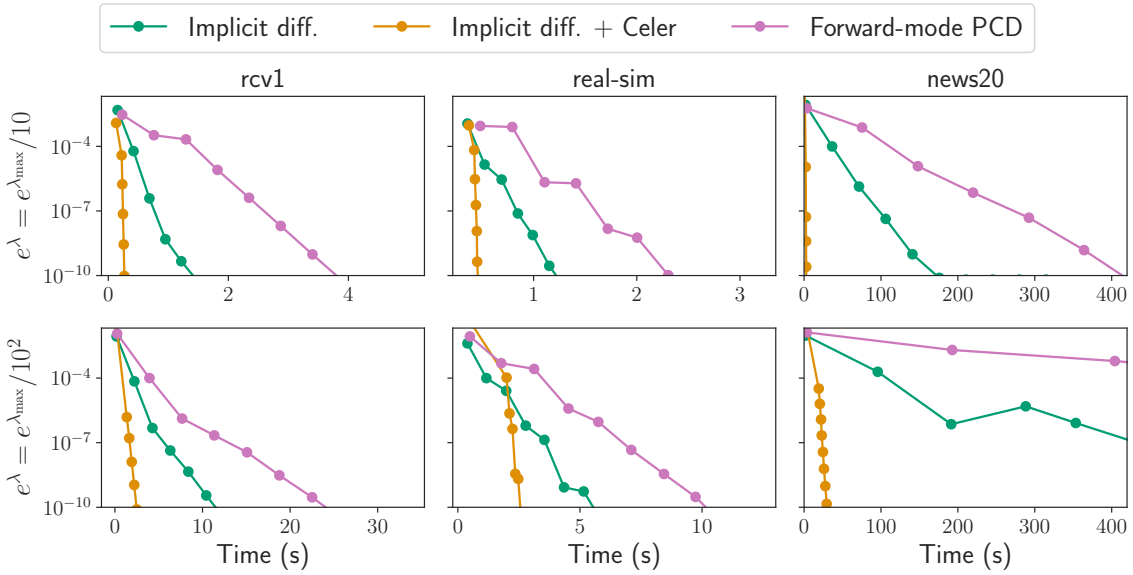

Figure 5: **Lasso with hold-out criterion:** absolute difference between the exact hypergradient (using $\hat{\beta}$) and the iterate hypergradient (using $\beta^{(k)}$) of the Lasso as a function of time. Results are for three datasets and two different regularization parameters. "Implicit diff. + `Celer`)" uses `Celer` (Massias et al., 2020) instead of our proximal coordinate descent implementation.

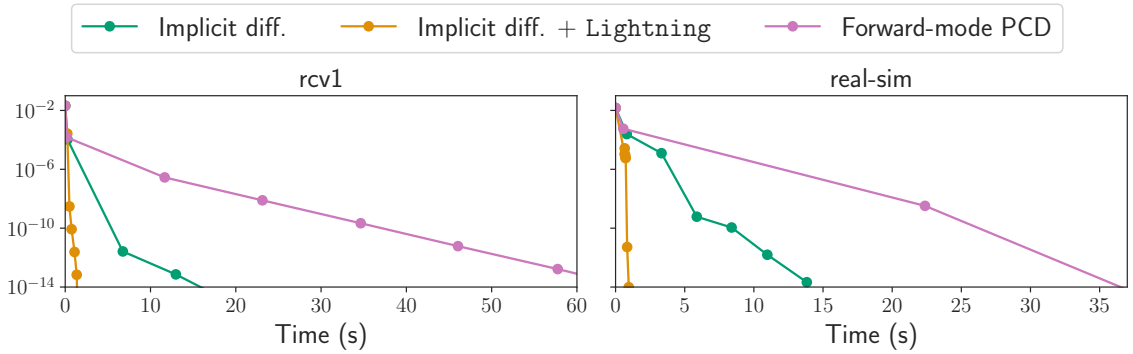

Figure 6: **SVM with hold-out criterion:** absolute difference between the exact hypergradient (using $\hat{\beta}$) and the iterate hypergradient (using $\beta^{(k)}$) of the SVM as a function of time. "Implicit diff. + `Lightning`" uses `Lightning` (Blondel and Pedregosa, 2016), instead of our proximal coordinate descent implementation.

on the same datasets, except *news20*, which is not well suited for SVM, due to limited number of samples.

First, it demonstrates that implicit differentiation methods are faster than the forward-mode of proximal coordinate descent (pink). This illustrates the benefits of restricting the gradient computation to the support of the Jacobian, as described in Section 3.5. Second, thanks to the flexibility of our approach, we obtain additional speed-ups by combining implicit differentiation with a state-of-the-art solver, `Celer`. The resulting method (orange) significantly improves over implicit differentiation using a vanilla proximal coordinate descent (green).

4.2 Resolution of the bilevel optimization problem

In this section we compare multiple methods to find the optimal hyperparameters for the Lasso, elastic net and multiclass sparse logistic regression. The following methods are compared:

- **Grid-search**: for the Lasso and the elastic net, the number of hyperparameters is small, and grid-search is tractable. For the Lasso we chose a grid of 100 hyperparameters λ, uniformly spaced between $\lambda_{\max} - \ln(10^4)$ and $\lambda_{\max}$. For the elastic net we chose for each of the two hyperparameters a grid of 10 values uniformly spaced between $\lambda_{\max}$ and $\lambda_{\max} - \ln(10^4)$. The product grid thus has 10^2 points.
- **Random-search**: we chose 30 values of λ sampled uniformly between $\lambda_{\max}$ and $\lambda_{\max} - \ln(10^4)$ for each hyperparameter. For the elastic net we chose 30 points sampled uniformly in $[\lambda_{\max} - \ln(10^4), \lambda_{\max}] \times [\lambda_{\max} - \ln(10^4), \lambda_{\max}]$.
- **SMBO**: this algorithm is SMBO using as criterion expected improvement (EI) and the Tree-structured Parzen Estimator (TPE) as model. First it evaluates $\mathcal{L}$ using 5 values of λ, chosen uniformly at random between $\lambda_{\max}$ and $\lambda_{\max} - \ln(10^4)$. Then a TPE model is fitted on the data points $(\lambda^{(1)}, \mathcal{L}(\lambda^{(1)})), \dots, (\lambda^{(5)}, \mathcal{L}(\lambda^{(5)}))$. Iteratively, the EI is used to choose the next point to evaluate $\mathcal{L}$ at, and this value is used to update the model. We used the `hyperopt` implementation (Bergstra et al., 2013).
- **1st order**: first-order method with exact gradient (Algorithm 6 with constant tolerances $\epsilon_i = 10^{-6}$), with $\lambda_{\max} - \ln(10^2)$ as a starting point.
- **1st order approx**: a first-order method using approximate gradient (Algorithm 6 with tolerances ϵ_i, geometrically decreasing from 10^{-2} to 10^{-6}), with $\lambda_{\max} - \ln(10^2)$ as a starting point.

Outer criterion. In the Lasso and elastic net experiments, we pick a K-fold CV loss as outer criterion[5]. Hence, the dataset (X, y) is partitioned into K hold-out datasets $(X^{\text{train}_k}, y^{\text{train}_k}), (X^{\text{val}_k}, y^{\text{val}_k})$. The bilevel optimization problems then write:

5. In our experiments the default choice is $K = 5$.

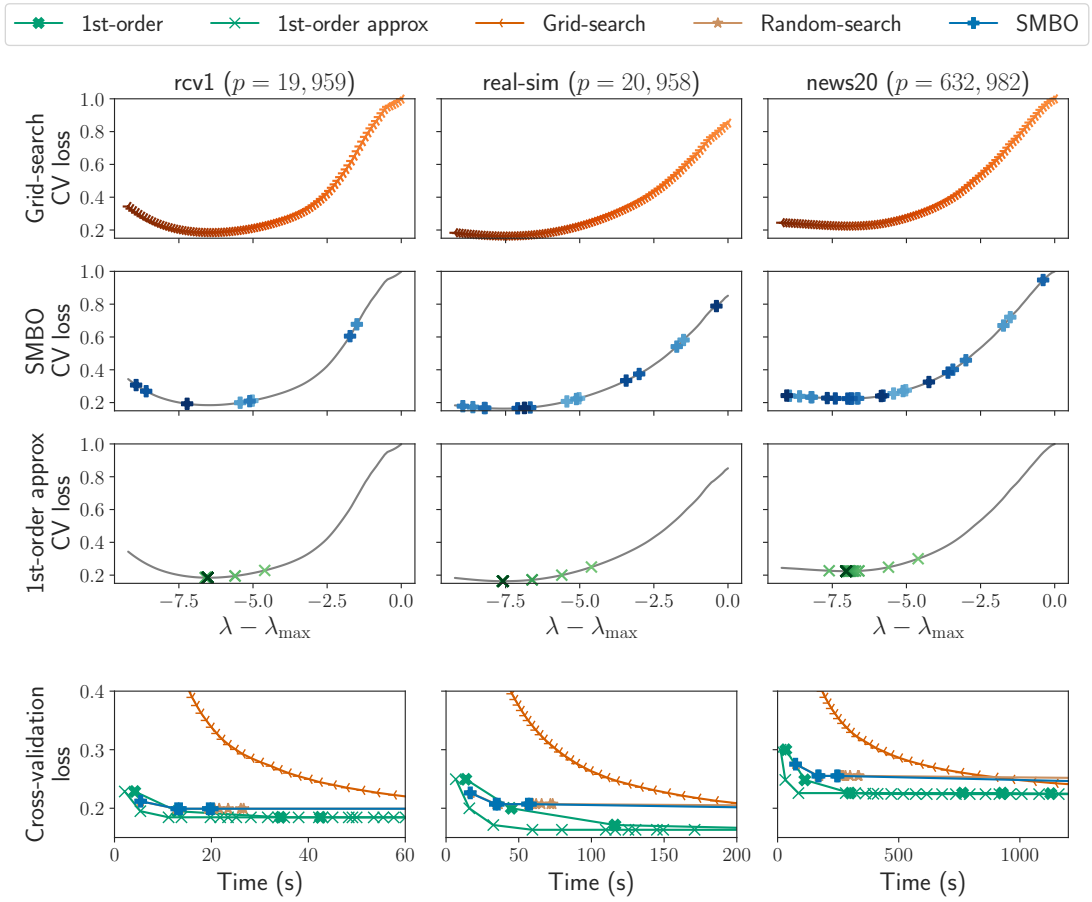

Figure 7: **Lasso with cross-validation criterion:** cross-validation loss as a function of λ (black line, top) and as a function of time (bottom). Lighter markers correspond to earlier iterations of the algorithm.

$$\begin{aligned} &\underset{\lambda=(\lambda_1,\lambda_2)\in\mathbb{R}^2}{\arg\min}\ \mathcal{L}(\lambda) = \frac{1}{K}\sum_{k=1}^{K} \|y^{\text{val}_k} - X^{\text{val}_k}\hat{\beta}^{(\lambda,k)}\|_2^2 \\ &s.t.\ \hat{\beta}^{(\lambda,k)} \in \underset{\beta\in\mathbb{R}^p}{\arg\min}\ \frac{1}{2n}\left\|y^{\text{train}_k} - X^{\text{train}_k}\beta\right\|_2^2 + e^{\lambda_1}\|\beta\|_1 + \frac{e^{\lambda_2}}{2}\|\beta\|_2^2, \quad \forall k \in [K] \ , \end{aligned} \tag{16}$$

while Lasso CV is obtained taking $\lambda_2 \to -\infty$ in the former. By considering an extended variable $\beta \in \mathbb{R}^{K\times p}$, cross-validation can be cast as an instance of Problem (2).

Figure 7 represents the cross-validation loss in Lasso CV as a function of the regularization parameter λ (black curve, three top rows) and as a function of time (bottom). Each point corresponds to the evaluation of the cross-validation criterion for one λ value. The top

rows show cross-validation loss as a function of λ, for the grid-search, the SMBO optimizer and the first-order method. The lightest crosses correspond to the first iterations of the algorithm and the darkest, to the last ones. For instance, Lasso grid-search starts to evaluate the cross-validation function with $\lambda = \lambda_{\max}$ and then decreases to $\lambda = \lambda_{\max} - \ln(10^4)$. On all the datasets, first-order methods are faster to find the optimal regularization parameter, requiring only 5 iterations.

Figure 8 represents the level sets of the cross-validation loss for the elastic net (three top rows) and the cross-validation loss as a function of time (bottom). One can see that after 5 iterations the SMBO algorithm (blue crosses) suddenly slows down (bottom) as the hyperparameter suggested by the algorithm leads to a costly optimization problem to solve, while first-order methods converge quickly as for Lasso CV. In the present context, inner problems are slower to solve for low values of the regularization parameters.

Multiclass sparse logistic regression (# classes hyperparameters, Figure 9). We consider a multiclass classification problem with q classes. The design matrix is noted $X \in \mathbb{R}^{n \times p}$, and the target variable $y \in \{1, \dots, q\}^n$. We chose to use a one-versus-all model with q regularization parameters. We use a binary cross-entropy for the inner loss:

$$\psi^k(\beta, \lambda_k; X, y) \triangleq -\frac{1}{n} \sum_{i=1}^{n} \left(\mathbb{1}_{y_i = k} \ln(\sigma(X_{i:}\beta)) + (1 - \mathbb{1}_{y_i = k}) \ln(1 - \sigma(X_{i:}\beta)) \right) + e^{\lambda_k} \|\beta\|_1 \ ,$$

and a multiclass cross-entropy for the outer criterion:

$$\mathcal{C}\left(\hat{\beta}^{(\lambda_1)}, \dots, \hat{\beta}^{(\lambda_q)}; X, y\right) \triangleq -\sum_{i=1}^{n} \sum_{k=1}^{q} \ln\left(\frac{e^{X_{i:}\hat{\beta}^{(\lambda_k)}}}{\sum_{l=1}^{q} e^{X_{i:}\hat{\beta}^{(\lambda_l)}}} \right) \mathbb{1}_{y_i = k} \ . \tag{17}$$

With a single train/test split, the bilevel problem to solve writes:

$$\begin{aligned} \underset{\lambda \triangleq (\lambda_1, \dots, \lambda_q) \in \mathbb{R}^q}{\arg\min} \quad & \mathcal{C}\left(\hat{\beta}^{(\lambda_1)}, \dots, \hat{\beta}^{(\lambda_q)}; X^{\text{test}}, y^{\text{test}}\right) \\ s.t. \ \hat{\beta}^{(\lambda_k)} & \in \underset{\beta \in \mathbb{R}^p}{\arg\min}\, \psi^k(\beta, \lambda_k; X^{\text{train}}, y^{\text{train}}) \quad \forall k \in [q] \ . \end{aligned} \tag{18}$$

Figure 9 represents the multiclass cross-entropy (top), the accuracy on the validation set (middle) and the accuracy on the test set (unseen data, bottom). When the number of hyperparameter is moderate ($q = 10$, on *mnist* and *usps*), the multiclass cross-entropy reached by SMBO and random techniques is as good as first-order techniques. This is expected and follows the same conclusion as Bergstra and Bengio (2012); Frazier (2018): when the number of hyperparameters is moderate, SMBO and random techniques can be used efficiently. However, when the number of hyperparameters increases (*rcv1*, $q = 53$ and *aloi*, $q = 1000$), the hyperparameter space is too large: zero-order solvers simply fail. On the contrary, first-order techniques manage to find hyperparameters leading to significantly better accuracy.

5. Conclusion

In this work we considered the problem of hyperparameter optimization to select the regularization parameter of linear models with non-smooth objective. Casting this problem as a bilevel optimization problem, we proposed to use first-order methods. We showed that the usual automatic differentiation techniques, implicit differentiation, forward and reverse modes, can be used to compute the hypergradient, despite the non-smoothness of the inner problem. Experimentally, we showed the interest of first-order techniques to solve bilevel optimization on a wide range of estimators (ℓ_1 penalized methods, SVM, etc.) and datasets. The presented techniques could also be extended to more general bilevel optimization problems, in particular implicit differentiation could be well suited for meta-learning problems, with a potentially large number of hyperparameters.

Acknowledgements

This work was partially funded by the ERC Starting Grant SLAB ERC-StG-676943 , the ANR CaMeLOt ANR-20-CHIA-0001-01, and the ANR grant GraVa ANR-18-CE40-0005. Part of this work has been carried out at the Machine Learning Genoa (MaLGa) center, Università di Genova (IT). M. M. acknowledges the financial support of the European Research Council (grant SLING 819789).

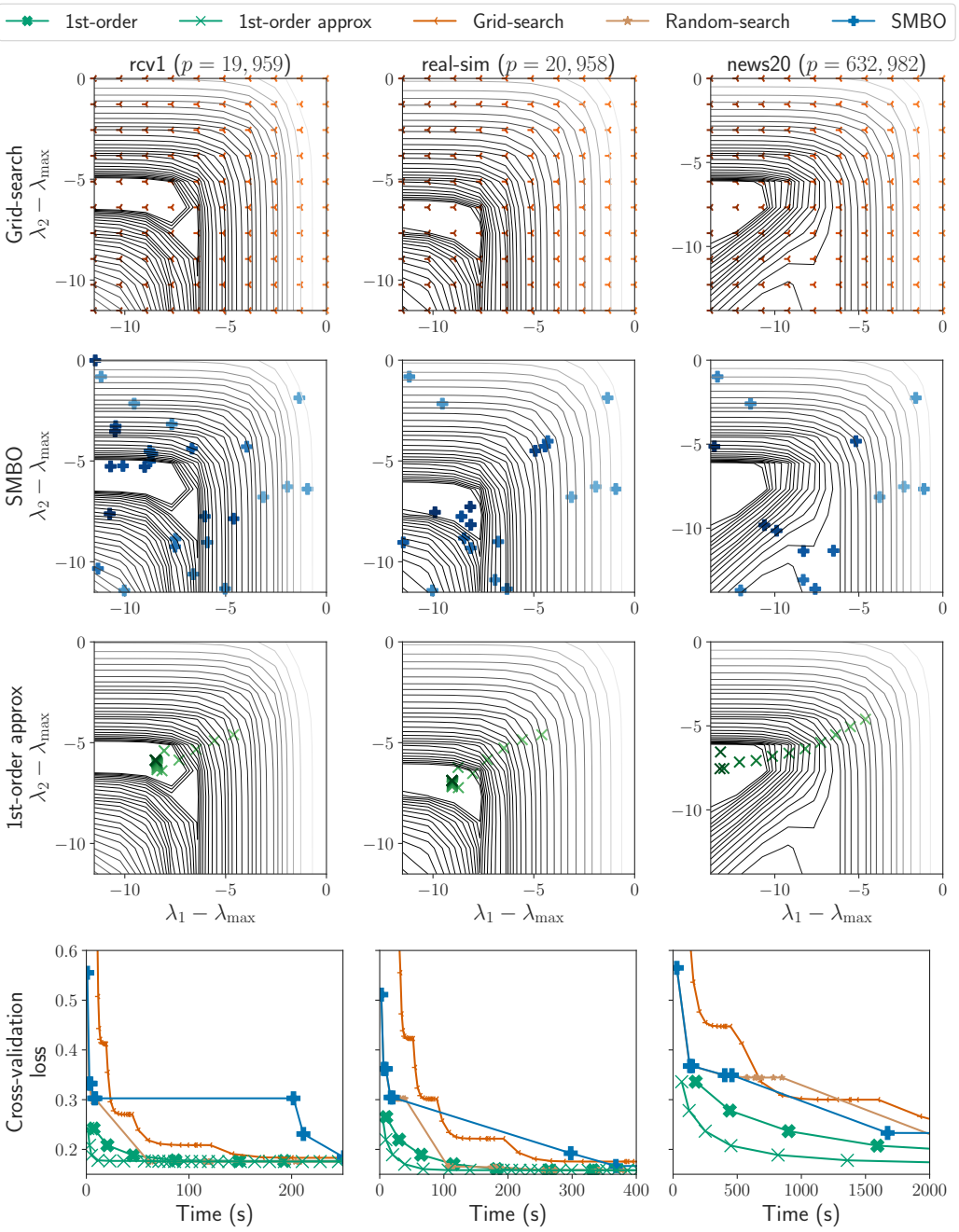

Figure 8: **Elastic net cross-validation, time comparison (2 hyperparameters).** Level sets of the cross-validation loss (black lines, top) and cross-validation loss as a function of time (bottom) on *rcv1*, *real-sim* and *news20* datasets.

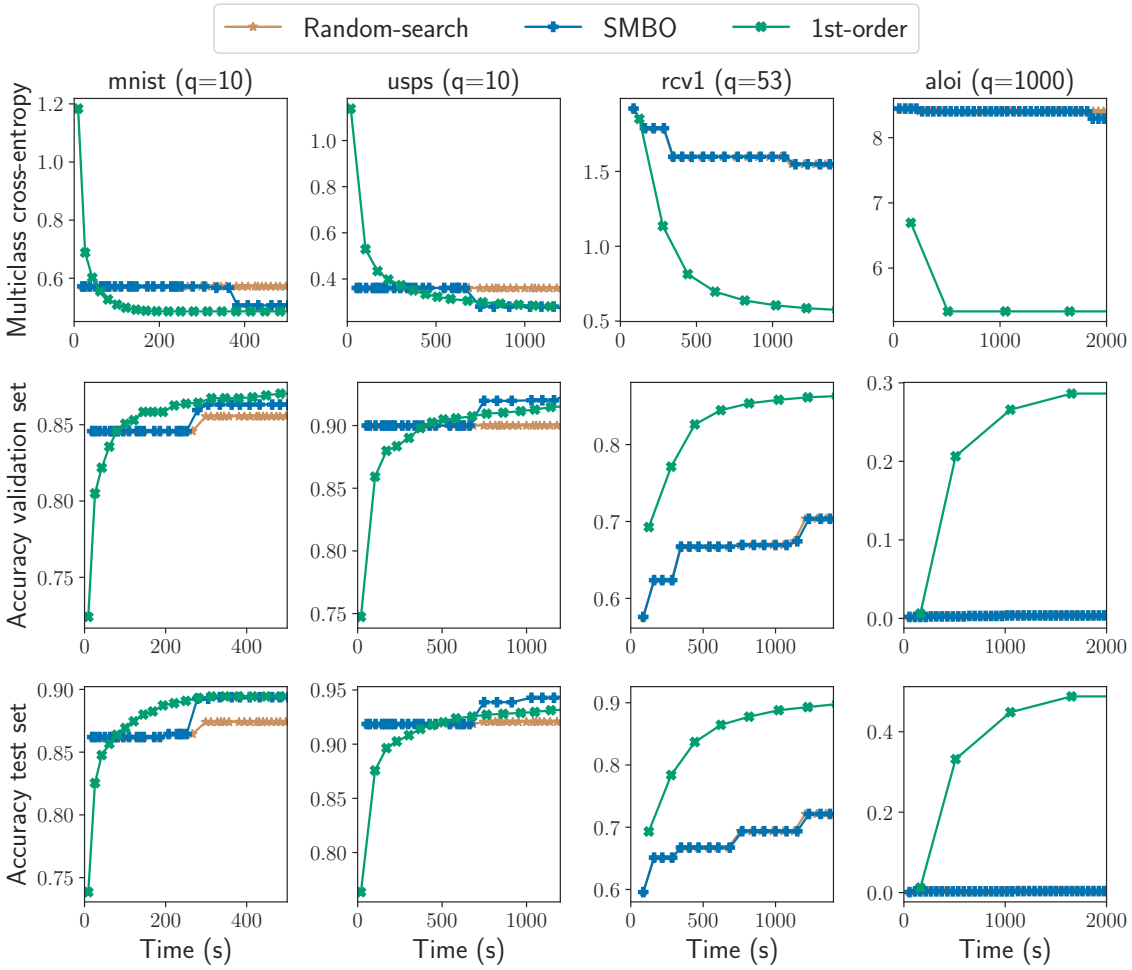

Figure 9: **Multiclass sparse logistic regression hold-out, time comparison (# classes hyperparameters).** Multiclass cross-entropy (top), accuracy on the validation set (middle), and accuracy on the test set (bottom) as a function of time on *mnist*, *usps* ($q = 10$ classes), *rcv1* ($q = 53$ classes), *aloi* ($q = 1000$ classes).

References

P. Ablin, G. Peyré, and T. Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *International Conference on Machine Learning*, pages 32–41. PMLR, 2020.

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. In *NeurIPS*, pages 9558–9570, 2019.

H. Akaike. A new look at the statistical model identification. *IEEE Trans. Automat. Control*, AC-19:716–723, 1974.

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

S. Arlot and A. Celisse. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *NeurIPS*, 2019.

S. Bai, V. Koltun, and J. Z. Kolter. Multiscale deep equilibrium models. *NeurIPS*, 2020.

A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

A. Belloni, V. Chernozhukov, and L. Wang. Square-root Lasso: pivotal recovery of sparse signals via conic programming. *Biometrika*, 98(4):791–806, 2011.

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. *Journal of Machine Learning Research*, 13(2), 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *NeurIPS*, 2011.

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.

Q. Bertrand, Q. Klopfenstein, M. Blondel, S. Vaiter, A. Gramfort, and J. Salmon. Implicit differentiation of Lasso-type models for hyperparameter optimization. *ICML*, 2020.

P. J. Bickel, Y. Ritov, and A. B. Tsybakov. Simultaneous analysis of Lasso and Dantzig selector. *Ann. Statist.*, 37(4):1705–1732, 2009.

M. Blondel and F. Pedregosa. Lightning: large-scale linear classification, regression and ranking in python, 2016.

J. Bolte and E. Pauwels. Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning. *Mathematical Programming*, pages 1–33, 2020a.

J. Bolte and E. Pauwels. A mathematical model for automatic differentiation in machine learning. *arXiv preprint arXiv:2006.02080*, 2020b.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

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.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM J. Sci. Comput.*, 20(1):33–61, 1998.

H. Cherkaoui, J. Sulam, and T. Moreau. Learning to solve TV regularised problems with unrolled algorithms. *NeurIPS*, 33, 2020.

B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

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

A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *NeurIPS*, pages 1646–1654, 2014.

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.

S. Dempe, V. Kalashnikov, G. A. Pérez-Valdés, and N. Kalashnykova. Bilevel programming problems. *Energy Systems. Springer, Berlin*, 2015.

L. Devroye and T. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory*, 25(5):601–604, 1979.

S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

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

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.

J. Fan and J. Lv. Sure independence screening for ultrahigh dimensional feature space. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 70(5):849–911, 2008.

M. Feurer and F. Hutter. Hyperparameter optimization. In *Automated Machine Learning*, pages 3–33. Springer, Cham, 2019.

C. S. Foo, C. B. Do, and A. Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *NeurIPS*, pages 377–384, 2008.

A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

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

L. Franceschi, P. Frasconi, S. Salzo, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, pages 1563–1572, 2018.

P.I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

J. Frecon, S. Salzo, and M. Pontil. Bilevel learning of the group lasso structure. In *NeurIPS*, 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.

S. Ghadimi and M. Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

I. Goodfellow, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

S. Gould, B. Fernando, A. Cherian, P. Anderson, R. S. Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447.*, 2016.

R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo. On the iteration complexity of hypergradient computation. *ICML*, 2020.

W. L. Hare and A. S. Lewis. Identifying active manifolds. *Algorithmic Operations Research*, 2(2):75–75, 2007.

C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fern'andez del R'ıo, M. Wiebe, P. Peterson, P. G'erard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5(Oct):1391–1415, 2004.

M. R. Hestenes and E. Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC, 1952.

N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

J. D. Hunter. Matplotlib: A 2d graphics environment. *IEEE Annals of the History of Computing*, 9(03):90–95, 2007.

F. Hutter, J. Lücke, and L. Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29(4):329–337, 2015.

K. Ji, J. Yang, and Y. Liang. Provably faster algorithms for bilevel optimization and applications to meta-learning. *arXiv preprint arXiv:2010.07962*, 2020.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NeurIPS*, pages 315–323, 2013.

T. B. Johnson and C. Guestrin. Blitz: A principled meta-algorithm for scaling sparse optimization. In *ICML*, pages 1171–1179, 2015.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Q. Klopfenstein, Q. Bertrand, A. Gramfort, J. Salmon, and S. Vaiter. Model identification and local linear convergence of coordinate descent. *arXiv preprint arXiv:2010.11825*, 2020.

K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine Learning Research*, 8(8):1519–1555, 2007.

R. Kohavi and G. H. John. Automatic parameter selection by minimizing estimated error. In *Machine Learning Proceedings 1995*, pages 304–312. Elsevier, 1995.

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.

Y. A. LeCun, L. Bottou, G. B. Orr, and K-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 1998.

J. Liang, J. Fadili, and G. Peyré. Local linear convergence of forward–backward under partial smoothness. In *NeurIPS*, pages 1970–1978, 2014.

J. Liang, J. Fadili, and G. Peyré. Activity identification and local linear convergence of Forward–Backward-type Methods. *SIAM J. Optim.*, 27(1):408–437, 2017.

S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

P-L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

R. Liu, P. Mu, X. Yuan, S. Zeng, and J. Zhang. A generic first-order algorithmic framework for bi-level programming beyond lower-level singleton. *ICML*, 2020.

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

K. Lounici. Sup-norm convergence rate and sign concentration property of Lasso and Dantzig estimators. *Electron. J. Stat.*, 2:90–102, 2008.

J. Mairal and B. Yu. Complexity analysis of the lasso regularization path. In *ICML*, pages 353–360, 2012.

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. *Journal of Machine Learning Research*, 21(234):1–33, 2020.

S. Mehmood and P. Ochs. Differentiating the value function by using convex duality. In *AISTATS*, pages 3871–3879. PMLR, 2021.

J. Mockus. The bayesian approach to local optimization. In *Bayesian Approach to Global Optimization*, pages 125–156. Springer, 1989.

Y. Nesterov. *Introductory lectures on convex optimization*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, Boston, MA, 2004.

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

J. Nutini. *Greed is good: greedy optimization methods for large-scale structured problems*. PhD thesis, University of British Columbia, 2018.

J. Nutini, M. Schmidt, and W. Hare. "active-set complexity" of proximal gradient: How long does it take to find the sparsity pattern? *Optimization Letters*, 13(4):645–655, 2019.

P. Ochs, R. Ranftl, T. Brox, and T. Pock. Bilevel optimization with nonsmooth lower level problems. In *SSVM*, pages 654–665, 2015.

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.2, 2019.

F. Pedregosa. Hyperparameter optimization with approximate gradient. In *ICML*, volume 48, pages 737–746, 2016.

F. Pedregosa, R. Leblond, and S. Lacoste-Julien. Breaking the nonsmooth barrier: A scalable parallel method for composite optimization. *NeurIPS*, pages 56–65, 2017.

G. Peyré and J. M. Fadili. Learning analysis sparsity priors. In *Sampta*, 2011.

J. C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in large margin classifiers*, pages 61–74. MIT Press, 1999.

R. A. Poliquin and R. T. Rockafellar. Generalized hessian properties of regularized nonsmooth functions. *SIAM Journal on Optimization*, 6(4):1121–1137, 1996.

B. T. Polyak. Introduction to optimization. *Inc., Publications Division, New York*, 1, 1987.

A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. In *NeurIPS*, pages 113–124, 2019.

L. A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automaton & Remote Control*, 24:1337–1342, 1963.

S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *Ann. Statist.*, 35(3): 1012–1030, 2007.

G. Schwarz. Estimating the dimension of a model. *Ann. Statist.*, 6(2):461–464, 1978.

M. W. Seeger. Cross-validation optimization for large scale structured classification kernel methods. *Journal of Machine Learning Research*, 9:1147–1178, 2008.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NeurIPS*, pages 2960–2968, 2012.

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.

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

R. Tibshirani, J. Bien, J. Friedman, T. J. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *J. R. Stat. Soc. Ser. B Stat. Methodol.*, 74(2):245–266, 2012.

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

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, G. Peyré, and J. Fadili. Model consistency of partly smooth regularizers. *IEEE Trans. Inf. Theory*, 64(3):1725–1737, 2018.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020.

R. E. Wengert. A simple automatic derivative evaluation program. *Communications of the ACM*, 7(8):463–464, 1964.

E. Winston and Z. Kolter. Neural monotone operator equilibrium networks. *NeurIPS*, 2020.

L. Zhang, M. Mahdavi, and R. Jin. Linear convergence with condition number independent access of full gradients. *NeurIPS*, 26:980–988, 2013.

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.

Appendix A. Additional lemmas

A.1 Differentiability of the proximal operator

Here we recall results on the differentiability of the proximal operator at the optimum.

Lemma 18 (Klopfenstein et al. 2020, Lemmas 2 and 3) *Let $0 < \gamma_j \leq 1/L_j$. Let $\lambda \in \mathbb{R}^r$ and Λ a neighborhood of λ. Consider a solution $\hat{\beta} \in \arg\min_{\beta \in \mathbb{R}^p} \Phi(\beta, \lambda)$ and $\hat{S}$ its generalized support. Suppose*

1. *Assumptions 2, 3 and 6 hold.*
2. *Assumption 4 hold on Λ.*

Then, for all $j \in \hat{S}$, the map $\beta \mapsto \mathrm{prox}_{\gamma_j g_j(\cdot,\lambda)}$ is differentiable at $\hat{\beta}_{\hat{S}}$. Moreover, for all $j \in \hat{S}^c$, $\mathrm{prox}_{\gamma_j g_j(\cdot,\lambda)}$ is constant around $\hat{\beta}_j - \gamma_j \nabla_j f(\hat{\beta})$. Thus, $\beta \mapsto \mathrm{prox}_{\gamma_j g_j(\cdot,\lambda)}(\beta_j - \gamma_j \nabla_j f(\beta))$ is differentiable at $\hat{\beta}$ with gradient 0.

A.2 Linear convergence

We now detail the following result: an asymptotic vector autoregressive sequence, with an error term vanishing linearly to 0, converges linearly to its limit. In a more formal way:

Lemma 19 *Let $A \in \mathbb{R}^{p \times p}, b \in \mathbb{R}$ with $\rho(A) < 1$. Let $(\mathcal{J}^{(k)})_{k \in \mathbb{N}}$ be a sequence of $\mathbb{R}^p$ such that:*

$$\mathcal{J}^{(k+1)} = A\mathcal{J}^{(k)} + b + \epsilon^{(k)} \ , \tag{19}$$

with $(\epsilon^{(k)})_{k \in \mathbb{N}}$ a sequence which converges linearly to 0, then $(\mathcal{J}^{(k)})_{k \in \mathbb{N}}$ converges linearly to its limit $\hat{\mathcal{J}} \triangleq (\mathrm{Id} - A)^{-1} b$.

Proof Assume $(\epsilon^{(k)})_{k \in \mathbb{N}}$ converges linearly. Then, there exists $c_1 > 0, 0 < \nu < 1$ such that:

$$\|\epsilon^{(k)}\| \leq c_1 \nu^k \ .$$

Applying a standard result on spectral norms (see (Polyak, 1987, Chapter 2, Lemma 1)) yields a bound on $\|A^k\|_2$. More precisely, for every $\delta > 0$ there is a constant $c_2(\delta) = c_2$ such that

$$\|A^k\|_2 \leq c_2(\rho(A) + \delta)^k \ .$$

Without loss of generality, we consider from now on a choice of δ such that $\rho(A) + \delta < 1$. Since $\hat{\mathcal{J}} = (\mathrm{Id} - A)^{-1} b$ the limit $\hat{\mathcal{J}}$ of the sequence satisfies:

$$\hat{\mathcal{J}} = A\hat{\mathcal{J}} + b \ . \tag{20}$$

Taking the difference between Equations (19) and (20) yields:

$$\mathcal{J}^{(k+1)} - \hat{\mathcal{J}} = A(\mathcal{J}^{(k)} - \hat{\mathcal{J}}) + \epsilon^{(k)} \ . \tag{21}$$

Unrolling Equation (21) yields $\mathcal{J}^{(k+1)} - \hat{\mathcal{J}} = A^{k+1}(\mathcal{J}^{(0)} - \mathcal{J}) + \sum_{k'=0}^{k} A^{k'}\epsilon^{(k-k')}$. Taking the norm on both sides and using the triangle inequality leads to

$$\begin{aligned}
\|\mathcal{J}^{(k+1)} - \hat{\mathcal{J}}\|_2 &\leq \|A^{k+1}(\mathcal{J}^{(0)} - \mathcal{J})\|_2 + \sum_{k'=0}^{k} \|A^{k'}\|_2 \|\epsilon^{(k-k')}\| \\
&\leq \|A^{k+1}\|_2 \cdot \|\mathcal{J}^{(0)} - \hat{\mathcal{J}}\|_2 + c_1 \sum_{k'=0}^{k} \|A^{k'}\|_2 \cdot \nu^{k-k'} \\
&\leq c_2(\rho(A) + \delta)^{k+1} \cdot \|\mathcal{J}^{(0)} - \hat{\mathcal{J}}\|_2 + c_1 \sum_{k'=0}^{k} c_2(\rho(A) + \delta)^{k'} \nu^{k-k'}
\end{aligned}$$

We can now split the last summand in two parts and obtain the following bound, reminding that $\rho(A) + \delta < 1$:

$$\begin{aligned}
\|\mathcal{J}^{(k+1)} - \hat{\mathcal{J}}\|_2 &\leq c_2(\rho(A) + \delta)^{k+1} \cdot \|\mathcal{J}^{(0)} - \hat{\mathcal{J}}\|_2 \\
&\quad + c_1 c_2 \left(\sum_{k'=0}^{k/2} (\rho(A) + \delta)^{k'} \nu^{k-k'} + \sum_{k'=k/2}^{k} (\rho(A) + \delta)^{k'} \nu^{k-k'} \right) \\
&\leq c_2(\rho(A) + \delta)^{k+1} \cdot \|\mathcal{J}^{(0)} - \hat{\mathcal{J}}\|_2 + \frac{c_1 c_2 (\rho(A) + \delta)}{1 - \rho(A) - \delta} \sqrt{\nu}^{k} \\
&\quad + \frac{c_1 c_2 \nu}{1 - \nu} \sqrt{(\rho(A) + \delta)}^{k} \quad .
\end{aligned}$$

Thus, $(\mathcal{J}^{(k)})_{k \in \mathbb{N}}$ converges linearly towards its limit $\hat{\mathcal{J}}$. ■

Appendix B. Proof of Theorem 12

Theorem 12 (Local linear convergence of the Jacobian) *Let $0 < \gamma \leq 1/L$. Suppose Assumptions 2, 3 and 6 hold. Let $\lambda \in \mathbb{R}^r$, Λ be a neighborhood of λ, and $\Gamma^{\Lambda} \triangleq \left\{ \hat{\beta}^{(\lambda)} - \gamma \nabla f(\hat{\beta}^{(\lambda)}) : \lambda \in \Lambda \right\}$. In addition, suppose hypotheses (H1) to (H4) from Theorem 9 are satisfied and the sequence $(\beta^{(k)})_{k \in \mathbb{N}}$ generated by Algorithm 1 (respectively by Algorithm 3) converges toward $\hat{\beta}$.*

Then, the sequence of Jacobians $(\mathcal{J}^{(k)})_{k \geq 0}$ generated by the forward-mode differentiation of proximal gradient descent (Algorithm 1) (respectively by forward-mode differentiation of proximal coordinate descent, Algorithm 3) converges locally linearly towards $\hat{\mathcal{J}}$.

Proof We first prove Theorem 12 for proximal gradient descent.

Proximal gradient descent case. Solving Problem (1) with proximal gradient descent leads to the following updates:

$$\beta^{(k+1)} = \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\underbrace{\beta^{(k)} - \gamma \nabla f(\beta^{(k)})}_{z^{(k)}}) \ . \tag{22}$$

Consider the following sequence $(\mathcal{J}^{(k)})_{k\in\mathbb{N}}$ defined by:

$$\mathcal{J}^{(k+1)} = \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^{(k)}) \odot \left(\mathrm{Id} - \gamma \nabla^2 f(\beta^{(k)})\right) \mathcal{J}^{(k)} + \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^{(k)}) \ . \tag{23}$$

Note that if $\operatorname{prox}_{\gamma g(\cdot,\lambda)}$ is not differentiable with respect to the first variable at $z^{(k)}$ (respectively with respect to the second variable λ), any weak Jacobian can be used. When (H3) holds, differentiating Equation (22) with respect to λ yields exactly Equation (23).

Assumptions 2 to 4 and 6 and the convergence of $(\beta^{(k)})$ toward $\hat{\beta}$ ensure proximal gradient descent algorithm has finite identification property (Liang et al., 2014, Thm. 3.1): we note K the iteration when identification is achieved. As before, the separability of g, Assumptions 2 to 4 and 6 ensure (see Lemma 18) $\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^k)_{\hat{S}^c} = 0$, for all $k \geq K$. Thus, for all $k \geq K$,

$$\mathcal{J}^{(k)}_{\hat{S}^c:} = \hat{\mathcal{J}}_{\hat{S}^c:} = \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^{(k)})_{\hat{S}^c:} \ .$$

The updates of the Jacobian then become:

$$\mathcal{J}^{(k+1)}_{\hat{S}:} = \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^{(k)})_{\hat{S}} \odot \left(\mathrm{Id} - \gamma \nabla^2_{\hat{S},\hat{S}} f(\beta^{(k)})\right) \mathcal{J}^{(k)}_{\hat{S}:} + \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(z^{(k)})_{\hat{S}:} \ .$$

From Assumption 6, we have that f is locally $\mathcal{C}^3$ at $\hat{\beta}$, $g(\cdot,\lambda)$ is locally $\mathcal{C}^2$ at $\hat{\beta}$ hence $\operatorname{prox}_{g(\cdot,\lambda)}$ is locally $\mathcal{C}^2$. The function $\beta \mapsto \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\beta - \gamma \nabla f(\beta))_{\hat{S}} \odot (\mathrm{Id} - \gamma \nabla^2_{\hat{S},\hat{S}} f(\beta))$ is differentiable at $\hat{\beta}$. Using (H4) we have that $\beta \mapsto \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\beta - \gamma \nabla f(\beta))_{\hat{S}:}$ is also differentiable at $\hat{\beta}$. Using the Taylor expansion of the previous functions yields:

$$\mathcal{J}^{(k+1)}_{\hat{S}:} = \underbrace{\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}} \odot \left(\mathrm{Id} - \gamma \nabla^2_{\hat{S},\hat{S}} f(\hat{\beta})\right)}_{A} \mathcal{J}^{(k)}_{\hat{S}:} + \underbrace{\partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}:}}_{b} + \underbrace{o(\|\beta^{(k)} - \hat{\beta}\|)}_{\epsilon^{(k)}} \ . \tag{24}$$

Thus, for $0 < \gamma \leq 1/L$,

$$\rho(A) \leq \|A\|_2 \leq \underbrace{\|\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\hat{z})_{\hat{S}}\|}_{\leq 1 \text{ (non-expansiveness)}} \cdot \underbrace{\|\mathrm{Id} - \gamma \nabla^2_{\hat{S},\hat{S}} f(\hat{\beta})\|_2}_{<1 \text{ (Assumption 7 and } 0<\gamma\leq 1/L)} < 1 \ . \tag{25}$$

The inequality on the derivative of the proximal operator comes from the non-expansiveness of proximal operators. The second inequality comes from Assumption 7 and $0 < \gamma \leq 1/L$.

Assumptions 2 to 4, 6 and 7 and the convergence of $(\beta^{(k)})$ toward $\hat{\beta}$ ensure $(\beta^{(k)})_{k\in\mathbb{N}}$ converges locally linearly (Liang et al., 2014, Thm. 3.1). The asymptotic autoregressive sequence in Equation (24), $\rho(A) < 1$, and the local linear convergence of $(\epsilon^{(k)})_{k\in\mathbb{N}}$, yield our result using Lemma 19.

We now prove Theorem 12 for proximal coordinate descent.

Proximal coordinate descent. Compared to proximal gradient descent, the analysis of coordinate descent requires studying functions defined as a the composition of p applications, each of them only modifying one coordinate.

Coordinate descent updates read as follows:

$$\beta_j^{(k,j)} = \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)} \underbrace{\left(\beta_j^{(k,j-1)} - \gamma_j \nabla_j f(\beta^{(k,j-1)})\right)}_{\triangleq z_j^{(k,j-1)}} \quad . \tag{26}$$

We consider the following sequence:

$$\begin{aligned}\mathcal{J}_{j:}^{(k,j)} = \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) \left(\mathcal{J}_{j:}^{(k,j-1)} - \gamma_j \nabla^2_{j:} f(\beta^{(k,j-1)}) \mathcal{J}^{(k,j-1)}\right) \\ + \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) \quad .\end{aligned} \tag{27}$$

Note that if $\operatorname{prox}_{\gamma g(\cdot,\lambda)}$ is not differentiable with respect to the first variable at $z^{(k)}$ (respectively with respect to the second variable λ), any weak Jacobian can be used. When (H3) holds, differentiating Equation (26) with respect to λ yields exactly Equation (27).

Assumptions 2 to 4 and 6 and the convergence of $(\beta^{(k)})_{k\in\mathbb{N}}$ toward $\hat{\beta}$ ensure proximal coordinate descent has finite identification property (Klopfenstein et al., 2020, Thm. 1): we note K the iteration when identification is achieved. Once the generalized support $\hat{S}$ (of cardinality $\hat{s}$) has been identified, we have that for all $k \geq K$, $\beta_{\hat{S}^c}^{(k)} = \hat{\beta}_{\hat{S}}$ and for any $j \in \hat{S}^c$, $\partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) = 0$. Thus $\mathcal{J}_{j:}^{(k,j)} = \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)})$. Then, we have that for any $j \in \hat{S}$ and for all $k \geq K$:

$$\begin{aligned}\mathcal{J}_{j:}^{(k,j)} = \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) \left(\mathcal{J}_{j:}^{(k,j-1)} - \gamma_j \nabla^2_{j,\hat{S}} f(\beta^{(k,j-1)}) \mathcal{J}_{\hat{S}:}^{(k,j-1)}\right) \\ + \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) - \gamma_j \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(z_j^{(k,j-1)}) \nabla^2_{j,\hat{S}^c} f(\beta^{(k,j-1)}) \mathcal{J}_{\hat{S}^c:}^{(k,j-1)} \quad .\end{aligned}$$

Let $e_1, \ldots, e_{\hat{s}}$ be the vectors of the canonical basis of $\mathbb{R}^{\hat{s}}$. We can consider the applications

$$\begin{aligned}&\mathbb{R}^p \to \mathbb{R}^{\hat{s}} \\ &\beta \mapsto \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)} \left(\beta_j - \gamma_j \nabla_j f(\beta)\right) \left(e_j - \gamma_j \nabla^2_{j,\hat{S}} f(\beta)\right) \quad ,\end{aligned}$$

and

$$\begin{aligned}&\mathbb{R}^p \to \mathbb{R}^{\hat{s}\times r} \\ &\beta \mapsto \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)} \left(\beta_j - \gamma_j \nabla_j f(\beta)\right) - \gamma_j \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)} \left(\beta_j - \gamma_j \nabla_j f(\beta)\right) \nabla^2_{j,\hat{S}^c} f(\beta) \hat{\mathcal{J}}_{\hat{S}^c:} \quad ,\end{aligned}$$

which are both differentiable at $\hat{\beta}$ using Assumption 6 and (H4). The Taylor expansion of the previous functions yields:

$$\begin{aligned}\mathcal{J}_{j:}^{(k,j)} &= \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(\hat{z}_j)\left(e_j - \gamma_j \nabla^2_{j,\hat{S}} f(\hat{\beta})\right)\mathcal{J}_{\hat{S}:}^{(k,j-1)} \\ &\quad + \partial_\lambda \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(\hat{z}_j) - \gamma_j \partial_z \operatorname{prox}_{\gamma_j g_j(\cdot,\lambda)}(\hat{z}_j)\,\nabla^2_{j,\hat{S}^c} f(\hat{\beta})\mathcal{J}_{\hat{S}^c:}^{(k,j-1)} \\ &\quad + o(||\beta^{(k,j-1)} - \hat{\beta}||) \ .\end{aligned}$$

Let $j_1, \dots, j_{\hat{s}}$ be the indices of the generalized support of $\hat{\beta}$. When considering a full epoch of coordinate descent, the Jacobian is obtained as the product of matrices of the form

$$A_s^\top = \left(\ e_1 \ \middle| \ \dots \ \middle| \ e_{s-1} \ \middle| \ v_{j_s} \ \middle| \ e_{s+1} \ \middle| \ \dots \ \middle| \ e_{\hat{s}} \ \right) \in \mathbb{R}^{\hat{s}\times\hat{s}} \ ,$$

where $v_{j_s} = \partial_z \operatorname{prox}_{\gamma_{j_s} g_{j_s}}(\hat{z}_{j_s})\left(e_s - \gamma_{j_s}\nabla^2_{j_s,\hat{S}} f(\hat{\beta})\right) \in \mathbb{R}^{\hat{s}}$. A full epoch can then be written

$$\mathcal{J}_{\hat{S}:}^{(k+1)} = \underbrace{A_{\hat{s}} A_{\hat{s}-1} \dots A_1}_{A} \mathcal{J}_{\hat{S}:}^{(k)} + b + \epsilon^{(k)} \ ,$$

for a certain $b \in \mathbb{R}^{\hat{s}}$.

The spectral radius of A is strictly bounded by 1 (Klopfenstein et al., 2020, Lemma 8): $\rho(A) < 1$. Assumptions 2 to 4 and 6 and the convergence of $(\beta^{(k)})_{k\in\mathbb{N}}$ toward $\hat{\beta}$ ensure local linear convergence of $(\beta^{(k)})_{k\in\mathbb{N}}$ (Klopfenstein et al., 2020, Thm. 2). Hence, we can write the update for the Jacobian after an update of the coordinates from 1 to p:

$$\mathcal{J}_{\hat{S}:}^{(k+1)} = A\mathcal{J}_{\hat{S}:}^{(k)} + b + \epsilon^{(k)} \ , \tag{28}$$

with $(\epsilon^{(k)})_{k\in\mathbb{N}}$ converging linearly to 0.

Recalling $\rho(A) < 1$, Lemma 19 and the last display yield our result using. ■

B.1 Proof of Theorem 13 (approximate hypergradients)

Theorem 13 (Bound on the error of approximate hypergradient) *For $\lambda \in \mathbb{R}^r$, let $\hat{\beta}^{(\lambda)} \in \mathbb{R}^p$ be the exact solution of the inner Problem (1), and $\hat{S}$ its generalized support. Suppose Assumptions 2, 3 and 6 hold. Let Λ be a neighborhood of λ, and $\Gamma^\Lambda \triangleq \left\{\hat{\beta}^{(\lambda)} - \gamma\nabla f(\hat{\beta}^{(\lambda)}) \ : \ \lambda \in \Lambda\right\}$. Suppose hypotheses (H1) to (H4) from Theorem 9 are satisfied. In addition suppose*

(H5) The application $\beta \mapsto \nabla^2 f(\beta)$ is Lipschitz continuous.

(H6) The criterion $\beta \mapsto \nabla\mathcal{C}(\beta)$ is Lipschitz continuous.

(H7) Both optimization problems in Algorithm 5 are solved up to precision ϵ with support identification: $\|\beta^{(\lambda)} - \hat{\beta}^{(\lambda)}\| \leq \epsilon$, $A^\top$ is invertible, and $\|A^{-1\top}\nabla_{\hat{S}}\mathcal{C}(\beta^{(\lambda)}) - v\| \leq \epsilon$.

Then the error on the approximate hypergradient h returned by Algorithm 5 is of the order of magnitude of the error ϵ on $\beta^{(\lambda)}$ and v:

$$\|\nabla\mathcal{L}(\lambda) - h\| = \mathcal{O}(\epsilon) \ .$$

Proof

Overview of the proof. Our goal is to bound the error between the approximate hypergradient h returned by Algorithm 5 and the true hypergradient $\nabla\mathcal{L}(\lambda)$. Following the analysis of Pedregosa (2016), two sources of approximation errors arise when computing the hypergradient:

- Approximation errors from the inexact computation of $\hat{\beta}$. Dropping the dependency with respect to λ, we denote β the approximate solution and suppose the problem is solved to precision ϵ with support identification (H7):

$$\begin{cases} \beta_{\hat{S}^c} = \hat{\beta}_{\hat{S}^c} \\ \|\beta_{\hat{S}} - \hat{\beta}_{\hat{S}}\| \leq \epsilon \ . \end{cases}$$

- Approximation errors from the approximate resolution of the linear system, using (H7) yields:

$$\|A^{-1\top}\nabla_{\hat{S}}\mathcal{C}(\beta) - v\| \leq \epsilon \ .$$

  The exact solution of the exact linear system $\hat{v}$ satisfies:

$$\hat{v} = \hat{A}^{-1\top}\nabla_{\hat{S}}\mathcal{C}(\hat{\beta}) \ ,$$

  with

$$A \triangleq \mathrm{Id}_{|\hat{S}|} - \underbrace{\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}(\beta - \gamma\nabla f(\beta))_{\hat{S}}}_{\triangleq C} \underbrace{\left(\mathrm{Id}_{|\hat{S}|} - \gamma\nabla^2_{\hat{S},\hat{S}} f(\beta)\right)}_{\triangleq D} \ ,$$

$$\hat{A} \triangleq \mathrm{Id}_{|\hat{S}|} - \underbrace{\partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)}\left(\hat{\beta} - \gamma\nabla f(\hat{\beta})\right)_{\hat{S}}}_{\triangleq \hat{C}} \underbrace{\left(\mathrm{Id}_{|\hat{S}|} - \gamma\nabla^2_{\hat{S},\hat{S}} f(\hat{\beta})\right)}_{\triangleq \hat{D}} \ .$$

- Using the last two points, the goal is to bound the difference between the exact hypergradient and the approximate hypergradient, $\|\nabla\mathcal{L}(\lambda) - h\|$. Following Algorithm 5, the exact hypergradient reads

$$\nabla\mathcal{L}(\lambda) = \hat{B}\hat{v} + \hat{\mathcal{J}}^\top_{\hat{S}^c:}\nabla_{\hat{S}^c}\mathcal{C}(\hat{\beta}) \ ,$$

and similarly for the approximate versions:

$$h = Bv + \mathcal{J}_{\hat{S}^c:}^\top \nabla_{\hat{S}^c} \mathcal{C}(\beta) \ ,$$

with

$$B \triangleq \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)} (\beta - \gamma \nabla f(\beta))_{\hat{S}:} - \gamma \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)} (\beta - \gamma \nabla f(\beta))_{\hat{S}} \odot \left(\nabla^2_{\hat{S},\hat{S}^c} f(\beta) \right) \hat{\mathcal{J}}_{\hat{S}^c:}$$
$$\hat{B} \triangleq \partial_\lambda \operatorname{prox}_{\gamma g(\cdot,\lambda)} \left(\hat{\beta} - \gamma \nabla f(\hat{\beta}) \right)_{\hat{S}:} - \gamma \partial_z \operatorname{prox}_{\gamma g(\cdot,\lambda)} \left(\hat{\beta} - \gamma \nabla f(\hat{\beta}) \right)_{\hat{S}} \odot \left(\nabla^2_{\hat{S},\hat{S}^c} f(\hat{\beta}) \right) \hat{\mathcal{J}}_{\hat{S}^c:} \ .$$

We can exploit these decompositions to bound the difference between the exact hypergradient and the approximate hypergradient:

$$\begin{aligned}
\|\nabla \mathcal{L}(\lambda) - h\| &= \|\hat{B}\hat{v} - Bv + \hat{\mathcal{J}}_{\hat{S}^c:}^\top \nabla_{\hat{S}^c} \mathcal{C}(\hat{\beta}) - \hat{\mathcal{J}}_{\hat{S}^c:}^\top \nabla_{\hat{S}^c} \mathcal{C}(\beta)\| \\
&\leq \|\hat{B}\hat{v} - Bv\| + \|\hat{\mathcal{J}}_{\hat{S}^c:}^\top \nabla_{\hat{S}^c} \mathcal{C}(\hat{\beta}) - \hat{\mathcal{J}}_{\hat{S}^c:}^\top \nabla_{\hat{S}^c} \mathcal{C}(\beta)\| \\
&\leq \|\hat{B}\hat{v} - B\hat{v} + B\hat{v} - Bv\| + \|\hat{\mathcal{J}}_{\hat{S}^c:}^\top (\nabla_{\hat{S}^c} \mathcal{C}(\hat{\beta}) - \nabla_{\hat{S}^c} \mathcal{C}(\beta))\| \\
&\leq \|\hat{v}\| \cdot \|\hat{B} - B\| + \|B\| \cdot \|\hat{v} - v\| + L_{\mathcal{C}} \|\hat{\mathcal{J}}_{\hat{S}^c:}^\top\| \cdot \|\beta - \hat{\beta}\| \ . \qquad (29)
\end{aligned}$$

Bounding $\|\hat{v} - v\|$ and $\|\hat{B} - B\|$ in Equation (29) yields the desired result which is bounding the difference between the exact hypergradient and the approximate hypergradient $\|\nabla \mathcal{L}(\lambda) - h\|$.

Bound on $\|\hat{v} - v\|$. We first prove that $\|A - \hat{A}\| = \mathcal{O}(\epsilon)$. Let L_H be the Lipschitz constant of the application $\beta \mapsto \nabla^2 f(\beta)$, then we have:

$$\begin{aligned}
\|A - \hat{A}\|_2 &= \|CD - \hat{C}\hat{D}\|_2 \\
&\leq \|CD - C\hat{D}\|_2 + \|C\hat{D} - \hat{C}\hat{D}\|_2 \\
&\leq \underbrace{\|C\|_2}_{\leq 1 \text{ (non-expansiveness)}} \underbrace{\|D - \hat{D}\|_2}_{\leq L_H \|\beta - \hat{\beta}\| \text{ using (H5)}} + \underbrace{\|\hat{D}\|_2}_{\leq 1} \underbrace{\|C - \hat{C}\|_2}_{\mathcal{O}(\|\beta - \hat{\beta}\|) \text{ using (H4)}} \\
&\leq L_H \|\beta - \hat{\beta}\| + \mathcal{O}(\|\beta - \hat{\beta}\|) \\
&= \mathcal{O}(\|\beta - \hat{\beta}\|) \ . \qquad (30)
\end{aligned}$$

Let $\tilde{v}$ be the exact solution of the approximate system $A^\top \tilde{v} \triangleq \nabla_{\hat{S}} \mathcal{C}(\beta)$. The following conditions are met:

- $\hat{v}$ is the exact solution of the exact linear system and $\tilde{v}$ is the exact solution of the approximate linear system

$$\hat{A}^\top \hat{v} \triangleq \nabla_{\hat{S}} \mathcal{C}(\hat{\beta})$$
$$A^\top \tilde{v} \triangleq \nabla_{\hat{S}} \mathcal{C}(\beta) \ .$$

- One can control the difference between the exact matrix in the linear system $\hat{A}$ and the approximate matrix A.

$$\|A - \hat{A}\|_2 \leq \delta \|\beta - \hat{\beta}\| \ ,$$

for a certain $\delta > 0$ (Equation (30)).

- One can control the difference between the two right-hand side of the linear systems

$$\|\nabla_{\hat{S}} \mathcal{C}(\beta) - \nabla_{\hat{S}} \mathcal{C}(\hat{\beta})\| \leq L_{\mathcal{C}} \|\beta - \hat{\beta}\| \ ,$$

since $\beta \mapsto \nabla \mathcal{C}(\beta)$ is $L_{\mathcal{C}}$-Lipschitz continuous (H6).

- One can control the product of the perturbations

$$\delta \cdot \|\beta - \hat{\beta}\| \cdot \|\hat{A}^{-1}\|_2 \leq \rho < 1 \ .$$

Conditions are met to apply the result by Higham (2002, Thm 7.2), which leads to

$$\begin{aligned}
\|\tilde{v} - \hat{v}\| &\leq \frac{\epsilon}{1 - \epsilon \|\hat{A}^{-1}\| \delta} \left(L_{\mathcal{C}} \|\hat{A}^{-1}\| + \|\hat{v}\| \cdot \|\hat{A}^{-1}\| \delta \right) \\
&\leq \frac{\epsilon}{1 - \rho} \left(L_{\mathcal{C}} \|\hat{A}^{-1}\| + \|\hat{v}\| \cdot \|\hat{A}^{-1}\| \delta \right) \\
&= \mathcal{O}(\epsilon) \ . \qquad (31)
\end{aligned}$$

The bound on $\|\tilde{v} - \hat{v}\|$ finally yields a bound on the first quantity in Equation (3), $\|v - \hat{v}\|$:

$$\begin{aligned}
\|v - \hat{v}\| &= \|v - \tilde{v} + \tilde{v} - \hat{v}\| \\
&\leq \|v - \tilde{v}\| + \|\tilde{v} - \hat{v}\| \\
&\leq \|A^{-1} A (v - \tilde{v})\| + \|\tilde{v} - \hat{v}\| \\
&\leq \|A^{-1}\|_2 \times \underbrace{\|A(v - \tilde{v})\|}_{\leq \epsilon \text{ (H7)}} + \underbrace{\|\tilde{v} - \hat{v}\|}_{\mathcal{O}(\epsilon) \text{ (Equation (31))}} \\
&= \mathcal{O}(\epsilon) \ . \qquad (32)
\end{aligned}$$

Bound on $\|B - \hat{B}\|_2$. We now bound the second quantity in Equation (3) $\|B - \hat{B}\|_2$:

$$\begin{aligned}
\|B - \hat{B}\|_2 &\leq \|\partial_\lambda \operatorname{prox}_{\gamma g(\cdot, \lambda)}(\beta - \gamma \nabla f(\beta))_{\hat{S}:} - \partial_\lambda \operatorname{prox}_{\gamma g(\cdot, \lambda)}(\hat{\beta} - \gamma \nabla f(\hat{\beta}))_{\hat{S}:}\|_2 \\
&\quad + \gamma \|\partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)}(\hat{\beta} - \gamma \nabla f(\hat{\beta}))_{\hat{S}} \nabla^2_{\hat{S}, \hat{S}^c} f(\hat{\beta}) \hat{\mathcal{J}}_{\hat{S}^c:} - \partial_z \operatorname{prox}_{\gamma g(\cdot, \lambda)}(\beta - \gamma \nabla f(\beta))_{\hat{S}} \nabla^2_{\hat{S}, \hat{S}^c} f(\beta) \hat{\mathcal{J}}_{\hat{S}^c:}\|_2 \\
&\leq L_1 \|\beta - \gamma \nabla f(\beta)_{\hat{S}:} - \hat{\beta} + \gamma \nabla f(\hat{\beta})\| \text{ using (H4)} \\
&\quad + L_2 \|\hat{\beta} - \beta\| \cdot \|\hat{\mathcal{J}}_{\hat{S}^c:}\| \text{ using (H4) and Assumption 6} \\
&= \mathcal{O}(\|\hat{\beta} - \beta\|) \ . \qquad (33)
\end{aligned}$$

Plugging Equations (32) and (33) into Equation (3) yields the desired result: $\|\nabla \mathcal{L}(\lambda) - h\| = \mathcal{O}(\epsilon)$. ■

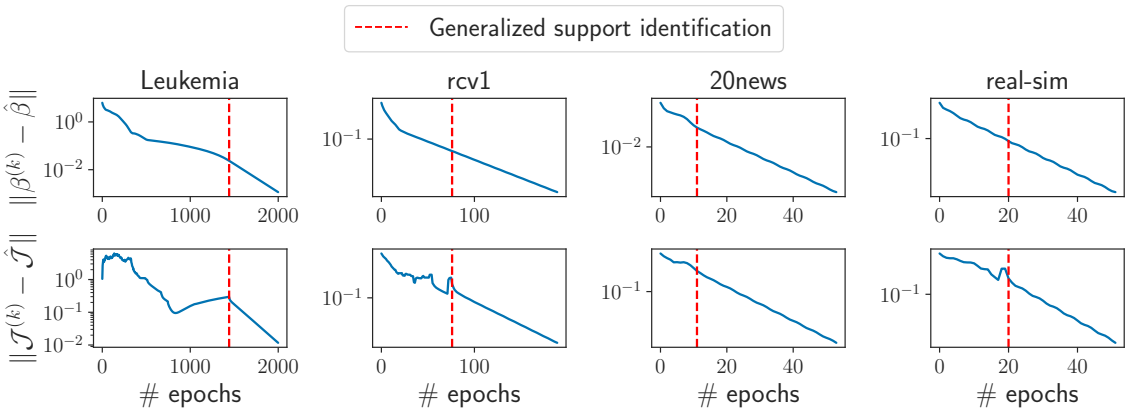

Figure 10: **Local linear convergence of the Jacobian for the Lasso.** Distance to optimum for the coefficients β (top) and the Jacobian $\mathcal{J}$ (bottom) of the forward-mode differentiation of proximal coordinate descent (Algorithm 3) on multiple datasets.

Table 6: Dataset characteristics and regularization parameters used in Figures 3, 10 and 11.

Datasets	*leukemia*	*rcv1*	*news20*	*real-sim*
# samples	$n = 38$	$n = 20{,}242$	$n = 19{,}996$	$n = 72{,}309$
# features	$p = 7{,}129$	$p = 19{,}959$	$p = 632{,}982$	$p = 20{,}958$
Lasso	$e^{\lambda} = 0.01\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.075\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.3\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.1\, e^{\lambda_{\max}}$
Logistic regression	$e^{\lambda} = 0.1\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.25\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.8\, e^{\lambda_{\max}}$	$e^{\lambda} = 0.15\, e^{\lambda_{\max}}$
SVM	$e^{\lambda} = 10^{-5}$	$e^{\lambda} = 3 \times 10^{-2}$	$e^{\lambda} = 10^{-3}$	$e^{\lambda} = 5 \times 10^{-2}$

Appendix C. Additional experiments

Figures 10 and 11 are the counterparts of Figure 3 for the Lasso and sparse logistic regression. It shows the local linear convergence of the Jacobian for the Lasso, obtained by the forward-mode differentiation of coordinate descent. The solvers used to determine the exact solution up to machine precision are `Celer` (Massias et al., 2018, 2020) for the Lasso and `Blitz` (Johnson and Guestrin, 2015) for the sparse logistic regression. Table 6 summarizes the values of the hyperparameters λ used in Figures 3, 10 and 11.

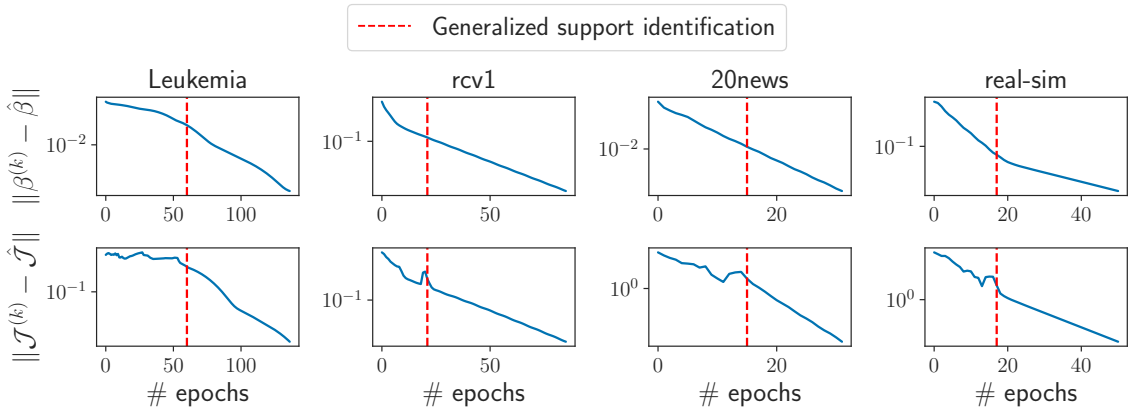

Figure 11: **Local linear convergence of the Jacobian for sparse logistic regression.** Distance to optimum for the coefficients β (top) and the Jacobian $\mathcal{J}$ (bottom) of the forward-mode differentiation of proximal coordinate descent (Algorithm 3) on multiple datasets.