European Journal of Control 81 (2025) 101150

Contents lists available at ScienceDirect

European Journal of Control

journal homepage: www.sciencedirect.com/journal/european-journal-of-control

Gradient-based bilevel optimization for multi-penalty Ridge regression through matrix differential calculus[☆]

Gabriele Maroni [*], Loris Cannelli, Dario Piga

SUPSI, IDSIA Dalle Molle Institute for Artificial Intelligence Research, Via la Santa 1, Lugano, 6900, Switzerland

A R T I C L E I N F O

Recommended by T. Parisini

Keywords:
Regularization
Regularized least-squares
Ridge regression
Gradient-based optimization
Multi-hyperparameter optimization
Bilevel optimization
Multi-penalty Ridge regression

A B S T R A C T

Common regularization algorithms for linear regression, such as LASSO and Ridge regression, rely on a regularization hyperparameter that balances the trade-off between minimizing the fitting error and the norm of the learned model coefficients. As this hyperparameter is scalar, it can be easily selected via random or grid search optimizing a cross-validation criterion. However, using a scalar hyperparameter limits the algorithm's flexibility and potential for better generalization. In this paper, we address the problem of linear regression with ℓ_2-regularization, where a different regularization hyperparameter is associated with each input variable. We optimize these hyperparameters using a gradient-based approach, wherein the gradient of a cross-validation criterion with respect to the regularization hyperparameters is computed analytically through matrix differential calculus. Additionally, we introduce two strategies tailored for sparse model learning problems aiming at reducing the risk of overfitting to the validation data. Numerical examples demonstrate that the proposed multi-hyperparameter regularization approach outperforms LASSO, Ridge, and Elastic Net regression in terms of R^2 score both in a static regression and in a system identification problem. Moreover, the analytical computation of the gradient proves to be more efficient in terms of computational time compared to automatic differentiation, especially when handling a large number of input variables, with an improvement of more than an order of magnitude. Application to the identification of over-parameterized Linear Parameter-Varying models is also presented.

1. Introduction

1.1. Research problem

Linear regression problems often require regularization of the model parameters for two main reasons: (*i*) to ensure better numerical conditioning in cases of multicollinearity (i.e., when input variables are highly correlated); (*ii*) to enhance the generalization capabilities of the estimated model, as regularization drives the model coefficients towards zero, thereby reducing model complexity and preventing overfitting.

The most widely used regularization techniques for linear regression are the LASSO (Tibshirani, 1996) and the Ridge algorithm (Hoerl & Kennard, 1970). These techniques penalize the ℓ_1 and the ℓ_2 norm of the model coefficients, respectively. In the simplest case, the importance of the penalty term is governed by a scalar hyperparameter, which controls the balance between fitting the training data and enforcing a penalty on the norm of the model coefficients. The use of a singular scalar hyperparameter simplifies its selection, usually accomplished through random or grid search within a cross-validation framework. However, relying solely on a single hyperparameter presents certain limitations. Specifically, it applies the same regularization strength to all input variables without considering the varied importance or relevance of different features. Furthermore, regularization inherently introduces bias by shrinking model coefficients towards zero, and relying on just one hyperparameter limits the flexibility in managing the bias–variance trade-off. In cases involving inherently sparse models, where only a few input variables are relevant, it may be more suitable to encourage only the model parameters associated with non-relevant variables to approach zero, rather than penalizing the other model coefficients as much or at all.

While addressing these limitations by introducing multiple hyperparameters, potentially assigning one to each input feature, seems theoretically appealing, it comes with challenges. The curse of dimensionality becomes apparent, and the risk of overfitting increases. The search space for hyperparameters grows significantly, making grid search

☆ This work was partially supported by the European Union, Grant Agreement n. 101093126 (project ACES: Autopoiesis Cognitive Edge-cloud Services) and by the Hasler Foundation, Switzerland (project ACCIPEAK: Accurate and Interpretable Deep Learning models for Peak Load Forecasting).

* Corresponding author.
E-mail address: gabriele.maroni@idsia.ch (G. Maroni).

https://doi.org/10.1016/j.ejcon.2024.101150
Received 26 June 2024; Received in revised form 4 November 2024; Accepted 10 November 2024
Available online 19 November 2024

or random search strategies practically infeasible. Even more scalable gradient-free algorithms, such as Bayesian optimization (Snoek, Larochelle, & Adams, 2012), are generally limited to handling a maximum of roughly 50 hyperparameters. Paradoxically, while regularization aims to prevent overfitting, introducing numerous hyperparameters, such as a regularization coefficient for each feature, can inadvertently lead to overfitting the validation set during hyperparameter tuning.

1.2. Contribution

This paper focuses on linear regression with an ℓ_2 penalty on the norm of the model coefficients, employing multiple hyperparameters. We consider the general scenario where each input variable is associated with its unique hyperparameter. Essentially, our problem can be seen as a generalization of Ridge regression with multiple hyperparameters. Throughout this paper, we will refer to this problem as *multi-ridge regression*, adopting the terminology in van de Wiel, van Nee, and Rauschenberger (2021). To select these hyperparameters, we employ a gradient-based approach that optimizes a cross-validation criterion. The gradient with respect to the regularization hyperparameters is analytically computed using *matrix differential calculus*. This computation leverages efficient matrix–vector multiplication available in modern scientific computing libraries such as NumPy, PyTorch, and JAX.

As outlined in Bengio (2000), optimizing a large number of hyperparameters through cross-validation can lead to overfitting to the validation set. To mitigate this risk, we introduce two strategies that are particularly suitable for solving sparse model learning problems.

In the numerical examples, we will demonstrate that using multiple regularization hyperparameters enhances the performance compared to single-hyperparameter Ridge regression. Furthermore, it outperforms both LASSO and Elastic Net, even in the context of sparse model learning problems.

To ensure the reproducibility of our results and to encourage further contributions to the field, we have made a PyTorch implementation of our proposed approach. This implementation is tailored with a scikit-learn compatible interface, ensuring seamless integration into existing workflows. Both the implementation and the results presented in this paper are freely available on the GitHub repository at the following link: https://github.com/gabribg88/Multiridge.

1.3. Related works

The problem of Ridge regression with multiple penalties was early proposed in Hoerl and Kennard (1970). A gradient-based approach to optimize the algorithm's hyperparameter was later addressed in the seminal work (Bengio, 2000). The author here proposes to solve the inefficient step of computing the gradient of the problem variables with respect to the hyperparameters through a combination of Cholesky decomposition and backpropagation. The approach introduced in Bengio (2000), later implemented in Latendresse and Bengio (2000), has been then extensively explored in subsequent studies. For instance, the works (Bertrand et al., 2022; Franceschi, Donini, Frasconi, & Pontil, 2017; Hataya, Zdenek, Yoshizoe, & Nakayama, 2022; Lorraine, Vicol, & Duvenaud, 2020; Pedregosa, 2016) propose efficient solutions to compute the computationally expensive gradient step mentioned in Bengio (2000). These solutions employ approximators (e.g., Neumann series as demonstrated in Hataya et al. (2022), Lorraine et al. (2020)) or iterative procedures (e.g., conjugate gradient in Pedregosa (2016), reverse-hyper gradient in Franceschi et al. (2017), or proximal non-smooth techniques in Bertrand et al. (2022)). Although these approaches provide approximate solutions, they have proven to be satisfactory, particularly for machine learning tasks, as discussed (Blondel et al., 2022; Pedregosa, 2016).

Another research avenue that has gained momentum from Bengio (2000) is the exploration of optimizing regularization hyperparameters by targeting specific machine-learning problems. By selecting a particular machine learning application, researchers can frame the task of finding the optimal variable fitting, while simultaneously determining the best configuration for the hyperparameters, as a bilevel optimization problem. This allows for tailored approaches to effectively address the challenge. Interesting tools have been proposed, among others, in Bennett, Hu, Ji, Kunapuli, and Pang (2006) for regularized Support Vector Machines, in Do, Foo, and Ng (2007) for log-linear models, in Frecon, Salzo, and Pontil (2018) for the group-LASSO setting, and in Kunisch and Pock (2013) for Ridge and LASSO regularizations. An alternative approach to nested optimization for selecting the multi-ridge hyperparameters is based on maximization of the marginal likelihood and discussed in van de Wiel et al. (2021).

In relation to the existing literature, we present a custom approach to linear regression with ℓ_2 regularization, that enhances flexibility and generalization by assigning unique regularization hyperparameters to each input variable. The proposed approach is a generalization of Ridge regression, and allows to perform model structure selection as LASSO (as discussed in Section 3.3). The novelty of the proposed approach with respect to the previous literature is the introduction of a framework for the optimization of these hyperparameters through an efficient gradient-based method using matrix differential calculus, making it practically applicable, and the incorporation of tailored mitigation strategies to prevent overfitting.

A preliminary short version of this paper, which focuses more on the identification of dynamical systems and does not include detailed derivations of the results, is presented in Maroni, Cannelli, and Piga (2024).

1.4. Paper organization

The paper is organized as follows. Section 2 introduces the mathematical notation used throughout the paper. Section 3 presents the multi-hyperparameter regression problem addressed in our contribution, along with an intuitive motivation of the advantages of using multiple regularization hyperparameters over a single one, especially in sparse model learning problems. In Section 4, we present the analytical computation of the gradient of the cross-validation criterion with respect to the regularization hyperparameters. Detailed derivations can be found in the Appendix to facilitate a smooth reading flow. Section 5 discusses the two strategies we propose to reduce the risk of overfitting to the validation set. Section 6 provides numerical examples showcasing the effectiveness of our approach. One of these examples is focused on the identification of over-parameterized Linear Parameter-Varying models, a topic previously tackled in the literature using LASSO-like and kernel-based algorithms (see, e.g., Laurain, Tóth, Piga, and Darwish (2020), Piga and Tóth (2013), Tóth, Hjalmarsson, and Rojas (2012)). Finally, Section 7 offers conclusions and directions for future research.

2. Notation

We denote with $\otimes$ the Kronecker product and with $\odot$ the elementwise matrix multiplication. We denote with $\mathrm{Diag} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$, $v \mapsto \mathrm{Diag}(v)$, the operator that maps a column vector $v \in \mathbb{R}^n$ in a $n \times n$ diagonal matrix $\mathrm{Diag}(v)$ having the elements of v itself on the main diagonal and zeros otherwise; conversely, we denote with $\mathrm{diag} : \mathbb{R}^{n\times n} \to \mathbb{R}^n$, $V \mapsto \mathrm{diag}(V)$ the operator that maps a matrix $V \in \mathbb{R}^{n\times n}$ in $\mathrm{diag}(V)$, that is the column vector $v \in \mathbb{R}^n$ collecting the entries of the main diagonal of V. $\mathrm{vec}(\cdot)$ denotes the vectorization operator that stacks the columns of a matrix vertically to form a column vector. Given a matrix $A \in \mathbb{R}^{m\times n}$, $\|A\|_F$ denotes its Frobenius norm, and A' the transpose of A. Finally, we indicate with I_N the identity matrix of size N.

Given a matrix $U \in \mathbb{R}^{m\times n}$ and a real-valued function $\phi : \mathbb{R}^{m\times n} \to \mathbb{R}$, $U \mapsto \phi(U)$, the gradient of ϕ w.r.t. the vectorized version of U ($\mathrm{vec}(U)$)

is denoted as $\nabla_U \phi$, with $\nabla_U \phi \in \mathbb{R}^{mn}$. If $U \in \mathbb{R}^{m\times m}$ is a diagonal matrix, with diagonal elements $u = \mathrm{diag}(U) \in \mathbb{R}^m$, $\nabla_u \phi \in \mathbb{R}^m$ is the gradient of the function ϕ w.r.t. the diagonal elements of the matrix U.

Given a finite-dimensionals set $\mathcal{S}$, $|\mathcal{S}|$ is its cardinality. Finally, with $\{\alpha^{(i)}\}_{i=1}^n$ we denote a sequence of elements indexed from 1 to n, i.e., $\{\alpha^{(1)},\dots,\alpha^{(n)}\}$, where $\alpha^{(i)}$ can also be a tuple.

3. Problem formulation

3.1. Parametric estimate: setting

In a parametric supervised learning setting, we aim to select a function f that maps input features $x \in \mathcal{X}$ to output targets $y \in \mathcal{Y}$ from a set $\mathcal{F}$ of candidate functions, parameterized by a vector of parameters θ, such that each value assumed by the vector of parameters θ corresponds to a different function in the set $\mathcal{F}$. In this work we focus on regression problems, and we assume $\mathcal{X} = \mathbb{R}^D$, with D being the number of input features; and $\mathcal{Y} = \mathbb{R}$ in the case of simple regression with a real-valued target, or $\mathcal{Y} = \mathbb{R}^M$ in the case of multi-task/target regression, with M denoting the number of real-valued target variables.

In a real-world setting, we are given a dataset of N input–output pairs $\mathcal{D} = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^N$ which are assumed to be independently drawn from the same unknown probability distribution $\mathbb{P}_{x,y}$. In principle, we would like to select the values of θ (and hence the function $f_\theta \in \mathcal{F}$) which minimize the expected risk, defined as the expected value $\mathbb{E}_{x,y}[\ell(y, f_\theta)]$ of some scalar loss function $\ell(y, f_\theta)$, which is a measure of the discrepancy between the target and the model's prediction. In general we do not have access to the data distribution $\mathbb{P}_{x,y}$, so we approximate the expected risk with an empirical risk $L(\theta)$ defined as the average loss over the dataset $\mathcal{D}$:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(y^{(i)}, f_\theta\left(x^{(i)}\right)\right). \tag{1}$$

In solving regression problems, a regularization term $R(\theta,\lambda)$ is typically added to control the complexity of the estimated function f_θ. Such a regularization term is parameterized by a vector of hyperparameters λ that regulate the trade-off between the fit and the regularization term.

The overall training criterion $C(\theta,\lambda)$ is then defined as:

$$C(\theta,\lambda) = L(\theta) + R(\theta,\lambda), \tag{2}$$

and, for a given value of λ, the optimal parameterized function is the solution of an optimization problem, i.e.,

$$\hat{\theta}(\lambda) = \arg\min_\theta C(\theta,\lambda). \tag{3}$$

Note that $\hat{\theta}$ is a function mapping the hyperparameter vector λ into the vector of parameters $\hat{\theta}(\lambda)$. Among different possibilities, the hyperparameters λ can be chosen by optimizing an evaluation criterion $E(\hat{\theta}(\lambda))$, such as minimizing the average loss over a new dataset that was not used in evaluating $L(\theta)$. For example, in the celebrated K-fold cross-validation strategy, the original dataset $\mathcal{D}$ is divided into K partitions $\mathcal{D}_1,\dots,\mathcal{D}_K$, and, consequently, the evaluation criterion takes the form:

$$E\left(\hat{\theta}(\lambda)\right) = \frac{1}{K}\sum_{k=1}^{K} L_k\left(\hat{\theta}^{(\setminus k)}(\lambda)\right), \tag{4}$$

where L_k is the average loss over the subset $\mathcal{D}_k$ (namely, *validation fold*). Specifically:

$$L_k\left(\hat{\theta}^{(\setminus k)}(\lambda)\right) = \frac{1}{|\mathcal{D}_k|}\sum_{(x^{(i)},y^{(i)})\in\mathcal{D}_k} \ell\left(y^{(i)}, f_{\theta^{(\setminus k)}(\lambda)}\left(x^{(i)}\right)\right), \tag{5}$$

and $\hat{\theta}^{(\setminus k)}(\lambda)$ represents the estimate of the model parameters θ obtained from constructing $C(\theta,\lambda)$ using all remaining folds $\mathcal{D}_j$ (with $j = 1,\dots,K$ and $j \neq k$), i.e.,

$$\hat{\theta}^{(\setminus k)}(\lambda) = \arg\min_\theta \frac{1}{|\bigcup_{j\neq k}\mathcal{D}_j|}\sum_{\substack{(x^{(i)},y^{(i)})\\ \in\bigcup_{j\neq k}\mathcal{D}_j}} \ell\left(y^{(i)}, f_\theta\left(x^{(i)}\right)\right) + R(\theta,\lambda). \tag{6}$$

Thus, the hyperparameters λ are selected by solving the minimization problem:

$$\lambda^\star = \arg\min_\lambda E\left(\hat{\theta}(\lambda)\right). \tag{7}$$

The hyperparameter optimization problem is thus a bilevel optimization problem, and the optimal vector of hyperparameters $\lambda^\star$ is obtained as:

$$\begin{aligned}\lambda^\star = \arg\min_\lambda \quad & E\left(\hat{\theta}(\lambda)\right)\\ \text{s.t.} \quad & \hat{\theta}(\lambda) = \arg\min_\theta C(\theta,\lambda),\end{aligned} \tag{8}$$

wherein the objective function of the outer optimization problem (7) depends on the solution of the inner optimization task (3).

3.2. Linear-in-the parameter models and quadratic loss

In this work, we consider:

- as the set of candidate functions, linear-in-the-parameter functions of the form

$$f_\Theta(x) = x'\Theta, \tag{9}$$

where $\Theta \in \mathbb{R}^{D\times M}$ is the coefficient matrix and $f_\Theta(x)$ is a row vector of size M whose its ith component represents the model prediction for the ith output;
- a quadratic fitting loss:

$$\ell\left(y^{(i)}, f_\Theta\left(x^{(i)}\right)\right) = \frac{1}{2}\left\|y^{(i)} - \left(x^{(i)}\right)'\Theta\right\|^2; \tag{10}$$

- a quadratic regularization term:

$$R(\Theta,\lambda) = \frac{1}{2}\|\Lambda\Theta\|_F^2, \tag{11}$$

where $\Lambda = Diag(\lambda)$, and $\lambda \in \mathbb{R}^D$ is the vector of hyperparameters.

Thus, the general expressions (6) and (7) of the K-fold cross-validation specialized, respectively, for the above choices (9)–(10)–(11), become:

$$\hat{\Theta}^{(\setminus k)}(\Lambda) = \arg\min_\Theta \frac{1}{2|\bigcup_{j\neq k}\mathcal{D}_j|}\sum_{\substack{(x^{(i)},y^{(i)})\\ \in\bigcup_{j\neq k}\mathcal{D}_j}} \left\|y^{(i)} - \left(x^{(i)}\right)'\Theta\right\|^2 + \frac{\|\Lambda\Theta\|_F^2}{2}, \tag{12}$$

and

$$\Lambda^\star = \arg\min_\Lambda \frac{1}{2K}\sum_{k=1}^{K}\frac{1}{|\mathcal{D}_k|}\sum_{(x^{(i)},y^{(i)})\in\mathcal{D}_k}\left\|y^{(i)} - \left(x^{(i)}\right)'\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|^2. \tag{13}$$

To ease readability, without any loss of generality, in the rest of the paper we assume that all cross-validation folds have equal size, namely: $|\mathcal{D}_1| = \cdots = |\mathcal{D}_K| = N_V = \frac{N}{K}$. This implies that the amount of samples used in the validation step (see (13)) of each cycle of the cross-validation is $\frac{N}{K}$, while $N_T = (K-1)\frac{N}{K}$ is the number of samples used to estimate the model parameters $\hat{\Theta}^{(\setminus k)}(\Lambda)$ in (12).

To leverage the matrix differential calculus approach, presented in Section 4, and to enhance notation clarity, we introduce the following matrix notation. Let $X_{\setminus k} \in \mathbb{R}^{N_T\times D}$ and $Y_{\setminus k} \in \mathbb{R}^{N_T\times M}$ be matrices stacking in their rows, respectively, the input samples $x^{(i)}$ and the output samples $y^{(i)}$, with $\left(x^{(i)}, y^{(i)}\right) \in \bigcup_{j\neq k}\mathcal{D}_j$. Similarly, let $X_k \in \mathbb{R}^{N_V\times D}$ and $Y_k \in \mathbb{R}^{N_V\times M}$ be matrices stacking in their rows, respectively, the input samples $x^{(i)}$ and the output samples $y^{(i)}$, with $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{D}_k$.

Based on the introduced notation, the model parameter estimate (12) can be rewritten in the following compact Tikhonov regularization form:

$$\begin{aligned}\hat{\Theta}^{(\setminus k)}(\Lambda) &= \arg\min_{\Theta} C(\Theta, \Lambda) \\ &= \arg\min_{\Theta} L(\Theta) + R(\Theta, \Lambda) \\ &= \arg\min_{\Theta} \frac{1}{2N_T} \left\| Y_{\setminus k} - X_{\setminus k}\Theta \right\|_F^2 + \frac{\|\Lambda\Theta\|_F^2}{2},\end{aligned} \tag{14}$$

while the optimal hyperparameters Λ in (13) can be expressed as:

$$\begin{aligned}\Lambda^{\star} &= \arg\min_{\Lambda} E\left(\hat{\Theta}(\Lambda)\right) \\ &= \arg\min_{\Lambda} \frac{1}{K}\sum_{k=1}^{K} L_k\left(\hat{\Theta}^{(\setminus k)}(\Lambda)\right) \\ &= \arg\min_{\Lambda} \frac{1}{K}\sum_{k=1}^{K} \frac{1}{2N_V}\left\| Y_k - X_k\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|_F^2 .\end{aligned} \tag{15}$$

3.3. Motivation

The main motivation behind the optimization of multiple hyperparameters stems from the well-known fact that conventional regularization techniques, such as LASSO, Ridge regression, or Elastic Net, reduce the variance of the estimate while concurrently introducing bias in the estimation of model parameters, by shrinking them closer to zero. In a intuitive sense, employing multiple hyperparameters enables variance reduction of the estimate variance, while upholding an unbiased estimation of the model parameters.

In order to intuitively explain the concept, let us consider an illustrative example where the output variable y is scalar and generated according to:

$$y = x'\Theta_o + e, \tag{16}$$

where e is white noise, independent of the input x, and $\Theta_o \in \mathbb{R}^D$ denotes the "true" (unknown) parameter vector. Let us take a scalar $\bar{\lambda} \in \mathbb{R}$, $\bar{\lambda} \neq 0$, and let $\Theta_{o,j}$ be the jth element of Θ_o. If the true parameter vector Θ_o is assumed to be sparse, then a hyperparameter matrix $\Lambda \in \mathbb{R}^{D\times D}$ with diagonal elements

$$\lambda_j = \begin{cases} \bar{\lambda} & \text{if } \Theta_{o,j} = 0 \\ 0 & \text{if } \Theta_{o,j} \neq 0 \end{cases}, \quad j = 1, \dots, D, \tag{17}$$

would regularize (thus shrinking towards zero) the model parameters Θ_j such that the corresponding true parameters are $\Theta_{o,j} = 0$. On the other hand, it would not shrink towards zero the model parameters Θ_j such that $\Theta_{o,j} \neq 0$. Hence, this type of regularization does not introduce bias into the model while it reduces the variance of the estimate compared to non-regularized approaches. Furthermore, for $\bar{\lambda} \to \infty$, $\Theta_j \to 0$, and thus input variable selection is also performed without introducing model bias.

Nevertheless, it is important to note, as also discussed in Bengio (2000), that using multiple hyperparameters λ introduces greater adaptability in the average loss $L_k\left(\hat{\theta}^{(\setminus k)}(\lambda)\right)$ in (5). More flexibility could potentially increase the risk of overfitting to the validation data employed for hyperparameter selection. In Section 5, we discuss two strategies aimed at mitigating the risk of overfitting in the case where the true underlying parameter matrix Θ_o is sparse.

4. Gradient-based multi-hyperparameter optimization

In this section we show how to compute the gradient $\nabla_{\Lambda} E$ of the evaluation criterion E (appearing as a cost function of the outer optimization problem (15)) with respect to the hyperparameter matrix Λ. This allows us to compute the optimal hyperparameters $\Lambda^{\star}$ through any gradient-based numerical optimization algorithm, such as Vanilla (stochastic) gradient descent; ADAM (Kingma, 2014), Adagrad (Duchi, Hazan, & Singer, 2011), RMSProp, etc. We point out that while global optimality cannot be guaranteed, nevertheless gradient descent is a well-established method for finding local optima, which can still provide valuable and practical solutions, especially when the objective function is non-convex (Murphy, 2022), as in the problem at hand. Moreover, to further reduce the risk of being trapped in local optima that could lead to poor generalization properties, the outer problem in (15) can be solved using different initial conditions for Λ. Nevertheless, in the experiments we conducted, we observed that the approach is quite robust to the choice of initial conditions. A practical choice for the initial condition, which enabled fast convergence to a satisfactory solution, can be guided by the solution of a Lasso problem, as discussed in the simulation example in Section 6.3.

The estimation of the model parameters $\hat{\Theta}^{(\setminus k)}(\Lambda)$ in the Tikhonov regularized form (14) has the following analytical solution:

$$\hat{\Theta}^{(\setminus k)}(\Lambda) = \left(X'_{\setminus k}X_{\setminus k} + N_T\Lambda\Lambda\right)^{-1} X'_{\setminus k}Y_{\setminus k}, \tag{18}$$

where the transpose of Λ is omitted in the equation above since Λ is diagonal, thus symmetric.

The gradient $\nabla_{\Lambda} E$ can be readily computed via automatic differentiation (e.g., using the deep learning Python libraries such as PyTorch or TensorFlow). Nevertheless, employing backpropagation to solve the Tikhonov regularization problem (14) was observed to be inefficient when handling a growing number of features, as also discussed in the numerical example in Section 6.2. For this reason, an analytical strategy for gradient computation is adopted in this paper. The proposed approach leverages the framework of matrix differential calculus (Magnus & Neudecker, 2019), enabling the derivation of gradients in a matrix form, thereby exploiting the efficient matrix–vector multiplication features available in scientific computing libraries such as NumPy, PyTorch, and JAX.

The following expression for the gradient $\nabla_{\Lambda} E$ is derived:

$$\nabla_{\Lambda} E = -\frac{N_T}{KN_V}\sum_{k=1}^{K} \operatorname{vec}\left(\Lambda B_k + B_k\Lambda\right), \tag{19a}$$

with $B_k = A'_k X'_k R_k \left(\hat{\Theta}^{(\setminus k)}\right)'$, $A_k = \left(X'_{\setminus k}X_{\setminus k} + N_T\Lambda\Lambda\right)^{-1}$, and $R_k = X_k\hat{\Theta}^{(\setminus k)}(\Lambda) - Y_k$.

Since Λ is constrained to be diagonal, at each update step of gradient-descent, only the gradient of the evaluation criterion E with respect to the diagonal elements λ of the hyperparameter matrix Λ is of practical interest, and its expression is given by:

$$\nabla_{\lambda} E = -\frac{N_T}{KN_V}\sum_{k=1}^{K} \operatorname{diag}\left(\Lambda B_k + B_k\Lambda\right). \tag{19b}$$

All the details behind the derivation of the gradients (19a) and (19b) are reported in Appendix A.

5. Reducing the risk of overfitting

Assuming that the true parameter matrix Θ_o is sparse, this section introduces two strategies with the objective of guiding the hyperparameters Λ towards the form outlined in (17). These strategies aim to mitigate the potential of overfitting the hyperparameters Λ during the cross-validation stage.

5.1. Optimal-solution augmentation

The strategy discussed in this subsection is based on the observation that any non-zero scaling of the hyperparameter matrix Λ does not affect its "optimal" structure in (17). Thus, given: (i) true model parameters Θ_o, and (ii) the optimal structure of $\Lambda^{\star}$ from (17), (i) and (ii) implies that the term $\|\gamma\Lambda^{\star}\Theta_o\|_F^2$, equals 0 for any scaling parameter $\gamma \in \mathbb{R}$ ($\gamma \neq 0$). Nevertheless, the model parameters Θ_j associated with zero elements of the true parameters $\Theta_{o,j}$ (or equivalently associated with non-zero hyperparameter $\lambda_j^\star$) remain subject to regularization towards zero.

To promote the sparse structure of $\Lambda^\star$ in (17), we consider a finite set Γ of non-zero scaling parameters γ. The optimal hyperparameters $\Lambda^\star$ are then computed as follows:

$$\Lambda^\star = \arg\min_{\Lambda} \frac{1}{2N}\frac{1}{|\Gamma|}\sum_{k=1}^{K}\sum_{\gamma\in\Gamma}\left\|Y_k - X_k\hat{\Theta}^{(\setminus k)}(\gamma\Lambda)\right\|_F^2, \tag{20}$$

with model parameters given by:

$$\hat{\Theta}^{(\setminus k)}(\gamma\Lambda) = \tag{21}$$
$$\arg\min_{\Theta} \frac{1}{2N_T}\sum_{\substack{(x^{(i)},y^{(i)})\\ \in\bigcup_{j\neq k}\mathcal{D}_j}}\left\|y^{(i)} - \left(x^{(i)}\right)'\Theta\right\|^2 + \frac{1}{2}\|\gamma\Lambda\Theta\|_F^2.$$

This approach bears similarity to data augmentation, a well-established technique often employed in deep learning tasks such as image recognition. Data augmentation involves introducing synthetic images (generated through operations like, for instance, rotation or adjustments in illumination) to the training dataset. This helps enforce desirable properties such as invariance to rotations and illumination. Instead of adding artificial data, our approach ensures that the hyperparameters Λ are not just adjusted to match the data, but rather ensure a small fitting error $\left\|Y_k - X_k\hat{\Theta}^{(\setminus k)}(\gamma\Lambda)\right\|_F^2$ in (20), for any $\gamma\in\Gamma$, as anticipated by the theoretical optimal hyperparameters $\Lambda^\star$ in (17). In other words, the optimal model parameters $\Theta^{(\setminus k)}(\gamma\Lambda)$ are expected to be invariant to any non-zero scaling factor γ. To further highlight the similarity with data augmentation, it is interesting to note that newly constructed artificial model parameters $\hat{\Theta}^{(\setminus k)}(\gamma\Lambda)$ (for each $\gamma\in\Gamma$) are used in the computation of Λ in (20).

The practical implementation of the regularization strategy discussed above requires minor changes to (18), (19a) and (19b), which become, respectively:

$$\hat{\Theta}^{(\setminus k)}(\gamma\Lambda) = \left(X_{\setminus k}'X_{\setminus k} + N_T\gamma^2\Lambda\Lambda\right)^{-1}X_{\setminus k}'Y_{\setminus k}, \tag{22}$$

$$\nabla_\Lambda E = -\gamma^2\frac{N_T}{KN_V}\frac{1}{|\Gamma|}\sum_{k=1}^{K}\sum_{\gamma\in\Gamma}\mathrm{vec}\left(\Lambda\tilde{B}_k + \tilde{B}_k\Lambda\right), \tag{23}$$

$$\nabla_\lambda E = -\gamma^2\frac{N_T}{KN_V}\frac{1}{|\Gamma|}\sum_{k=1}^{K}\sum_{\gamma\in\Gamma}\mathrm{diag}\left(\Lambda\tilde{B}_k + \tilde{B}_k\Lambda\right). \tag{24}$$

with $\tilde{B}_k = \tilde{A}_k'X_k'R_k\left(\hat{\Theta}^{(\setminus k)}\right)'$, $R_k = X_k\hat{\Theta}^{(\setminus k)}(\Lambda) - Y_k$, and $\tilde{A}_k = \left(X_{\setminus k}'X_{\setminus k} + N_T\gamma^2\Lambda\Lambda\right)^{-1}$.

5.2. Regularization in validation

The second strategy for reducing the risk of overfitting stems from the same considerations discussed in the previous paragraph. Indeed, in the case of a sparse true parameter vector Θ_o, we would aim to guide the hyperparameter Λ to take the structure in (17), rather than simply minimizing the fitting herror $\left\|Y_k - X_k\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|_F^2$ of the kth validation fold. To this aim, the validation loss function in (15) is modified as follow, by introducing a regularization term:

$$\Lambda^\star = \arg\min_{\Lambda}\frac{1}{2N}\sum_{k=1}^{K}\left\|Y_k - X_k\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|_F^2 +$$
$$+\frac{1}{2}\mu\sum_{k=1}^{K}\left\|\Lambda\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|_F^2, \tag{25}$$

where $\mu\in\mathbb{R}$, with $\mu > 0$, acts as an additional hyperparameter. Thus, unlike the "optimal-solution augmentation" strategy discussed in Section 5.1, this approach requires the tuning of an additional (albeit scalar) regularization hyperparameter, hence the requirement for an additional hyper-validation dataset.

The practical implementation of the regularization strategy discussed in this subsection requires the derivation of the gradient of the regularization term:

$$Q = \frac{1}{2}\mu\sum_{k=1}^{K}\left\|\Lambda\hat{\Theta}^{(\setminus k)}(\Lambda)\right\|_F^2, \tag{26}$$

with respect to Λ (or equivalently λ), whose analytical expressions are given by:

$$\nabla_\Lambda Q = \mu\sum_{k=1}^{K}\mathrm{vec}\left(D_k\left(\hat{\Theta}^{(\setminus k)}\right)' - N_T\left(\Lambda G_k + G_k\Lambda\right)\right), \tag{27a}$$

$$\nabla_\lambda Q = \mu\sum_{k=1}^{K}\mathrm{diag}\left(D_k\left(\hat{\Theta}^{(\setminus k)}\right)' - N_T\left(\Lambda G_k + G_k\Lambda\right)\right), \tag{27b}$$

with $D_k = \Lambda\hat{\Theta}^{(\setminus k)}$ and $G_k = A_k'\Lambda' D_k\left(\hat{\Theta}^{(\setminus k)}\right)'$.

Derivations of the gradients $\nabla_\Lambda Q$ (resp. $\nabla_\lambda Q$) in (27a) (resp. (27b)) are based on similar differential matrix calculus arguments used to compute (19a) and (19b), and are reported in Appendix B.

6. Examples

In this section, we present numerical examples to demonstrate the effectiveness of the multi-hyperparameter optimization approach. In this context, we conduct a comparative analysis between this approach and established benchmark techniques, such as LASSO, Ridge and Elastic Net regressions, all of which are readily available through the Python *scikit-learn* package. Comparison in terms of final model performance and computational complexity is discussed in Sections 6.1 and 6.2, respectively. Parametric identification of *Linear Parameter-Varying* (LPV) dynamical systems with an unknown model structure is addressed in Section 6.3.

Model performance is measured on a test set (used neither for training nor for validation) through the R^2 index, defined as:

$$\mathrm{R}^2 = \max\left\{0; 1 - \frac{\sum_{i=1}^{N_{\text{test}}}\left\|y^{(i)} - \hat{y}^{(i)}\right\|^2}{\sum_{i=1}^{N_{\text{test}}}\left\|y^{(i)} - \bar{y}\right\|^2}\right\}, \tag{28}$$

where $\hat{y}^{(i)}$ is the estimated model output for the ith sample, $\bar{y}$ is the sample mean of the outputs over the test dataset, and N_{test} is length of the test dataset. In assessing algorithms' performance, we use test datasets larger (up 100x) than the training set. Although this is not realistic in practice, it allows us to have a fair and statistically significant evaluation and comparison of different algorithms.

Output data (including test data) is corrupted by an additive noise e. The effect of the noise is measured in terms of the Signal-to-Noise Ratio (SNR), defined as:

$$\mathrm{SNR} = 10\log\left(\frac{\|y-e\|^2}{\|e\|^2}\right). \tag{29}$$

For the multi-ridge case discussed in the paper, numerical optimization is performed through gradient descent, with learning rate chosen manually. In the first and third examples and in all algorithms discussed in this section, 5-fold cross-validation is implemented to select the regularization hyperparameters. Once the hyperparameters are chosen, models are re-estimated on the complete training dataset. In the second example, a simple holdout validation is used, as the focus is only on the numerical computation of the gradient $\nabla_\lambda E$ and not on assessing model performance.

All computations are performed on a server with two 64-core AMD EPYC 7742 Processors, 256 GB of RAM, and 4 Nvidia RTX 3090 GPUs. The hardware resources of the server have been limited to 10 CPU threads and 1 GPU.

The software has been developed in PyTorch with a scikit-learn compatible interface. All the codebase, along with documentation, is available in the GitHub repository https://github.com/gabribg88/Multiridge.

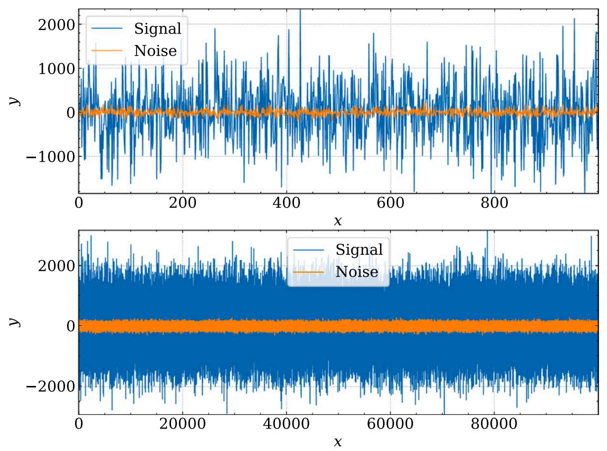


Fig. 1. Example of dataset realization. The training dataset (top) varies with each experiment, while the test dataset (bottom) remains constant across all experiments. Output signal (blue) and noise (orange).

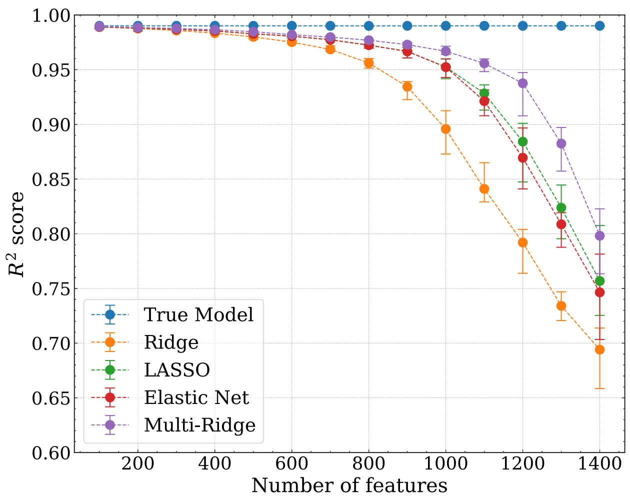


Fig. 2. Example 1: model performance *vs* number of input features for different regularization algorithms. Median over 10 repetitions (dots), along with 5-th and 95-th percentile (bars) are depicted.

6.1. Example 1: Sensitivity to the number of features

In this numerical example, we investigate the sensitivity of different regularization strategies with respect to the number of features, while keeping the number of data points and the information ratio (i.e., the ratio between the total number of input features available in the dataset and the number of informative features) constant.

Since the number of features effectively determines the complexity of the model, we expect that as the ratio of input features to data points grows, the model's capacity to capture not only the underlying pattern but also the noise within the training data, increases, potentially resulting in overfitting. As the overfitting increases, the impact of the regularization term becomes more significant. This provides a basis for comparing the effectiveness of various regularization strategies.

14 distinct conditions are simulated, with the number of features D incrementally ranging from 100 to 1400 with a step of 100. To enhance the statistical significance of the results, we replicated each condition 10 times, utilizing different random seeds for each replication. In each individual experiment, a total of $N_{train} = 10^3$ training data samples and $N_{test} = 10^5$ test samples were generated, according to the following linear data generation procedure:

$$y = x'\tilde{\Theta}_o + e, \tag{30}$$

where each component of the feature vector $x \in \mathbb{R}^D$ is drawn from a Normal distribution $\mathcal{N}(0,1)$. The noise e is also Normally distributed, with $e \sim \mathcal{N}(0,\sigma_e^2)$ and variance σ_e^2 chosen to ensure an SNR of approximately 20 dB. The actual coefficient vector $\tilde{\Theta}_o$ is a sparse version of a vector Θ_o, whose components are drawn from a uniform distribution $\mathcal{U}_{[-50,50]}$ between -50 and 50. The relation between $\tilde{\Theta}_o$ and Θ_o is thus expressed as:

$$\tilde{\Theta}_{o,j} = \begin{cases} \Theta_{o,j} & \text{if } j \in \mathcal{I} \\ 0 & \text{if } j \notin \mathcal{I} \end{cases}, \tag{31}$$

where $\mathcal{I} \subseteq \{1,2,\dots,D\}$ represents the index set of informative features. The cardinality of $\mathcal{I}$ is $|\mathcal{I}| = \lfloor D I_{frac} \rfloor$, with $\lfloor\cdot\rfloor$ being the floor operator. The fraction of informative features I_{frac} is set to 0.5 and kept constant throughout this example. Each element $j \in \mathcal{I}$ is uniformly drawn at random, without replacement, in $\{1,2,\dots,D\}$. An example of a dataset realization can be seen in Fig. 1.

In each experiment, the benchmark learning algorithms underwent a systematic training process with the following sequential stages: (i) standardization of both features and target variable; (ii) hyperparameter optimization through 5-fold cross-validation;, (iii) re-training of the models on the complete training dataset; (iv) evaluation of algorithms' performances on a test set. Hyperparameter optimization varies depending on the regularization algorithm, as detailed below:

- Ridge regression: a grid-search was conducted on a logarithmically spaced grid of 10^3 values for the l_2-regularization strength. The grid values ranged from 10^{-3} to 10^6.
- LASSO: a grid-search was performed using a logarithmically spaced grid of 10^3 values for the l_1-regularization strength. The grid values spanned from 10^{-5} to 10^2.
- Elastic Net regression: a random-search was executed with 10^3 repetitions on pairs of: logarithmically spaced values of the regularization strength ranging from 10^{-5} to 10^3, and ratio between ℓ_1 and ℓ_2 regularization spanning linearly from 0 to 1.

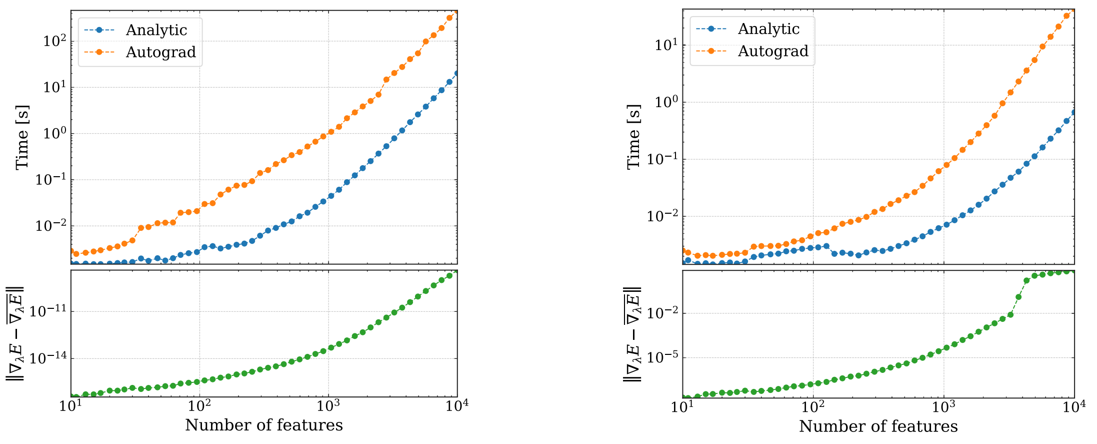
For multi-ridge regression, the hyperparameters Λ are optimized through a gradient-based approach, and initialized as an identity matrix I_D. The learning rate is set to 350 across all scenarios, with a decay rate of 0.999 applied at each epoch over the course of 300 training epochs.

Results are presented in Fig. 2, illustrating the sensitivity of model performance with respect to the number of input features for Ridge, LASSO, Elastic Net, and the proposed multi-ridge regression. For less than 300 features, all methods deliver performance comparable to the oracle model with the true parameter vector $\tilde{\Theta}_o$. However, as the number of input features grows, each method exhibits a gradual decline in performance due to overfitting.

Multi-ridge regression consistently outperforms the other benchmark regularization strategies. Notably, simple Ridge regression records the lowest performance. This outcome is expected for the given problem with a sparse true parameter vector. Thus, algorithms that promote sparse solutions, such as LASSO, are more suitable. Nevertheless, even though multi-ridge regression employs ℓ_2 regularization instead of ℓ_1, it still outperforms LASSO. This enhanced performance stems from our method's flexibility in assigning a distinct regularization strength to each feature, based on its inherent informativeness.

6.2. Example 2: Computational performances

In this numerical example we show the advantages in terms of both computational efficiency (measured in terms of GPU time) and numerical accuracy (assessed by quantifying the norm of the difference between the analytic calculation and the automatic differentiation, as implemented within the PyTorch framework) during the computation of the gradient in (19b). These benefits become increasingly pronounced as the number of features in the simulated dataset grows. More specifically, we constructed 50 distinct scenarios, each characterized by a progressively growing number of features, with logarithmically spaced values. For every scenario, we conducted 10 experiments,



(a) Double-precision floating-point format (float64) (b) Single-precision floating-point format (float32)

Fig. 3. Example 2: Results of computational performance. GPU time required to compute the gradient $\nabla_\lambda E$ (upper panels); norm of the difference between analytic computation of the gradient $\nabla_\lambda E$ and its numerical computation through automatic differentiation in PyTorch, denoted as $\overline{\nabla_\lambda E}$ (lower panels).

each repeated with varying random seeds to enhance the statistical significance of the results. In all the experiments, a fixed number of $N_{train} = 10^3$ training data samples were generated. Each data sample was composed of an input–output pair (x, y), where $x \in \mathbb{R}^D$ and $y \in \mathbb{R}^M$, with $M = 10$. Generated samples were subsequently partitioned into training fold data and validation fold data, according to an 80:20 ratio. This is equivalent to simple holdout validation instead of K fold.

The input features x, the targets y, as well as the hyperparameters λ, are generated according to a Normal distribution, with zero mean and unitary variance.

The experiment is conducted in 2 different scenarios. Firstly, with variables represented in double-precision floating-point format (float64), and secondly, with variables represented in single-precision floating-point format (float32). The number of features ranges from 10 to 10^4.

Results are illustrated in Fig. 3. The upper panel of the figure displays the GPU time required for the computation of the gradient $\overline{\nabla_\lambda E}$. The orange line corresponds to the time taken by the PyTorch autograd, whereas the blue line indicates the time consumed by the analytic calculation. As the number of features increases, the analytic calculation showcases a significant enhancement in computational efficiency, obtaining computational speed more than an order of magnitude faster w.r.t. automatic differentiation.

The lower panel of the figure displays the norm of the difference between the analytic calculation and the computation through automatic differentiation. In the scenario where variables are represented in double-precision floating-point format, this difference is negligible. Nonetheless, when variables are represented in single-precision floating-point format, this difference becomes noteworthy as the number of features increases.

6.3. LPV identification

As a last example, we consider the identification of an LPV system described by the AutoRegressive eXogenous (ARX) form:

$$\begin{aligned} y(k) = {} & 0.5\cos(p(k))y(k-2) - 0.1\sin^2(p(k))y(k-3) \\ & + (\cos(p(k)) - \sin(p(k)))u(k-2) \\ & + 3\sin(p(k))u(k-3) + e(k), \end{aligned} \tag{32}$$

where $u(k)$, $y(k)$, and $p(k)$ represent the measurement of the input, output, and scheduling variable at time step k, respectively, while $e(k)$ is a random Gaussian white noise.

A small-size dataset $\mathcal{D}$ of size $N = 50$ is gathered by exciting the system (32) with a white noise input sequence $\{u(k)\}_{k=1}^N$, with $u(k)$ drawn from a Gaussian distribution $\mathcal{N}(0,1)$. The scheduling variable $p(k)$ is also drawn from a Gaussian distribution, with $p(k) \sim \mathcal{N}(0,\pi)$. The amplitude of the noise $e(k)$ is chosen such that the SNR is about 14 dB (namely, the ratio between the power of the noise and the power of the noise-free output is approximately 4%).

In estimating the LPV system (32), we assume to be in a setting where the system structure is not known, and we consider an over-parameterized LPV-ARX model of the form:

$$\hat{y}(k) = \sum_{j=1}^{n_a} a_j(p(k))y(k-j) + \sum_{j=1}^{n_b} b_j(p(k))u(k-j), \tag{33}$$

with $n_a = n_b = 30$. Furthermore, the dependence of the coefficient functions a_j and b_j ($j = 1,\dots,30$) on the scheduling parameter p is parameterized in terms of a linear combination of a finite number of *a-priori* chosen basis functions, as follows:

$$a_j(p(k)) = \sum_{s=1}^{n_\alpha} a_{j,s}\psi_s(p(k)), \qquad i = 1,\dots,n_a, \tag{34a}$$

$$b_j(p(k)) = \sum_{s=1}^{n_\alpha} b_{j,s}\psi_s(p(k)), \qquad j = 1,\dots,n_b, \tag{34b}$$

where $n_\alpha = 8$, $\{a_{j,s} \in \mathbb{R}\}_{s=1}^{n_\alpha}$, and $\{b_{j,s} \in \mathbb{R}\}_{s=1}^{n_\alpha}$ are the unknown constant parameters to be estimated, and $\psi_s(p(k))$ is the sth element of the vector function:

$$\psi_s(p) = \left[1 \;\; p \;\; p^2 \;\; p^3 \;\; \sin(p) \;\; \cos(p) \;\; \sin^2(p) \;\; \cos^2(p)\right]. \tag{35}$$

Summarizing, we have to estimate $n_\alpha(n_a+n_b) = 480$ parameters from a dataset of length $N = 50$. In such a setting, regularization is expected to play a fundamental role for the estimation process.

For multi-ridge regression, the initial conditions of the hyperparameters λ are extracted from the solution of a LASSO problem. More specifically, if the model parameter $\boldsymbol{\Theta}_i$ estimated by the LASSO is zero, then the related hyperparameter λ_j is set to 10, otherwise $\lambda_j = 1$. This means that, in the initialization of λ, a 10x higher penalization is given to the model parameters estimated as zero by the LASSO. The optimal-solution augmentation strategy discussed in Section 5.1 is implemented to reduce the risk of overfitting to the validation datasets, by minimizing the evaluation criterion in (20) for $\gamma \in \Gamma = \{0.5,\ 1,\ 2\}$.

The performance of all the tested algorithms is evaluated on a test data of length $N_{\text{test}} = 3000$. Fig. 4 shows the boxplots of the different approaches over 200 Monte Carlo simulations, with: input, noise and scheduling variable independently regenerated at each run. The figure shows that LASSO, Elastic Net and the multi-hyperparameter approach proposed in this paper provide satisfactory performance, achieving a median R^2 value of 0.88, 0.86 and 0.91, respectively. However,

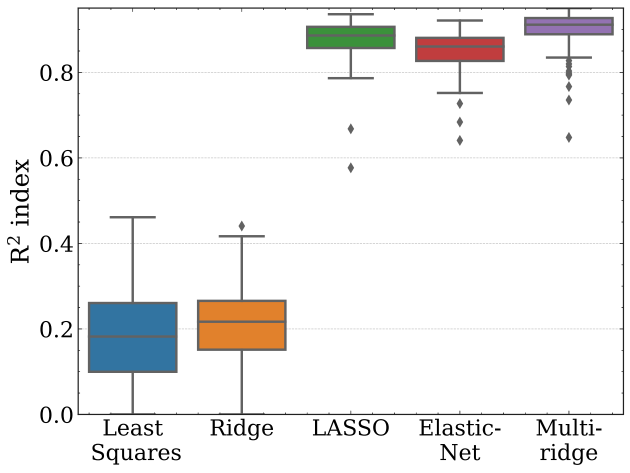


Fig. 4. LPV identification: boxplots of the R^2 index over 200 Monte Carlo runs. Medians: Least Squares (0.18); Ridge (0.21); LASSO (0.88); Elastic Net (0.86); multi-ridge (0.91).

this example also highlights how simple Ridge regression does not have enough flexibility in its regularization term. In contrast, multi-ridge regression, having a much larger flexibility, surpasses LASSO's performances too, even when dealing with a sparse structure in the "true" parameter vector.

7. Conclusions

Multi-penalty Ridge regression increases the degrees of freedom of the regression algorithm, thus allowing to enhance model's performance compared to widely adopted regularization techniques such as LASSO, Ridge, and Elastic Net regression. The optimization of multiple regularization hyperparameters can be achieved through gradient descent. Here, the analytical expression of the gradient for a cross-validation criterion with respect to the regularization hyperparameters can be derived using matrix differential calculus, avoiding the need for automatic differentiation and backpropagation. As shown in the numerical examples, this offers numerical benefits, especially when dealing with a large number of features. In particular, simulation results show that for small (100) to medium (800) feature sets, the improvement percentages are modest, but they noticeably increase as the number of features grows. For instance, the improvement over Ridge is 0.046% at 100 features and approximately 2% by 800 features. For large feature sets (900 to 1400 features), the improvements become quite substantial. Overall, Multi-Ridge outperforms Ridge, Lasso, and Elastic Net by significant margins, achieving over a 20% improvement over Ridge, and 7% over Lasso at 1300 features. This notable performance improvement highlights the effectiveness of Multi-Ridge in high-dimensional spaces, where traditional methods may struggle with the increased risk of overfitting.

The approach is specifically tailored to ℓ_2 regularization and takes advantage of the fact that the optimal model parameters can be determined analytically. However, ongoing research aims to extend the matrix differential calculus-based approach presented in this paper to other types of problems where optimal model parameters cannot be expressed analytically and must be determined through iterative numerical optimization. These types of problems include LASSO with multiple hyperparameters or nested optimization problems arising in meta-learning, where models are trained on the task of learning themselves, often with the aim of rapidly adapting to new tasks using only a small-size dataset.

CRediT authorship contribution statement

Gabriele Maroni: Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Loris Cannelli:** Writing – review & editing, Validation, Software, Methodology, Data curation, Conceptualization. **Dario Piga:** Writing – review & editing, Supervision, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Derivation of (19a) and (19b)

By the *first identification theorem* (Magnus & Neudecker, 2019, Chapter 9, Section 5), the relationship between the differential and the gradient of the evaluation criterion E is given by:

$$dE = \left(\nabla_{\Lambda} E\right)' d\,\mathrm{vec}\Lambda, \tag{36a}$$

or equivalently:

$$dE = \left(\nabla_{\lambda} E\right)' d\lambda. \tag{36b}$$

Thus, from (36), both $\nabla_{\Lambda} E$ and $\nabla_{\lambda} E$ can be readily obtained, as discussed in the following. By differentiating both sides of (4) and leveraging linearity of the differential operator, we obtain:

$$dE = d\left(\frac{1}{K}\sum_{k=1}^{K} L_k\right) = \frac{1}{K}\sum_{k=1}^{K} dL_k. \tag{37}$$

To derive dL_k (and thus dE), L_k is rewritten in terms of the trace operator as follows:

$$L_k = \frac{1}{2N_V}\left\|R_k\right\|_F^2 = \frac{1}{2N_V}\mathrm{tr}\left(R_k' R_k\right), \tag{38}$$

where $R_k = X_k \hat{\Theta}^{(\setminus k)}(\Lambda) - Y_k$. By differentiating (38):

$$\begin{aligned} dL_k &= \frac{1}{2N_V} d\,\mathrm{tr}\left(R_k' R_k\right) \\ &= \frac{1}{2N_V}\mathrm{tr}\left[\left(dR_k'\right) R_k + R_k' dR_k\right] \\ &= \frac{1}{2N_V}\mathrm{tr}\left[\left(dR_k\right)' R_k + R_k' dR_k\right] \\ &= \frac{1}{2N_V}\mathrm{tr}\left(R_k' dR_k + R_k' dR_k\right) \\ &= \frac{1}{N_V}\mathrm{tr}\left(R_k' dR_k\right), \end{aligned} \tag{39}$$

where the following properties of the differential operator are used, for any matrix $U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{m\times p}$:

$$d(U') = dU', \tag{40a}$$

$$d(UV) = (dU)V + U dV \quad (\mathit{Product\ rule}). \tag{40b}$$

The computation of dL_k in (39) requires to compute the differential dR_k. From the definition of R_k and the analytical expression of the model parameters $\hat{\Theta}$ in (18):

$$\begin{aligned} dR_k &= d\left(X_k \hat{\Theta}^{(\setminus k)}\right) \\ &= d\left(X_k\left(X_{\setminus k}' X_{\setminus k} + N_T \Lambda\Lambda\right)^{-1} X_{\setminus k}' Y_{\setminus k}\right) \\ &= d\left(X_k A_k X_{\setminus k}' Y_{\setminus k}\right) = R_k' X_k \left(dA_k\right) X_{\setminus k}' Y_{\setminus k}, \end{aligned} \tag{41}$$

with $A_k = \left(X_{\setminus k}' X_{\setminus k} + N_T \Lambda\Lambda\right)^{-1}$. By substituting (41) into (39), and from the cyclic property of the trace operator, we obtain:

$$dL_k = \frac{1}{N_V}\mathrm{tr}\left[R_k' X_k \left(dA_k\right) X_{\setminus k}' Y_{\setminus k}\right]$$

$$= \frac{1}{N_V}\operatorname{tr}\left(X'_{\setminus k}Y_{\setminus k}R'_kX_k\,dA_k\right). \tag{42}$$

The differential dA_k appearing in (41) can be easily computed starting from the definition of the definition of A_k. Specifically, we have:

$$\begin{aligned} dA_k &= d\left(X'_{\setminus k}X_{\setminus k} + N_T\Lambda\Lambda\right)^{-1} \\ &= -N_T A_k d\left(\Lambda\Lambda\right)A_k, \end{aligned} \tag{43}$$

where the following identity is used:

$$d\left(W^{-1}\right) = -W^{-1}\left(dW\right)W^{-1}, \tag{44}$$

for any invertible square matrix $W \in \mathbb{R}^{m\times m}$. By substituting (43) into (42), we obtain:

$$\begin{aligned} dL_k &= -\frac{N_T}{N_V}\operatorname{tr}\left[X'_{\setminus k}Y_{\setminus k}R'_kX_kA_kd\left(\Lambda\Lambda\right)A_k\right] \\ &= -\frac{N_T}{N_V}\operatorname{tr}\left[A_kX'_{\setminus k}Y_{\setminus k}R'_kX_kA_kd\left(\Lambda\Lambda\right)\right] \\ &= -\frac{N_T}{N_V}\operatorname{tr}\left[\hat{\Theta}^{(\setminus k)}R'_kX_kA_kd\left(\Lambda\Lambda\right)\right]. \end{aligned} \tag{45}$$

Finally, from the product rule and by defining $B_k = A'_kX'_kR_k\left(\hat{\Theta}^{(\setminus k)}\right)'$, (45) can be written as:

$$\begin{aligned} dL_k &= -\frac{N_T}{N_V}\operatorname{tr}\Big[\hat{\Theta}^{(\setminus k)}R'_kX_kA_k\left(d\Lambda\right)\Lambda \\ &\qquad + \hat{\Theta}^{(\setminus k)}R'_kX_kA_k\Lambda d\Lambda\Big] \\ &= -\frac{N_T}{N_V}\operatorname{tr}\left[B'_k\left(d\Lambda\right)\Lambda + B'_k\Lambda d\Lambda\right] \\ &= -\frac{N_T}{N_V}\operatorname{tr}\left[\Lambda B'_k\left(d\Lambda\right) + B'_k\Lambda d\Lambda\right] \\ &= -\frac{N_T}{N_V}\operatorname{tr}\left[\left(\Lambda B'_k + B'_k\Lambda\right)d\Lambda\right]. \end{aligned} \tag{46}$$

Using the following property: for any matrix $U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{m\times p}$:

$$\operatorname{tr}\left(U'V\right) = \operatorname{vec}\left(U\right)'\operatorname{vec}\left(V\right), \tag{47}$$

(46) becomes:

$$dL_k = -\frac{N_T}{N_V}\operatorname{vec}\left(\Lambda B_k + B_k\Lambda\right)'\,d\operatorname{vec}\Lambda. \tag{48}$$

By combining (37) and (48), the differential dE can be written as:

$$dE = -\frac{N_T}{KN_V}\sum_{k=1}^{K}\operatorname{vec}\left(\Lambda B_k + B_k\Lambda\right)'\,d\operatorname{vec}\Lambda. \tag{49}$$

Thus, from (49) and (36a), we observe:

$$\nabla_\Lambda E = -\frac{N_T}{KN_V}\sum_{k=1}^{K}\operatorname{vec}\left(\Lambda B_k + B_k\Lambda\right), \tag{50}$$

which proves (19a). In order to prove (19b), we rewrite (46) as:

$$dL_k = -\frac{N_T}{N_V}\operatorname{diag}\left(\Lambda B_k + B_k\Lambda\right)'\,d\lambda, \tag{51}$$

where we used the following properties:

$$\operatorname{tr}\left(W\operatorname{Diag}\left(v\right)\right) = \operatorname{diag}\left(W\right)'v, \tag{52a}$$

$$d\operatorname{Diag}(v) = \operatorname{Diag}(dv), \tag{52b}$$

for any square matrix $W \in \mathbb{R}^{m\times m}$ and vector $v \in \mathbb{R}^m$. By combining (37) and (51), the differential dE can be written as:

$$dE = -\frac{N_T}{KN_V}\sum_{k=1}^{K}\operatorname{diag}\left(\Lambda B_k + B_k\Lambda\right)'\,d\lambda. \tag{53}$$

Thus, from (53) and (36b), we observe:

$$\nabla_\lambda E = -\frac{N_T}{KN_V}\sum_{k=1}^{K}\operatorname{diag}\left(\Lambda B_k + B_k\Lambda\right), \tag{54}$$

which proves (19b).

Appendix B. Derivation of (27a) and (27b)

First, let us rewrite the regularization term Q in (26) as:

$$Q = \sum_{k=1}^{K} Q_k, \tag{55}$$

with:

$$\begin{aligned} Q_k &= \frac{1}{2}\mu\left\|\Lambda\hat{\Theta}^{(\setminus k)}\right\|_F^2 = \frac{1}{2}\mu\operatorname{tr}\left(\left(\hat{\Theta}^{(\setminus k)}\right)'\Lambda\Lambda\hat{\Theta}^{(\setminus k)}\right) \\ &= \frac{1}{2}\mu\operatorname{tr}\left(D'_kD_k\right), \end{aligned} \tag{56}$$

where $D_k = \Lambda\hat{\Theta}^{(\setminus k)}$ and the dependence of $\Theta^{(\setminus k)}$ on Λ is omitted to ease reading. Differentiating both sides of (56) and using the trace and differential operator properties already adopted in Appendix A we obtain:

$$\begin{aligned} dQ_k &= \frac{1}{2}\mu\, d\left[\operatorname{tr}\left(D'_kD_k\right)\right] = \frac{1}{2}\mu\operatorname{tr}\left[d\left(D'_kD_k\right)\right] \\ &= \frac{1}{2}\mu\operatorname{tr}\left[\left(dD'_k\right)D_k + D'_kdD_k\right] \\ &= \frac{1}{2}\mu\operatorname{tr}\left[\left(dD_k\right)'D_k + D'_kdD_k\right] \\ &= \frac{1}{2}\mu\operatorname{tr}\left(D'_kdD_k + D'_kdD_k\right) \\ &= \mu\operatorname{tr}\left(D'_kdD_k\right). \end{aligned} \tag{57}$$

From the definition of D_k, (57) can be rewritten as:

$$\begin{aligned} dQ_k &= \mu\operatorname{tr}\left[D'_kd\left(\Lambda\hat{\Theta}^{(\setminus k)}\right)\right] \\ &= \mu\operatorname{tr}\left[D'_k\left(d\Lambda\right)\hat{\Theta}^{(\setminus k)} + D'_k\Lambda d\hat{\Theta}^{(\setminus k)}\right] \\ &= \mu\left[\operatorname{tr}\left(\hat{\Theta}^{(\setminus k)}D'_kd\Lambda\right) + \operatorname{tr}\left(D'_k\Lambda d\hat{\Theta}^{(\setminus k)}\right)\right]. \end{aligned} \tag{58}$$

Let us now focus on the second term in the right side of (58), which is rewritten by using the same properties adopted in Appendix A, as:

$$\begin{aligned} &\operatorname{tr}\left(D'_k\Lambda d\hat{\Theta}^{(\setminus k)}\right) = \\ &= \operatorname{tr}\left[D'_k\Lambda d\left(\left(X'_{\setminus k}X_{\setminus k} + N_T\Lambda\Lambda\right)^{-1}X'_{\setminus k}Y_{\setminus k}\right)\right] \\ &= \operatorname{tr}\left[D'_k\Lambda\left(dA_k\right)X'_{\setminus k}Y_{\setminus k}\right] \\ &= \operatorname{tr}\left(X'_{\setminus k}Y_{\setminus k}D'_k\Lambda dA_k\right) \\ &= -N_T\operatorname{tr}\left[X'_{\setminus k}Y_{\setminus k}D'_k\Lambda A_kd\left(\Lambda\Lambda\right)A_k\right] \\ &= -N_T\operatorname{tr}\left[\hat{\Theta}^{(\setminus k)}D'_k\Lambda A_kd\left(\Lambda\Lambda\right)\right] \\ &= -N_T\operatorname{tr}\left[\hat{\Theta}^{(\setminus k)}D'_k\Lambda A_k\left(d\Lambda\right)\Lambda + \hat{\Theta}^{(\setminus k)}D'_k\Lambda A_k\Lambda d\Lambda\right]. \end{aligned} \tag{59}$$

By defining $G_k = A'_k\Lambda'D_k\left(\hat{\Theta}^{(\setminus k)}\right)'$, the above equation can be written in the compact form:

$$\begin{aligned} \operatorname{tr}\left(D'_k\Lambda d\hat{\Theta}^{(\setminus k)}\right) &= -N_T\operatorname{tr}\left[G'_k\left(d\Lambda\right)\Lambda + G'_k\Lambda d\Lambda\right] \\ &= -N_T\operatorname{tr}\left[\left(\Lambda G'_k + G'_k\Lambda\right)d\Lambda\right]. \end{aligned} \tag{60}$$

Then, by substituting (60) into (58), we obtain:

$$\begin{aligned} dQ_k &= \mu\left[\operatorname{tr}\left(\hat{\Theta}^{(\setminus k)}D'_kd\Lambda\right) - N_T\operatorname{tr}\left(\left(\Lambda G'_k + G'_k\Lambda\right)d\Lambda\right)\right] \\ &= \mu\left[\operatorname{tr}\left(\left(\hat{\Theta}^{(\setminus k)}D'_k - N_T\left(\Lambda G'_k + G'_k\Lambda\right)\right)d\Lambda\right)\right]. \end{aligned} \tag{61}$$

Thanks to (47), (61) can be written as:

$$dQ_k = \mu\operatorname{vec}\left(D_k\hat{\Theta}^{(\setminus k)} - N_T\left(\Lambda G_k + G_k\Lambda\right)\right)'\,d\operatorname{vec}\Lambda. \tag{62}$$

Based on the same considerations discussed in Appendix A, from (62), and the definition of Q in (26), we obtain:

$$\nabla_\Lambda Q = \mu\sum_{k=1}^{K}\operatorname{vec}\left(D_k\left(\hat{\Theta}^{(\setminus k)}\right)' - N_T\left(\Lambda G_k + G_k\Lambda\right)\right), \tag{63}$$

thus proving (27a). Using the matrix properties in (52), (61) can be also written as:

$$dQ_k = \mu\operatorname{diag}\left(D\left(\hat{\Theta}^{(\setminus k)}\right)' - N_T\left(\Lambda G_k + G_k\Lambda\right)\right)'\,d\lambda. \tag{64}$$

Therefore:

$$\nabla_{\lambda} Q = \mu \sum_{k=1}^{K} \text{diag}\left(D_k \left(\hat{\Theta}^{(\backslash k)}\right)' - N_T \left(\Lambda G_k + G_k \Lambda\right)\right), \tag{65}$$

thus proving (27b).

References

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

Bennett, K. P., Hu, J., Ji, X., Kunapuli, G., & Pang, J.-S. (2006). Model selection via bilevel optimization. In *The 2006 IEEE international joint conference on neural network proceedings* (pp. 1922–1929). IEEE.

Bertrand, Q., Klopfenstein, Q., Massias, M., Blondel, M., Vaiter, S., Gramfort, A., et al. (2022). Implicit differentiation for fast hyperparameter selection in non-smooth convex learning. *Journal of Machine Learning Research*, *23*(1), 6680–6722.

Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., et al. (2022). Efficient and modular implicit differentiation. *Advances in Neural Information Processing Systems*, *35*, 5230–5242.

Do, C. B., Foo, C.-S., & Ng, A. (2007). Efficient multiple hyperparameter learning for log-linear models. *Advances in Neural Information Processing Systems*, *20*.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(7).

Franceschi, L., Donini, M., Frasconi, P., & Pontil, M. (2017). Forward and reverse gradient-based hyperparameter optimization. In *International conference on machine learning* (pp. 1165–1173). PMLR.

Frecon, J., Salzo, S., & Pontil, M. (2018). Bilevel learning of the group lasso structure. *Advances in Neural Information Processing Systems*, *31*.

Hataya, R., Zdenek, J., Yoshizoe, K., & Nakayama, H. (2022). Meta approach to data augmentation optimization. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision* (pp. 2574–2583).

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

Kingma, D. P. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kunisch, K., & Pock, T. (2013). A bilevel optimization approach for parameter learning in variational models. *SIAM Journal on Imaging Sciences*, *6*(2), 938–983.

Latendresse, S., & Bengio, Y. (2000). Linear regression and the optimization of hyper-parameters. *Web Published Writings*, 1–7.

Laurain, V., Tóth, R., Piga, D., & Darwish, M. A. H. (2020). Sparse RKHS estimation via globally convex optimization and its application in LPV-IO identification. *Automatica*, *115*, Article 108914.

Lorraine, J., Vicol, P., & Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *International conference on artificial intelligence and statistics* (pp. 1540–1552). PMLR.

Magnus, J. R., & Neudecker, H. (2019). *Matrix differential calculus with applications in statistics and econometrics*. John Wiley & Sons.

Maroni, G., Cannelli, L., & Piga, D. (2024). Differentiable multi-ridge regression for system identification. In *20th IFAC symposium on system identification*. Boston, MA, USA: URL https://github.com/gabribg88/Multiridge-SYSID/blob/master/SYSID_multiridge.pdf.

Murphy, K. P. (2022). *Probabilistic machine learning: an introduction*. MIT Press.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. In *International conference on machine learning* (pp. 737–746). PMLR.

Piga, D., & Tóth, R. (2013). LPV model order selection in an LS-SVM setting. In *52nd IEEE conference on decision and control* (pp. 4128–4133).

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, *25*.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *58*(1), 267–288.

Tóth, R., Hjalmarsson, H., & Rojas, C. R. (2012). Order and structural dependence selection of LPV-ARX models revisited. In *51st IEEE conference on decision and control* (pp. 6271–6276).

van de Wiel, M. A., van Nee, M. M., & Rauschenberger, A. (2021). Fast cross-validation for multi-penalty high-dimensional ridge regression. *Journal of Computational and Graphical Statistics*, *30*(4), 835–847.