IMT School for Advanced Studies, Lucca

Lucca, Italy

Model learning from data: from centralized multi-model regression to distributed cloud-aided single-model estimation

PhD Program in Institutions, Markets and Technologies
Curriculum in Control Systems

XXX Cycle

By

Valentina Breschi

2018

The dissertation of Valentina Breschi has been approved.

Program Coordinator: Prof. Alberto Bemporad,
IMT School for Advanced Studies Lucca

Advisor: Prof. Alberto Bemporad,
IMT School for Advanced Studies Lucca (IT)

Co-Advisor: Dr. Dario Piga,
Dalle Molle Institute for Artificial Intelligence (CH)

The dissertation of Valentina Breschi has been reviewed by:

Prof. Paul M. J. Van den Hof ,
Technische Universiteit Eindhoven (NL)

Prof. Alessio Benavoli,
Dalle Molle Institute for Artificial Intelligence (CH)

IMT School for Advanced Studies, Lucca

2018

A Edda e a Fabio, che mi voleva "dottore".

Contents

List of Figures

List of Tables

Acknowledgements

First of all I would like to thank my advisors, Alberto Bemporad and Dario Piga. Alberto you offered me the opportunity to pursue my dream and I will always be thankful for your guidance, support and for all the opportunities you gave me in the last three years. Dario, I cannot thank you enough for all your invaluable help and your patience. I would also like to thank Ilya Kolmanovsky for hosting me at the University of Michigan and making that experience enriching with his kindness and expertise. This thesis would not have been possible without you all. I would like to express my gratitude to Prof. Paul Van den Hof and Prof. Alessio Benavoli for reviewing my thesis.

I am thankful to all the people who have become like a family to me: Elena, Lucia, Claudia, Maria Cristina, Irene, Pakhee, Anita, Soumali, Loredana, Daria, Paolo, Alessandro, Valerio, Valerio, Andreas, Manas, Vihang, Luca, Viggy, Puya, Vitaly, Vincenzo, Mika. I want to thank Giovanna for constantly trying to boost my confidence and for all the support she gave me during my grimmest moments. I am thankful to Eleonora, Giulia, Andrea, Gabriele, Elisa, Andrea, Alberto, Giuditta, Matteo, Matteo, Caterina, Alessio, Carolina and the friends of a lifetime: Fiamma, Lucrezia and Riccardo. Probably, I have never thanked you but you have been an invaluable support to me.

Finally, I would like to thank my family for encouraging and standing me. In particular, I would like to thank my parents, Chiara and Paolo, and my brother, Giulio. I owe you everything and this thesis is also dedicated to you.

Vorrei ringraziare la mia famiglia che mi ha sempre sostenuto e sopportato. Sarò sempre grata di avervi nella mia vita. In particolare, vorrei ringraziare i miei genitori, Chiara e Paolo, e mio fratello Giulio. Vi devo tutto e questa tesi é dedicata anche a voi.

Vita

Jan. 25, 1990 Born, Florence, Italy

2014-2018 **Ph.D.**, *IMT School for Advanced Studies*, Lucca
Dynamical Systems, Control and Optimization Research Unit.
Advisors: Prof. Alberto Bemporad and Dr. Dario Piga

2017 **Visiting scholar**, *University of Michigan - Aerospace Engineering Dept.*, Ann Arbor (MI).

2011-2014 **M. Eng.**, *University of Florence*, Florence (IT).
Electrical and Automation Engineering degree.
Final Mark: 110/110 cum Laude.
Thesis title: *Quadcopters advanced model design for autonomous navigation.*

2008-2011 **B. Eng.**, *University of Florence*, Florence (IT).
Electronic and Telecommunication Engineering degree.
Final Mark: 110/110 cum Laude.
Thesis title: *Adaptive control and the hypersonic aircraft X-15-3.*

Publications

1. V. Breschi, D. Piga, A. Bemporad, "Jump model learning and filtering for energy end-use disaggregation ", *SYSID 2018* (submitted)
2. A. Bemporad, V. Breschi, D. Piga, S. Boyd, "Fitting jump models", submitted, 2018
3. V. Breschi, I.V. Kolmanovsky, A. Bemporad, "Cloud-aided collaborative estimation by ADMM-RLS algorithms for connected vehicle prognostics", *American Control Conference 2018*
4. V. Breschi, D.Piga, A. Bemporad, "Piecewise affine regression via recursive multiple least squares and multicategory discrimination", *Automatica*, vol.73, pp. 155-162, Nov. 2016.
5. V. Breschi, D. Piga, A. Bemporad, "Learning hybrid models with logical and continuous dynamics via multiclass linear separation", in *Proc. 55th IEEE Conf. on Decision and Control*, Las Vegas, NV, 2016, pp. 353-358.
6. V. Breschi, A. Bemporad, D. Piga, "Identification of hybrid and linear parameter varying models via recursive piecewise affine regression and discrimination", in *Proc. European Control Conf.*, Aalborg, Denmark, 2016.

Abstract

This thesis presents a collection of methods for learning models from data, looking at this problem from two perspectives: learning multiple models from a single data source and how to switch among them, and learning a single model from data collected from multiple sources.
Regarding the first, to describe complex phenomena with simple but yet complete models, we propose a computationally efficient method for Piecewise Affine (PWA) regression. This approach relies on the combined use (i) multi-model Recursive Least-Squares (RLS) and (ii) piecewise linear multi- category discrimination, and shows good performances when used for the identification of Piecewise Affine dynamical systems with eXogenous inputs (PWARX) and Linear Parameter Varying (LPV) models. The technique for PWA regression is then extended to handle the problem of black-box identification of Discrete Hybrid Automata (DHA) from input/output observations, with hidden operating modes. The method for DHA identification is based on multi-model RLS and multi-category discrimination and it can approximate both the continuous affine dynamics and the Finite State Machine (FSM) governing the logical dynamics of the DHA. Two more approaches are presented to tackle the problem of learning models that jump over time. While the technique designed to learn Rarely Jump Models (RJMs) from data relies on the combined solution of a convex optimization problem and the use of Dynamic Programming, the method proposed for Markov Jump Models (MJMs) learning is based on the joint use of clustering plus multi-model RLS and a probabilistic clustering technique. The results of the tests performed on the method

for RJMs learning have motivated the design of two techniques for Non-Intrusive Load Monitoring, i.e., to estimate the power consumed by the appliances in an household from aggregated measurements, which are also presented in the thesis. In particular, methods based on (i) the optimization of a least-square error cost function, modified to account for the changes in the appliances operating regime, and relying on (ii) multi-model Kalman filters are proposed.
Regarding the second perspective, we propose methods for cloud-aided consensus-based parameter estimation over a multitude of similar devices (such as a mass production). In particular, we focus on the design of RLS-based estimators, which allow to handle (i) linear and (ii) nonlinear consensus constraints and (iii) multi-class estimation.

Notations

$\mathbb{R}^n$	Set of real vectors of dimension n
$\mathbb{N}$	Set of natural numbers, including zero
$\mathbb{Z}^+$	Set of positive integer numbers, excluding zero
$\mathbb{R}^+$	Set of positive real numbers, excluding zero
$\{0,1\}^n$	Set of n-dimensional vectors with Boolean-valued elements
$\|\mathcal{A}\|$	Cardinality of set $\mathcal{A}$
$\breve{\mathcal{A}}$	Complement of set $\mathcal{A}$
a_i	ith entry of vector a, with $a \in \mathbb{R}^n$
a_R	Sub-vector obtained collecting the entries a_r with $r \in R \subset \mathbb{Z}^+$
a_+	Vector with ith element equal to $\max\{a, 0\}$
$\|a\|_2$	Euclidean norm of a
$\max\{a, b\}$	Vector with ith component equal to $\max\{a_i, b_i\}$, $a, b \in \mathbb{R}^n$
A'	Transpose of matrix A, with $A \in \mathbb{R}^{n \times m}$
$tr(A)$	Trace of matrix A
A_i	ith row of matrix A
A_R	Sub-matrix of A collecting the rows A_r, with $r \in R \subset \mathbb{Z}^+$
$A_{R,C}$	Sub-matrix of A collecting $A_{r,c}$, $r \in R \subset \mathbb{Z}^+$ and $c \in C \subset \mathbb{Z}^+$
$A \otimes B$	Kroneker product of matrix A and B
I_n or $I_{n \times n}$	Identity matrix of dimension n
$0_{n \times n}$	Zero matrix of dimension n
$\propto$	Linear proportionality

Chapter 1

Introduction

Over the years, the interest towards model-based solutions for a wide variety of problems has been motivated by the reduction in the development time and engineering costs that can be attained using mathematical models for design. Among others, model-based methods are used for control design (*e.g.*, see [15, 49, 66]), state estimation (*e.g.*, see [19, 44, 63, 96]) and fault detection (*e.g.*, see [37, 60, 79]). One of the advantages of using model-based techniques is that the knowledge of a model describing the considered physical system allows one to perform extensive simulations and tests, thus reducing the experiments that have to be performed on the real system. Even though a model for the process of interest can be obtained from the physical laws describing the system, the direct derivation of mathematical models might be a difficult and time consuming task, especially when complex systems have to be modelled. Thanks to the increasing computational power of commonly used processing systems and the simplicity in handling large datasets, the interest towards data-based model learning has grown. On the one hand, data-based model estimation allows one to identify a model for a system having limited or no information about the underlying real process. On the other hand, collecting a large dataset accurately characterizing the model might require to carefully plan an identification campaign, to limit the costs related to model identification.

In this thesis, the problem of model learning is considered from two sides. First, the problem of identifying complex phenomena/systems through hybrid models is considered. In this case the process of interest is modeled through a collection of simpler local models, each one representing an operating regime of the actual system. The problem of estimating the power consumed by different appliances in an household from aggregate power measurements, *i.e.*, the disaggregation problem, will be considered as a possible use of hybrid models learned from data in a real-world application. Finally, assuming that data can be collected from a multitude of similar systems (*i.e.*, systems described by the same model) and that information can be shared in a cloud infrastructure, the problem of estimating the common parameters of the models is considered.

This chapter provides a review of the main contributions related to the problems of learning hybrid models and collaborative estimation that are presented in this thesis.

1.1 Learning hybrid models from data

Some physical phenomena and complex systems may not be accurately described using a single model, motivating investigations towards different modeling frameworks. The choice of the proper model structure might be a challenging problem. On the one hand, models that are too simple might not be accurate enough to reproduce the behavior of the underlying physical system. On the other hand, over-complicated models might overfit the data, leading to poor generalization on unseen data.

1.1.1 PWA regression

PWA models represent a good trade-off between model complexity and flexibility, as they approximate nonlinear, and possibly discontinuous, relationships using a set of affine sub-models defined over a polyhedral partition of the regressor space. The PWA regression problem thus

amounts to estimating both (i) the parameters of the affine sub-models and (ii) the partition of the regressor space from the data. A review on PWA regression techniques can be found in [51, 86].

Piecewise Affine (PWA) regression methods have been mainly studied in the context of dynamical system identification. In [97] the PWA regression problem is solved through mixed-integer programming. However, the number of integer variables increases with the dimension of the training set, making the approach suited only for PWA regression over small datasets. Instead of using mixed-integer programming, the approaches proposed in [14, 45, 62, 78] relies on a two-stage structure. The first step of all the methods is based on the simultaneous clustering of the regressor vectors and the estimation of the parameters of the affine sub-models, while the second stage is devoted to the computation of the polyhedral partition of the regressor space. The greedy approach presented in [14] to partition infeasible sets of inequalities can be computationally demanding when large training sets are used. Instead, the Expectation Maximization (EM) algorithm proposed in [78] for statistical clustering can be inefficient in estimating PWA maps with many parameters. The method in [62] relies on the description of the parameters of the sub-models through probability density functions, which are updated using particle filters. However, this method might require a high number of particles to accurately estimate the probability density functions. Differently from the approaches already introduced, the PWA regression technique presented in [45] can handle large training sets both in the clustering and in the parameter estimation phase. This method is based on clustering the regressors through a K-means like algorithm and the estimation of the sub-models parameters through weighted least squares. Nevertheless, due to the chosen clustering criterion, the approach proposed in [45] might perform poorly when the local models depend on redundant regressors, *i.e.*, when the local sub-models are over-parameterized. All the introduced approaches suffer from computational complexity issues. Furthermore, none of the methods presented in [14, 45, 62, 78, 97] is suited for on-line use, when the PWA map has to be updated in real-time when new data are acquired. The approach in [5] consists of a PWA

regression method based on the iterative clustering of the regressors and the identification of the sub-models parameters through Recursive Least-Squares (RLS). Although this approach is suited for on-line learning, the estimated polyhedral partition is given by the Voronoi diagram of the clusters' centroids. This structure is less flexible than general linear separation map and it might limit the regression capabilities of the method.
Chapter 2 describes a novel approach for the PWA approximation of vector-valued functions. The method is designed to overcome the main limitations of existing approaches, *i.e.*, their computational complexity and their applicability to batch estimation only. As the approaches presented in [14, 45, 62, 78], the proposed method is based on the execution of two stages: (i) simultaneous clustering of the regressor vectors and estimation of the model parameters; (ii) learning of the polyhedral partition from data. The first stage relies on the use of multi-model Recursive Least-Squares and a clustering policy, that allows one to consider both the modeling error and the spatial distribution of the data. Both parameter estimation and clustering can be performed iteratively, thus making the first stage suited for on-line estimation. To compute the polyhedral partition we propose two efficient multi-class linear separation methods. In particular, we use a batch Newton-like method and Averege Stochastic Gradient Descent (ASGD) for the online and offline computation of Piecewise Linear Separators (PWL), respectively. The use of general Piecewise Linear (PWL) separators, instead of the Voronoi diagram obtained from the clusters' centroids, increases the regression capabilities of the proposed method with respect to the one introduced in [5]. Via extensive simulation results we show that the PWA regression algorithm is computationally efficient, outperforming existing methods for PWA regression. Furthermore, the approaches proposed to compute the partition are proven to be more efficient than existing techniques to address the same problem.
The PWA regression approach is further extended to tackle the problems of data-driven estimation of Piecewise affine AutoRegressive models with eXogenous inputs (PWARX) and Linear Parameter Varying (LPV) systems. In the identification of LPV systems, the coefficients of the mo-

del are approximated with PWA functions of the scheduling variable. The main knob in LPV system identification is the selection of the number of pieces of the function. The presented LPV-PWA approach allows a good trade-off between model complexity and data fitting. In particular, it outperforms the method proposed in [6]. This result can be ascribed to the greater flexibility of PWA maps, where the polyhedral partition is learned from the data, with respect to the models used in [6, 89], where the unknown coefficients are a priori parametrized through a linear combination of given basis functions.

The results reported in chapter 2 are based on the publications [11, 24, 25]

- V. Breschi, D. Piga, and A. Bemporad. "Piecewise affine regression via recursive multiple least squares and multicategory discrimination", *Automatica*, vol. 73, pp.155–162, 2016.
- A. Bemporad, V. Breschi, D. Piga, "Piecewise Affine Regression via Recursive Multiple Least Squares and Multicategory Discrimination", Technical report TR-IMT-DYSCO-2016-01, 2016.
- V. Breschi, D. Piga, and A. Bemporad. "Identification of hybrid and linear parameter varying models via recursive piecewise affine regression and discrimination", in *Proc. European Control Conference* 2016, pp. 2632–2637.

1.1.2 Discrete Hybrid Automata and jump model learning

Even though PWA models are effective in describing complex phenomena, they do not provide any further insight on the mechanism leading to a switch between different modes, other than the polyhedral partition. However, when describing the behavior of systems characterized by the interaction between continuous and discrete dynamics, it might be important to capture the laws driving the transition from one operating regime to the other. Therefore, we consider the framework of hybrid

dynamical models, which are characterized by states assuming real and discrete values. The real-valued states describe the continuous dynamics of the system, which are usually driven by differential or difference equations, and the discrete-valued states indicates which sub-model is active at each time instant. Different hybrid models have been proposed in the literature based on the model for the process governing the discrete dynamics. Among them we mention: hybrid automata [57] and Discrete Hybrid Automata (DHA) [110], where the discrete dynamics of the system is modeled through a Finite State Machine (FSM); stochastic hybrid automata [13] , which are extensions of DHA to the stochastic case; Piecewise Deterministic Markov Processes; Switching Diffusion Processes; Stochastic Hybrid Systems [90]; Markov Jump Models (MJMs) [40], where the discrete state evolution is governed by a Markov Chain. In the thesis, we will also refer to Rarely Jump Models (RJMs), only assuming that the mode of the system changes rarely over time. Model belonging to this class are typically used in time-series segmentation [64], to divide an input time-series into a sequence of segments of finite length.
Even though the problem of identifying hybrid models from data has been studied by both computer scientists and control theorists, the literature lacks a well established unified framework for the identification of this class of systems. An overview of techniques for model identification of discrete events systems is provided in [31]. However, most of the works reviewed in [31] are devoted to the identification of the numbers of discrete states and the conditions leading to transitions between two operating regimes, while the problem of estimating the continuous dynamics is not addressed. The identification of both the continuous and the discrete dynamics of a timed automata is addressed in [109], through the use of a prefix tree and linear regression techniques. However, as the approach is limited to timed automata, only transitions triggered by time can be estimated. As it concerns the identification of MJM, research efforts have mainly been devoted to the design of Expectation Maximization (EM) methods [80, 81, 92, 107]. However, this techniques do not allow to process data sequentially and are, thus, not suited for on-line applications. An approach for recursive identification of Markov

Jump Linear Systems (MJLS), *i.e.*, MJM with linear continuous dynamics, is proposed in [38], where the sub-model parameters and the transition probabilities are estimated from ensemble properties of the overall MJLS. Another iterative method for MJLS is proposed in [36], which is based on recursive clustering and parameter estimation.

Identification of Discrete Hybrid Automata

In chapter 3 the problem of black-box identification of Discrete Hybrid Automata (DHA) is addressed. The proposed method for learning DHA relies only on a set of input/output data, assuming that no information on the discrete state of the system is available. Among the possible DHA, we focus on systems where the transition between different operating conditions is driven by either the regressor or the output crossing an (unknown) threshold. The ultimate objective is to retrieve both the parameters modeling the continuous dynamics of the systems and the law governing the transition between different models. The approach presented in chapter 3 relies on a proper extension of the method presented in Chapter 2 and in [24, 25], and it allows us to account for the information on the discrete dynamics that can be retrieved during the initial clustering phase. The method is still performed in two-steps: (i) simultaneous clustering and parameter estimation; (ii) computation of the polyhedral partitions. We also introduce a modification in the first stage of the method to handle modes sharing the same continuous dynamics. From both simulation and experimental tests, we show that the methods provide accurate models, while sharing the same characteristics of the methods proposed in Chapter 2 and in [24, 25] in terms of computational complexity.

The results in chapter 3 are based on the publication [28]

- V. Breschi, D. Piga, and A. Bemporad. "Learning hybrid models with logical and continuous dynamics via multiclass linear separation", in *Proc. Conference on Decision and Control (CDC)* 2016, pp. 353–358.

Learning jump models

Chapter 4 introduces two algorithms for learning Rarely Jump Models (RJMs) and Markov Jump Models (MJMs) from data. While for RJMs the mode is assumed to change rarely over time, for MJM the transitions between different operating conditions are assumed to be driven by an (unknown) Markov Chain. Not relying on measurements of the discrete state, the proposed approaches allow to estimate both the parameters characterizing the local models and the main features of the law driving the switches between different sub-models.

Consistently with the hypothesis on the discrete dynamics, the method tailored for RJMs learning is based on the minimization of a convex loss function penalizing a regularized fitting error with respect to the parameters of the local models and the number of time variations of the discrete state. This problem is solved through the sequential alternation of convex optimization, used to estimate the sub-model parameters, and dynamic programming, used to reconstruct the hidden sequence of discrete state. Note that a similar cost function is considered in [82] for model segmentation. However, in [82], the learning problem is solved in one step via convex optimization, approximating the number of mode variations in a *fused Lasso*-like fashion [109], *i.e.*, accounting for the difference between the parameters of the local model active at time t and the parameters of the local model active at time $t-1$.

The algorithm tailored for MJMs learning is based on a proper extension of the approach already presented in [24, 25]. In particular, instead of computing the polyhedral partition, the clusters obtained at the first step are refined with a probabilistic clustering method. This approach is designed to account for the hypothesis made on the law driving the discrete dynamics. The algorithm thus consists of two steps: (i) the simultaneous estimation of the (hidden) discrete state and update of the sub-model parameters via multi-model recursive least-squares; (ii) the refinement of the discrete-state sequence estimate and the update of the mode transition matrix. Besides its computational efficiency, one of the main advantages of the proposed algorithm with respect to other MJM learning

methods based on Expectation Maximization (*e.g.*, [80, 92, 107]) is that it processes the data sequentially, and thus it is suited for on-line learning. Test performed on both simulation and experimental data show the effectiveness of the methods.

The results in this chapter have been extended in the paper [12]:

- A. Bemporad, V. Breschi, D. Piga, S. Boyd. "Fitting Jump Models", Submitted for publication, Available at https://arxiv.org/abs/1711.09220.

1.2 Energy disaggregation problem

Among the possible applications of hybrid models, we will consider the problem of energy disaggregation, which aims at estimating the consumption of the single appliances in an household from aggregated power measurements.
Providing users with information on their energy-use behavior is of key importance in reducing energy consumptions and, thus, energy billings. Particular relevance has the real-time supply of appliance-level information [42], that can yield annual energy savings up to 12% of the total energy consumed in an household. On the one hand, a possible strategy to acquire information on the appliances is to use smart devices or smart plugs. However, this requires multiple sensors to be deployed, thus increasing installation and maintenance costs. On the other hand, a single-point smart meter, measuring the overall household consumption, can be used in combination with software techniques to retrieve appliance-level information from aggregate data. This choice allows one to lower costs and to reduce the amount of data to be processed.
Non-Intrusive Load Monitoring (NILM) techniques have been introduced to handle the disaggregation problem, requiring minimum information on the appliances behavior. A NILM approach for energy disaggregation was studied from Hart in the early 90's [54]. The method, based on five stages matched to the appliances signatures, allows one to detect and

track ON/OFF configurations of the single appliances but it poorly performs in the presence of (i) multi-state and varying-load appliances and when (ii) multiple ON/OFF events happen simultaneously. Following the seminal work by Hart, various solutions for non-intrusive load monitoring have been proposed in the literature. A review of existing disaggregation methods can be found in [117, 119]. NILM techniques usually fall into two categories: (i) machine learning methods [61, 87, 99, 105]; (ii) optimization-based strategies [32, 46, 106]. Among approaches based on machine learning, [61, 65, 87] use Hidden Markov Models (HMM) to represent the behavior of the single appliance and then a Factorial Hidden Markov Model (FHMM) to describe the overall household consumption pattern. Instead, the methods proposed in [32, 106] are based on integer programming and the one in [46] employs sparse coding techniques. All the aforementioned NILM methods have shown good performance in estimating the fraction of energy consumed by each appliance. However, they (i) do not reproduce accurately the appliances' consumption patterns over time and (ii) provide low-quality estimates when multiple appliances operate simultaneously. To overcome this limitations, an optimization-based method and a machine learning approach have been introduced in [88] and [39], respectively. While in [88] disaggregation is treated as a least-squares error minimization problem, [39] proposes a NILM method relying on the combined use of FHMM and Iterative Subsequence Dynamic Time Warping (ISDTW). Both methods can handle (i) the presence of multi-state appliances and (ii) multiple devices operating at the same time.

Two approaches for Non-Intrusive Load Monitoring (NILM) are presented in chapter 5, both relying on the assumption that consumption models for each appliance are available. In particular, the method proposed in chapter 4 for Rarely Jump Models learning is used to retrieve such models.
At first, disaggregation is addressed solving an optimization problem with least-square cost function, also accounting for the history of the appliances' consumption. The second approach is based on the use of multiple-model Kalman Filters [7, Chapter 11]. Consequently, energy

disaggregation is treated as a state estimation problem. In their initial formulation, both methods require to span all the tree of possible combinations of appliances modes to estimate the appliance profiles from the aggregate measurements. This might cause the proposed approaches to be inefficient when the number of appliances increases. We propose different heuristics to reduce the computational complexity of the methods. The results reported in chapter 5 show that the two NILM approaches allow us to handle (i) appliances with multiple states (*e.g.*, cloth dryer and dishwasher), thus not limiting the approaches to the identification of ON/OFF configurations, and to perform (ii) real-time power disaggregation, as all the strategies are based on the iterative processing of the collected aggregated measurements.

The results presented in chapter 5 are based on:

- V. Breschi, D. Piga, A. Bemporad, "Online end-use energy disaggregation via jump linear models", Working Paper.
- V. Breschi, D. Piga, A. Bemporad, "Jump model learning and filtering for energy-use disaggregation", Submitted to *SYSID 2018* [29].

1.3 Collaborative estimation

In this thesis, we also consider the dual problem of estimating a single model using data collected from multiple similar processes by means of collaborative estimation. In this case, our goal is to estimate a set of unknown parameters from observations collected from a multitude of systems, referred to as *agents* and *nodes*, which are assumed to share the same model, such as a set of devices produced in series that are usually equal.

The increasing connectivity between consumer devices have stimulated a growing interest towards distributed solution for a variety of problems, *e.g.*, in state estimation [83], control [50], machine learning [47] and fault diagnosis [20]. The main motivations encouraging research

on distributed strategies are (i) the reduction of the overall transmission complexity and (ii) the robustness of these solutions to node failures, as they are based only on communications between neighboring nodes.
Methods for parameter estimation over networks have been extensively studied in the context of Wireless Sensors Networks (WSNs), where the low computational power of the nodes demands for a reduction in the complexity of both the operations and the transmissions that have to be performed locally. The approaches proposed in the literature can be divided into three groups: (i) incremental methods [73, 95, 100], (ii) diffusion approaches [4, 21, 33, 34] and (iii) consensus-based distributed strategies [35, 75, 77, 102, 103, 113, 118]. While incremental methods require a cycle in the network, the other approaches do not constrain the network topology. Even though, in principle, diffusion and consensus based methods allow one to reduce transmission complexity and multihop communications, without constraining the network topology, different groups of neighbor nodes are required to share information to reach consensus over the whole network. Consensus-based estimators are usually designed through the solution of a convex optimization problem, which is commonly solved through the Alternating Direction Method of Multipliers (ADMM) [23].

In the last years research interest was mainly directed to the design of distributed solutions for estimation over groups of agents. However, recent advances in cloud-computing technologies [76] induce reconsidering more centralized strategies. These techniques rely on the fact that each node in the network can have on-demand access to shared resources, ideally characterized by unlimited computational power and memory. These resources can be acquired and released with minimum effort.
As a motivating example for possible applications of cloud-aided estimation, consider a fleet of vehicles connected to the cloud (see Figure 1). In this framework, suppose that a set of unknown parameters of interest for fault detection and/or diagnosis must be estimated. The measurements taken on-board of the vehicles can be used to compute local estimates of the unknowns. On the cloud, the local estimates can be further refined,

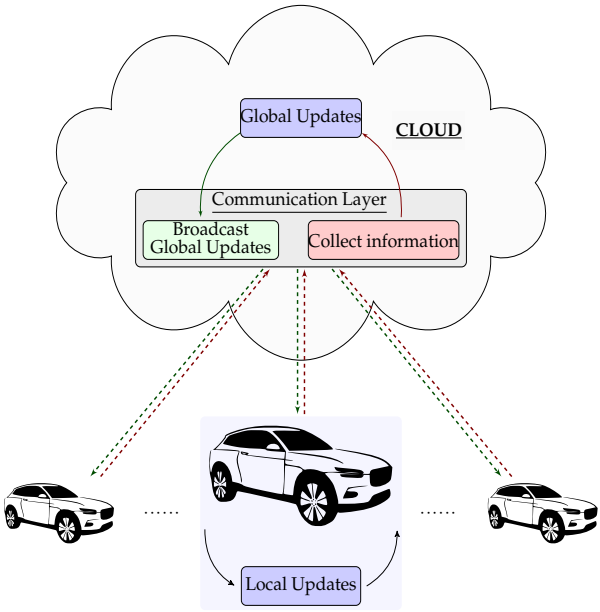


Figure 1: Cloud-connected vehicles.

accounting for the fleet behavior, and they can be fused, to identify parameters that might be common to all the vehicles.
Potential applications of cloud-aided consensus-based approaches are the prognostics of automotive fuel pumps [108] and brake pads [58]. In these cases, assuming that wear and fuel consumption models are known a priori, the component wear-rate as a function of the workload (cumulative fuel flow or energy dissipated in the brakes) can be considered as a global parameter. Note that the use of cloud computing for automotive applications has already been introduced in [84], for cloud-aided speed trajectory optimization, and in [69, 70], where cloud computing based solutions for route planning are proposed.

Chapter 6-7 present ADMM schemes for cloud-aided collaborative parameter estimation. All the methods are designed under the hypothesis that (i) the information exchanged between the cloud and the nodes

is not corrupted by noise and that (ii) all the nodes are described by the same model. Only one method relies on the hypothesis that all the parameters to be estimated are common to all the agents. The other approaches address more general settings as: (i) estimation of both global and purely local parameters with linear consensus constraints; (ii) identification in presence of nonlinear consensus constraints; (iii) multi-class estimation. The collaborative learning problem is always solved through a two-step strategy. In particular: (i) each node recursively computes a local estimate of the unknowns (through RLS), using the measurements acquired by the local sensors, then (ii) global computations are performed on the cloud to refine the local estimates and to update the estimates of the global parameters.
As remarked in [75], the main disadvantage due to the use of ADMM is the introduction of two time scales, with the local time-scale determined by the nodes' clocks, and the cloud time-scale depending on the resources available in the center of fusion and the selected stopping criteria used to terminate the ADMM iterations. To overcome this problem, collaborative estimation is addressed considering both Node-to-Cloud-to-Node (N2C2N) communication schemes and Node-to-Cloud (N2C) transmissions. The use of N2C transmission allows us to alleviate problems due to the communication latency between the agents and the cloud. Thanks to the use of RLS, the approaches proposed (i) are suited for online estimation and they can be (ii) easily integrated with preexisting Recursive Least-Squares (RLS) estimators, already running on board of the nodes.

The results in chapters 6 and 7 are based on the working paper [26] and the technical report [27]:

- V. Breschi, I. Kolmanovsky, and A. Bemporad. "Cloud-aided collaborative estimation by ADMM-RLS algorithms for connected vehicle prognostics", Submitted to the *American Control Conference 2018*.

- V. Breschi, I. Kolmanovsky, and A. Bemporad. "Cloud-aided collaborative estimation by ADMM-RLS algorithms for connected ve-

hicle prognostics", Technical report, Available at https://arxiv.org/abs/1709.07972, 2017.

Chapter 2

A novel approach to PWA regression

A numerically efficient two-stage method for Piecewise Affine (PWA) regression is presented in this Chapter. In particular, Section 2.1 is devoted to the formalization of the PWA regression problem, while Section 2.2 presents the proposed PWA regression algorithm. The first stage of the method, which relies on the combined use of clustering and multi-model Recursive Least-Squares (RLS), is presented in Section 2.2.1. In Section 2.2.2, two algorithms for piecewise linear multi-category discrimination are introduced. Three case studies are then presented in Section 2.3, to show the effectiveness of the approach. The proposed PWA regression technique is then used in Section 2.4 for data-driven modeling of PWA autoRegressive dynamical systems with eXogenous inputs (PWARX) and Linear Parameter Varying (LPV) systems. When the problem of LPV system identification is tackled with the PWA regression approach, the dependence of the model coefficients on the scheduling variable is approximated through a PWA map.

2.1 Problem statement

A vector-valued PWA function $f : \mathcal{X} \to \mathbb{R}^{n_y}$ is a collection of affine sub-models, defined as

$$f(x) = \begin{cases} \theta_1 \begin{bmatrix} 1 \\ x \end{bmatrix} & \text{if } x \in \mathcal{X}_1 \\ \vdots & \\ \theta_s \begin{bmatrix} 1 \\ x \end{bmatrix} & \text{if } x \in \mathcal{X}_s \end{cases}, \tag{2.1}$$

where $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$, $s \in \mathbb{N}$ denotes the number of affine local models defining f, $\theta_i \in \mathbb{R}^{n_y \times (n_x+1)}$ are parameter matrices, and the sets $\{\mathcal{X}_i\}_{i=1}^s$ are polyhedrons, forming a complete partition of the domain $\mathcal{X}$[1]. The function f in (2.1) is not assumed to be continuous over the boundaries of the polyhedra $\{\mathcal{X}_i\}_{i=1}^s$. Therefore, in the definition of the sets $\mathcal{X}_i$, $i = 1, \ldots, s$, some inequalities can be replaced by strict inequalities, to avoid ambiguities when evaluating f on the boundary between neighboring polyhedra, namely to avoid that f might take multiple values at the boundaries of $\{\mathcal{X}_i\}_{i=1}^s$.

The PWA regression problem addressed in this chapter aims at computing a PWA map f fitting a given set of N regressor/output pairs $\{x(k), y(k)\}_{k=1}^N$, such that $y(k) \approx f(x(k))$.

Estimating a PWA map f (see eq.(2.1)) from data requires:

(i) choosing the number of affine sub-models s;

(ii) estimating the parameter matrices $\{\theta_i\}_{i=1}^s$, that characterize the affine local models of the map f;

(iii) learning the polyhedral partition $\{\mathcal{X}_i\}_{i=1}^s$ of the domain $\mathcal{X}$, where the local models are defined.

[1] A complete polyhedral partition of space $\mathcal{X}$ is a collection of sub-spaces $\{\mathcal{X}_i\}_{i=1}^s$, such that $\bigcup_{i=1}^s \mathcal{X}_i = \mathcal{X}$ and $\mathring{\mathcal{X}}_i \cap \mathring{\mathcal{X}}_j = \emptyset, \forall i \neq j$, with $\mathring{\mathcal{X}}_i$ denoting the interior of $\mathcal{X}_i$.

We assume that s is fixed by the user. The value of s can be chosen through cross-validation based procedures, with a possible upper-bound dictated by the maximum tolerable complexity of the estimated model. When choosing s, one must take into account the trade-off between data fit and model complexity to avoid overfitting the data, with consequent poor generalization on unseen data. This is related to one of the most crucial aspects in function learning, known as *bias-variance trade-off* [112].

2.2 PWA regression algorithm

We tackle the PWA regression problem in two stages: Stage S1, which allows us to iterative cluster the data and estimate the parameters $\{\theta_i\}_{i=1}^s$; Stage S2, where we compute the polyhedral partition of the domain $\mathcal{X}$.

2.2.1 S1: recursive clustering and parameter estimation

Stage S1 is carried out as described in Algorithm 2, extending the computationally efficient approach presented in [2] for solving RLS problems using inverse QR decomposition to the case of multiple linear regressions.

Algorithm 2 updates the clusters and the model parameters iteratively and thus it is also suitable for on-line applications, where data are acquired in real time.

Step 2 initializes the inverse matrices $T^{i,j}$ needed by the RLS updates (see Step 1 of Algorithm 1) at a (large) value δI_{n_x+1}, where δ is a large number, for all output components $j = 1, \ldots, n_y$ and for all local linear models $i = 1, \ldots, s$. After computing the estimation error $e_i(k)$ for all models i at Step 3.1, Step 3.2 picks up the "best" sub-model $i(k)$, to which the current sample $x(k)$ must be assigned to, based on a trade-off between reducing the prediction error $e_i(k)$ and penalizing the weighted distance between $x(k)$ and the centroid c_i. The chosen clustering rule (see (S1.1)) is similar to the criterion used in [5] for on-line PWA regression. However, in [5] no guidance on how properly weight the prediction

Algorithm 1 Multi-model RLS based on inverse QR decomposition

Input: Observations $\{x(k), y(k)\}_{k=1}^{N}$, number s of affine sub-models, sub-model to be updated $\{i(k)\}_{i=1}^{N}$, forgetting factor κ, $0 < \kappa \leq 1$, inverse matrix init parameter δ, $\delta \gg 1$.

1. **let** $T^{i,j}(0) \leftarrow \delta I_{n_x+1}$, $j = 1, \ldots, n_y$, $i = 1, \ldots, s$;
2. **for** $k = 1, \ldots, N$ **do**
   - 2.1. **for** $j = 1, \ldots, n_y$ **do**
     - 2.1.1. **let** $u \leftarrow \mathbf{0_{n_x+1}}$, $b \leftarrow 1$;
     - 2.1.2. **for** $\ell = 1, \ldots, n_x + 1$ **do**
       - 2.1.2.1. $a \leftarrow \frac{1}{\sqrt{\kappa}} \sum_{h=1}^{\ell} [T^{i(k),j}]_{\ell,h} x_h(k)$;
       - 2.1.2.2. $b_1 \leftarrow b$; $b \leftarrow \sqrt{b^2 + a^2}$;
       - 2.1.2.3. $\sigma \leftarrow \frac{a}{b}$, $\rho \leftarrow \frac{b_1}{b}$;
       - 2.1.2.4. **for** $t = 1, \ldots, i$ **do**
         $d \leftarrow [T^{i(k),j}]_{\ell,t}$; $[T^{i(k),j}]_{\ell,t} \leftarrow \frac{1}{\sqrt{\kappa}} \rho d - \sigma u_t$;
         $u_t \leftarrow \rho u_t + \frac{1}{\sqrt{\kappa}} \sigma d$;
       - 2.1.2.5. **end for**;
     - 2.1.3. **end for**;
     - 2.1.4. **update** $[\theta_{i(k)}]_{j,:} \leftarrow [\theta_{i(k)}]_{j,:} + \frac{e_{i(k)}}{b} u'$;
   - 2.2. **end for**;
3. **end for**

Output: Estimated matrices $\{\theta_i\}_{i=1}^{s}$.

Algorithm 2 Recursive clustering and parameter estimation algorithm

Input: Observations $\{x(k), y(k)\}_{k=1}^{N}$, desired number s of affine submodels, noise covariance matrix Λ_e, forgetting factor κ, $0 < \kappa \leq 1$, inverse matrix init parameter δ, $\delta \gg 1$; initial condition for matrices θ_i, cluster centroids c_i, and covariance matrices R_i, $i = 1, \ldots, s$.

1. **let** $\mathcal{C}_i \leftarrow \emptyset$, $i = 1, \ldots, s$;
2. **let** $T^{i,j}(0) \leftarrow \delta I_{n_x+1}$, $j = 1, \ldots, n_y$, $i = 1, \ldots, s$;
3. **for** $k = 1, \ldots, N$ **do**
   - 3.1. **let** $e_i(k) \leftarrow y(k) - \theta_i \begin{bmatrix} 1 \\ x(k) \end{bmatrix}$, $i = 1, \ldots, s$;
   - 3.2. **let** $\varepsilon_i = x(k) - c_i$ **then**

$$i(k) \leftarrow \arg \min_{i=1,\ldots,s} \varepsilon_i' R_i^{-1} \varepsilon_i + e_i(k)' \Lambda_e^{-1} e_i(k); \tag{S1.1}$$

   - 3.3. **let** $\mathcal{C}_{i(k)} \leftarrow \mathcal{C}_{i(k)} \cup \{x(k)\}$;
   - 3.4. **update** $\theta_{i(k)}$ with Algorithm 1;
   - 3.5. **let** $\delta c_{i(k)} \leftarrow \frac{1}{|\mathcal{C}_{i(k)}|}(x(k) - c_{i(k)})$;
   - 3.6. **let** $c_{i(k)} \leftarrow c_{i(k)} + \delta c_{i(k)}$;
   - 3.7. **update** the cluster covariance matrix $R_{i(k)}$ for cluster $\mathcal{C}_{i(k)}$ through the matrix inversion Lemma

$$Q \leftarrow R_{i(k)}^{-1} - \frac{R_{i(k)}^{-1}\left(x(k) - c_{i(k)}\right)\left(x(k) - c_{i(k)}\right)' R_{i(k)}^{-1}}{|\mathcal{C}_{i(k)}| - 2 + \left(x(k) - c_{i(k)}\right)' R_{i(k)}^{-1}\left(x(k) - c_{i(k)}\right)};$$

$$R_{i(k)} \leftarrow \frac{|\mathcal{C}_{i(k)}| - 1}{|\mathcal{C}_{i(k)}| - 2}\left(Q - \frac{Q\delta_{c_i(k)}\delta_{c_i(k)}' Q}{\frac{|\mathcal{C}_{i(k)}|-2}{|\mathcal{C}_{i(k)}|-1} + \delta_{c_i(k)}' Q \delta_{c_i(k)}}\right);$$

4. **end for**;
5. **end**.

Output: Estimated matrices $\{\theta_i\}_{i=1}^{s}$, centroids $\{c_i\}_{i=1}^{s}$, clusters $\{\mathcal{C}_i\}_{i=1}^{s}$, covariance matrices $\{R_i\}_{i=1}^{s}$.

error and the distance from the cluster centroids is provided.
At Step 3.4 only the parameter vector $\theta_{i(k)}$ associated to the selected sub-model $i(k)$ is updated, using the extension of the inverse QR factorization algorithm of [2] presented in Algorithm 1. Steps 3.6 and 3.7 recursively update the centroid $c_{i(k)}$ and the inverse of the covariance matrix $R_{i(k)}$ of cluster $\mathcal{C}_{i(k)}$, respectively. The centroids and covariance matrices of the remaining clusters, $\mathcal{C}_j$, with $j \in \{1, \dots, s\}$ and $j \neq i(k)$, are not updated.

Algorithm 2 requires an initial guess for the parameter matrices θ_i, the cluster centroids c_i and the covariance matrices R_i, $i = 1, \dots, s$. Because of the greedy nature of Algorithm 2, the final estimate depends on the selected initial conditions, and no fit criterion to minimize $\{y(k) - f(x(k))\|_2^2\}_{k=1}^N$ is optimized. Zero matrices θ_i, randomly chosen centroids c_i, and identity covariance matrices R_i are a possible initialization. In alternative, if Algorithm 2 can be executed in a batch mode, one can initialize the parameter matrices $\theta_1, \dots, \theta_s$ all equal to the best linear model

$$\theta_i \equiv \arg\min_{\theta} \sum_{k=1}^{N} \left\| y(k) - \theta \begin{bmatrix} 1 \\ x(k) \end{bmatrix} \right\|_2^2, \ \forall i = 1, \dots, s \tag{2.2}$$

that fits all data. The regressors $\{x(k)\}_{k=1}^N$ can be classified through K-means and, based on the results of clustering, one can compute the centroids

$$c_i = \frac{1}{|\mathcal{C}_i|} \sum_{x(k) \in \mathcal{C}_i} x(k)$$

and the inverse of the covariance matrices

$$R_i = \frac{1}{|\mathcal{C}_i| - 1} \sum_{x(k) \in \mathcal{C}_i} (x(k) - c_i)(x(k) - c_i)'.$$

When working in a batch mode, estimation quality may be improved executing Algorithm 2 multiple times, using its output as initial condition for its following execution.

The chosen clustering policy (see (S1.1)) depends on the noise covariance matrix Λ_e. However, a prior knowledge of the noise covariance

matrix Λ_e is barely available in practice. A possible choice for Λ_e can be, for instance, $\Lambda_e = I_{n_y}$. Alternatively, if Algorithm 2 is executed in a batch mode and iteratively repeated, Λ_e can be approximated with the sample covariance matrix computed at the end of each execution as

$$\hat{\Lambda}_e = \frac{1}{N}\sum_{i=1}^{s}\sum_{\substack{k=1 \\ x(k)\in\mathcal{C}_i}}^{N}\left(y(k) - \theta_i\begin{bmatrix}1\\ x(k)\end{bmatrix}\right)\left(y(k) - \theta_i\begin{bmatrix}1\\ x(k)\end{bmatrix}\right)'.$$

Cluster selection: a stochastic interpretation

Differently from what is done in [5], we provide a stochastic interpretation for the chosen clustering criteria (see (S1.1)). Based on (S1.1), vector $x(k)$ is assigned to a cluster $\mathcal{C}_{i(k)}$ by trading off between minimizing the (weighted) regression error and the (weighted) distance between $x(k)$ and the clusters' centroids.
Assume that the conditional *probability density function* $p_x(x(k)|x(k) \in \mathcal{C}_i)$ is a Gaussian function centered at the centroid c_i with covariance matrix R_i, *i.e.*,

$$p_x(x(k)|x(k)\in\mathcal{C}_i) \propto \exp\left\{-\frac{1}{2}\,(x(k)-c_i)'R_i^{-1}\,(x(k)-c_i)\right\}.$$

Furthermore, suppose that the residual $e_i(k)$ (see step 3.1) given that $x(k)$ belongs to cluster $\mathcal{C}_i$ follows a Gaussian distribution with zero mean and covariance matrix Λ_e. Thus, the conditional *probability density function* $p_y(y(k)|x(k), x(k) \in \mathcal{C}_i)$ is equal to

$$p_y\left(y(k) - \theta_i\begin{bmatrix}1\\ x(k)\end{bmatrix}\right) \propto \exp\left\{-\frac{1}{2}\left(y(k)-\theta_i\begin{bmatrix}1\\ x(k)\end{bmatrix}\right)'\Lambda_e^{-1}\left(y(k)-\theta_i\begin{bmatrix}1\\ x(k)\end{bmatrix}\right)\right\}.$$

The criterion in (S1.1) thus maximizes over $i = 1, \ldots, s$ the *conditional posterior probability* $p_{xy}(x(k), y(k)|x(k) \in \mathcal{C}_i)$. The conditional posterior probability density function, which is equal to the product of $p_x(x(k)|x(k) \in \mathcal{C}_i)$ and $p_y(y(k)|x(k), x(k) \in \mathcal{C}_i)$ and, thus, it is proportional to

$$p_{xy}(x(k), y(k)|x(k) \in \mathcal{C}_i) \propto \exp\left\{-\frac{1}{2}e_i(k)'\Lambda_e^{-1}e_i(k) - \frac{1}{2}\varepsilon_i(k)'R_i^{-1}\varepsilon_i(k)\right\}.$$

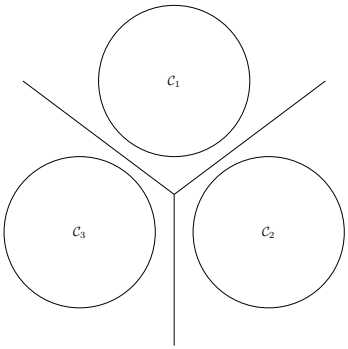


Figure 2: Three clusters separable by PWA functions

2.2.2 S2. Partitioning space $\mathcal{X}$

We propose a variation of the multi-category discrimination technique of [17] to separate the clusters $\{\mathcal{C}_i\}_{i=1}^s$ that partition the domain $\mathcal{X}$ in a much more efficient computational way, especially when dealing with a large number N of data points. The linear multi-category discrimination problem aims at computing a PWA separator function $\phi : \mathbb{R}^{n_x} \to \mathbb{R}$, discriminating between the clusters $\{\mathcal{C}_i\}_{i=1}^s$ (see Figure 2). The separator ϕ is defined as the maximum of s affine functions $\{\phi_i(x)\}_{i=1}^s$, *i.e.*,

$$\phi(x) = \max_{i=1,\dots,s} \phi_i(x). \tag{2.3}$$

Each affine function $\phi_i(x)_{i=1}^s$ is described by the parameters $\omega^i \in \mathbb{R}^{n_x}$ and $\gamma^i \in \mathbb{R}$, namely:

$$\phi_i(x) = \begin{bmatrix} x' & -1 \end{bmatrix} \begin{bmatrix} \omega^i \\ \gamma^i \end{bmatrix}. \tag{2.4}$$

Denote with m_i the cardinality of cluster $\mathcal{C}_i$, $i = 1, \dots, N$, where $\{\mathcal{C}_i\}_{i=1}^s$ are the clusters learned from data at stage S1. For $i = 1, \dots, s$, let M_i be a $m_i \times n_x$ dimensional matrix obtained by stacking the regressors $x(k)'$ belonging to $\mathcal{C}_i$ in its rows.
In case of piecewise linearly separable clusters, $\{\phi_i\}_{i=1}^s$ satisfy the inequality

$$\begin{bmatrix} M_i & -1_{m_i} \end{bmatrix} \begin{bmatrix} \omega^i \\ \gamma^i \end{bmatrix} > \begin{bmatrix} M_i & -1_{m_i} \end{bmatrix} \begin{bmatrix} \omega^j \\ \gamma^j \end{bmatrix}, \quad i, j = 1, \dots, s, \ i \neq j,$$

or, equivalently,

$$\begin{bmatrix} M_i - 1_{m_i} \end{bmatrix} \begin{bmatrix} \omega^i \\ \gamma^i \end{bmatrix} \geq \begin{bmatrix} M_i - 1_{m_i} \end{bmatrix} \begin{bmatrix} \omega^j \\ \gamma^j \end{bmatrix} + 1_{m_i}, \quad i, j = 1, \ldots, s, \; i \neq j, \quad (2.5)$$

with the constant vector 1_{m_i} on the right side of (2.5) used only for normalization purposes.

The piecewise-affine separator ϕ thus satisfies the conditions:

$$\begin{cases} \phi(x) = \begin{bmatrix} x' & -1 \end{bmatrix} \begin{bmatrix} \omega^i \\ \gamma^i \end{bmatrix}, \; \forall x \in \mathcal{C}_i, \;\; i = 1, \ldots, s, \\ \phi(x) \geq \begin{bmatrix} x' & -1 \end{bmatrix} \begin{bmatrix} \omega^j \\ \gamma^j \end{bmatrix} + 1, \; \forall x \in \mathcal{C}_i, \; i \neq j. \end{cases} \quad (2.6)$$

Based on the definition of ϕ (see (2.3)) each polyhedron $\mathcal{X}_i$ turns out to be described by

$$\mathcal{X}_i = \{x \in \mathcal{X} : \phi(x) = \phi_i(x)\},$$

which can be further modified as

$$\mathcal{X}_i = \left\{ x \in \mathbb{R}^{n_x} : \begin{bmatrix} x' & -1 \end{bmatrix} \begin{bmatrix} \omega^i - \omega^j \\ \gamma^i - \gamma^j \end{bmatrix} \geq 1, \; j = 1, \ldots, s, \; j \neq i \right\},$$

accounting for the conditions in (2.6).

Rather than solving a linear program as in [17], the parameters $\{\omega^i, \gamma^i\}_{i=1}^s$ are computed solving the convex unconstrained optimization problem

$$\min_{\{\omega^i, \gamma^i\}_{i=1}^s} r(\{\omega^i, \gamma^i\}_{i=1}^s) + \sum_{i=1}^{s} \sum_{\substack{j=1 \\ j \neq i}}^{s} \frac{1}{m_i} \left\| \left(\begin{bmatrix} M_i & -1_{m_i} \end{bmatrix} \begin{bmatrix} \omega^j - \omega^i \\ \gamma^j - \gamma^i \end{bmatrix} + 1_{m_i} \right)_+ \right\|_2^2, \quad (2.7)$$

where

$$r(\{\omega^i, \gamma^i\}_{i=1}^s) = \frac{\lambda}{2} \sum_{i=1}^{s} \left(\|\omega^i\|_2^2 + (\gamma^i)^2 \right), \quad \lambda > 0$$

is an ℓ_2-regularization term introduced to better conditioning problem (2.7) and to guarantee that (2.7) has a unique solution. Tuning the hyperparameter λ through cross-validation, the ℓ_2-regularization term may lead to an improvement in the generalization performance of the final separator ϕ.

Problem (2.7) generates a piecewise-affine function that minimizes the (averaged) squared 2-norm of the violation of the inequalities (2.5).

Multi-category discrimination: batch mode solution

Problem (2.7) is solved by using a regularized piecewise-smooth Newton method with Armijo's line search similar to the one proposed in [10] for functions $g : \mathbb{R}^{n_\xi} \to \mathbb{R}$ of the form

$$g(\xi) = \frac{\lambda}{2}\|\xi\|_2^2 + \sum_{j=1}^{n_g} \|g_j(\xi)_+\|_2^2, \tag{2.8}$$

with $g_j : \mathbb{R}^{n_\xi} \to \mathbb{R}$ convex and twice continuously differentiable functions. In particular, we exploit the linearity of the functions g_j's.
Solving (2.7), the optimization vector is $\xi = [(\omega^1)' \dots (\omega^s)' \gamma^1 \dots \gamma^s]'$, with $\xi \in \mathbb{R}^{n_\xi}$, $n_\xi = s(n_x + 1)$, and g_j's are affine functions:

$$g_j(\xi) = a_j'\xi - b_j, \ j = 1, \dots, n_g, \tag{2.9}$$

where $n_g = N(s-1)$ and $a_j \in \mathbb{R}^{n_\xi}$, $b_j \in \mathbb{R}$ are obtained from (2.7) as a function of matrices $\{M_i\}_{i=1}^s$ and coefficients $\{m_i\}_{i=1}^s$.
Let

$$\begin{aligned} \mathcal{A} &= [a_1 \ \dots \ a_{ng}]', \\ \mathcal{B} &= [b_1 \ \dots b_{n_g}]', \end{aligned} \tag{2.10}$$

and, given $\xi \in \mathbb{R}^{n_\xi}$, denote $I(\xi) = \{i \in \{1, \dots, n_g\} : \mathcal{A}_i\xi - \mathcal{B}_i > 0\}$. Then the function to minimize, its gradient and its generalized Hessian at ξ are

$$g(\xi) = \frac{\lambda}{2}\xi'\xi + \sum_{i \in I(\xi)} (\mathcal{A}_i\xi - \mathcal{B}_i)^2 \tag{2.11a}$$

$$\nabla g(\xi) = \lambda\xi + \mathcal{A}_{I(\xi)}'(\mathcal{A}_{I(\xi)}\xi - \mathcal{B}_{I(\xi)}) \tag{2.11b}$$

$$\nabla^2 g(\xi) = \lambda I + \mathcal{A}_{I(\xi)}'\mathcal{A}_{I(\xi)} = \lambda I + \sum_{i \in I(\xi)} \mathcal{A}_i'\mathcal{A}_i \tag{2.11c}$$

The proposed approach to solve (2.7), summarized in Algorithm 3, uses the solution d of the linear system

$$(\nabla^2 g(\xi) + \delta(\xi) I)d = -\nabla g(\xi) \tag{2.12}$$

at the current ξ as a search direction, where $\delta(\xi) = \zeta\|\nabla g(\xi)\|$ and $\zeta \in (0, 1)$. Thanks to the structure of $\nabla^2 g$ in (2.11c), the linear system (2.12) is solved

as the least squares problem

$$\min_d \frac{1}{2} \left\| \begin{bmatrix} \mathcal{A}_{I(\xi)} \\ \sqrt{\lambda + \delta(\xi)} I_{n_\xi} \end{bmatrix} d + \begin{bmatrix} \mathcal{A}_{I(\xi)}\xi - \mathcal{B}_{I(\xi)} \\ \frac{\lambda}{\sqrt{\lambda+\delta(\xi)}}\xi \end{bmatrix} \right\|_2^2 \tag{2.13}$$

using the QR factorization of $\begin{bmatrix} \mathcal{A}_{I(\xi)} \\ \sqrt{\lambda+\delta(\xi)} I_{n_\xi} \end{bmatrix}$ (steps 5.1–5.2). Since $\nabla g(\xi) > 0$ during the iterations, $\delta(\xi)$ is also positive. Therefore, R is full column rank and the upper-triangular linear system in Step 5.2 is always solvable.

Algorithm 3 requires one to choose an initial guess for ξ and the values of ζ and λ. An initial guess for $\xi \in \mathbb{R}^{n_\xi}$ can be obtained by running Algorithm 3 first on decimated clusters. The computed solution can then be used as the new initial condition for the full problem with N regressors. Numerical experiments have shown that allowing a varying ζ, given by

$$\zeta = \zeta_0 \frac{\min\{1, \|\nabla g\|\}}{\|\nabla g\|},$$

with $0 < \zeta_0 \ll 1$, reduces the number of iterations and prevents excessive regularization in (2.12) when $\|\nabla g\|$ is large.

Concerning the ℓ_2 hyper-parameter λ, on the one hand, setting $\lambda > 0$ complicates the number of operations required by the algorithm at each iteration (in particular to compute the solution of (S2.1)) and bias the solution with respect to the PWA multi-category discrimination function minimizing only the squared 2-norm of the violations of the inequalities (2.6). On the other hand, the choice of $\lambda > 0$ leads to a smaller number k of iterations, and overall to a reduced computation time.

On-line multi-category discrimination

Let us treat $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ as a random vector and let us assume that there exists an "oracle" function $i : \mathbb{R}^{n_x} \to \{1, \dots, s\}$, that assigns to any $x \in \mathbb{R}^{n_x}$ the corresponding mode $i(x) \in \{1, \dots, s\}$. By this definition, the "oracle" function implicitly defines clusters in the space $\mathcal{X}$. We further suppose that

$$\pi_i = \text{Prob}[i(x) = i] = \int_{\mathbb{R}^{n_x}} \delta(i, i(x)) p(x) dx$$

Algorithm 3 Piecewise-smooth Newton method for multi-category discrimination

Input: Regressors $\{x(k)\}_{k=1}^N$, clusters $\mathcal{C}_i$, $i = 1, \ldots, s$; $\sigma \in (0, 1/2)$, $\zeta \in (0, 1)$; ℓ_2-regularization hyper-parameter $\lambda \geq 0$; initial guess $\xi \in \mathbb{R}^{n_\xi}$; maximum number K of iterations; tolerances $g_{\rm tol} > 0$ and $\delta_{\rm tol} > 0$.

1. **Initialize** matrices $M_i \in \mathbb{R}^{m_i \times n_x}$, $i = 1, \ldots, s$; $n_\xi \leftarrow s(n_x + 1)$, $n_g \leftarrow N(s-1)$; **define** $\mathcal{A}$, $\mathcal{B}$ as in (2.9)–(2.10), $j = 1, \ldots, n_g$;
2. $k \leftarrow 0$;
3. $c \leftarrow \mathcal{A}\xi - \mathcal{B}$; $I \leftarrow \{i \in \{1, \ldots, n_g\} : \ c_i \geq 0\}$;
4. $g \leftarrow c_I'c_I + \frac{\lambda}{2}\xi'\xi$; $\nabla g \leftarrow \mathcal{A}_I'c_I + \lambda\xi$; $\delta \leftarrow \zeta\|\nabla g\|$;
5. **while** $g > g_{\rm tol}$ **and** $\delta > \delta_{\rm tol}$ **and** $k < K$ **do**
   5.1. $(Q, R) \leftarrow$ QR factorization of $\begin{bmatrix} \mathcal{A}_I \\ \sqrt{\lambda+\delta} I_{n_\xi} \end{bmatrix}$;
   5.2. **solve** the upper-triangular linear system
   $$R_{\{1,\ldots,n_\xi\}} d = -(Q_{\{1,\ldots,|I|\},\{1,\ldots,n_\xi\}})' c_I + - \frac{\lambda}{\lambda+\delta}(Q_{\{|I|+1,\ldots,|I|+n_\xi\},\{1,\ldots,n_\xi\}})'\xi; \tag{S2.1}$$
   5.3. $\alpha \leftarrow 1$; $q \leftarrow \mathcal{A}d$; $\xi_\alpha \leftarrow \xi + d$;
   5.4. $I_\alpha \leftarrow \{i \in \{1, \ldots, n_g\} : \ c + q \geq 0\}$;
   5.5. $g_\alpha \leftarrow (c_{I_\alpha} + q_{I_\alpha})'(c_{I_\alpha} + q_{I_\alpha}) + \frac{\lambda}{2}\xi_\alpha'\xi_\alpha$;
   5.6. **while** $g_\alpha > g + \alpha\sigma\nabla g'd$ **do**
      5.6.1. $\alpha \leftarrow \frac{1}{2}\alpha$; $\xi_\alpha \leftarrow \xi + \alpha d$;
      5.6.2. $c^\alpha \leftarrow c + \alpha q$;
      5.6.3. $I_\alpha \leftarrow \{i \in \{1, \ldots, n_g\} : \ c_i^\alpha \geq 0\}$;
      5.6.4. $g_\alpha \leftarrow (c_{I_\alpha}^\alpha)'c_{I_\alpha}^\alpha + \frac{\lambda}{2}\xi_\alpha'\xi_\alpha$;
   5.7. **end while**;
   5.8. $\xi \leftarrow \xi_\alpha$; $g \leftarrow g_\alpha$; $I \leftarrow I_\alpha$; $c \leftarrow c_\alpha$;
   5.9. $\nabla g \leftarrow \mathcal{A}_{I_\alpha}'c_{I_\alpha}^\alpha + \lambda\xi$; $\delta \leftarrow \zeta\|\nabla g\|$;
   5.10. $k \leftarrow k + 1$;
6. **retrieve** ω^i, γ^i, $i = 1, \ldots, s$, from the solution ξ;
7. **end**.

Output: Coefficients ω^i, γ^i, $i = 1, \ldots, s$ defining the piecewise affine separator ϕ in (2.3)–(2.4).

are known for all $i = 1, \ldots, s$, with

$$\delta(i,j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } i, j \in \{1, \ldots, s\}.$$

Each value π_i represents the relative "volume" of the i-th cluster where

$$\sum_{i=1}^{s} \pi_i = \int_{\mathcal{X}} \sum_{i=1}^{s} \delta(i, i(x)) p(x) dx = \int_{\mathcal{X}} p(x) dx = 1.$$

Problem (2.7)–(2.8) can be generalized to the unconstrained convex stochastic optimization problem given by

$$\begin{aligned} \xi^* &= \min_{\xi} E_x \left[\ell(x, \xi) \right] + \frac{\lambda}{2} \|\xi\|_2^2 \\ \ell(x, \xi) &= \sum_{\substack{j=1 \\ j \neq i(x)}}^{s} \frac{1}{\pi_{i(x)}} \left(x'(\omega^j - \omega^{i(x)}) - \gamma^j + \gamma^{i(x)} + 1 \right)_+^2, \end{aligned} \tag{2.14}$$

with $E_x[\cdot]$ denoting the expected value with respect to x. Problem (2.14) aims at finding ξ^* so to violate the least, on average over x, the conditions in (2.5) for $i = i(x)$.

The solution of problem (2.14) provides the PWA multi-category discrimination function satisfying (2.3)–(2.4). Problem (2.14) is solved using the averaged stochastic gradient descent (ASGD) method of [98] as proposed in [22] (cf. also [115]). The application of ASGD to the linear multi-category discrimination problem (2.14) is described in Algorithm 4, and it allows us to solve the problem of learning ϕ on-line, while the data-points x_k are acquired in real time, without the need of storing all past data-points $\{x(j)\}_{j=0}^{k-1}$.

The regularization hyper-parameter λ is supposed to be positive, so that the objective function in (2.14) is strongly convex[2]. Algorithm 4 requires the initialization for ξ and the choice of λ. If a batch of data is available to run Algorithm 3, the initial estimate ξ_0 can be chosen either as the result

[2] A differentiable function f with domain $\mathcal{D}$ is strongly convex if

$$\forall x, y \in \mathcal{D}, \exists m > 0 : f(y) \geq f(x) + \nabla f(x)^\top (y - x) + m \|x - y\|_2^2.$$

Algorithm 4 ASGD for linear multi-category discrimination.

Input: Regressor flow $x(0), x(1), \ldots$; cluster assignment function $i : \mathbb{R}^{n_x} \to \{1, \ldots, s\}$; ℓ_2-regularization hyper-parameter $\lambda > 0$; scalar $\nu_0 \geq 0$; initial guess $\xi \in \mathbb{R}^{n_\xi}$;

1. **for** $k = 0, 1, \ldots$ **do**:

   1.1. **compute** $\nabla_\xi \ell(\xi_k, x_k)$ as follows:

   1.1.1. $\varphi^j(k) \leftarrow x(k)'(\omega^j(k) - \omega^{i(x(k))}(k)) - \gamma^j(k) + \gamma^{i(x(k))}(k) + 1$;

   1.1.2. $I(k) \leftarrow \{j \in \{1, \ldots, s\}, j \neq i(x(k)) : \varphi^j(k) \geq 0\}$;

   1.1.3. $\psi^j(k) \leftarrow \begin{cases} \sum_{j \in I(k)} \varphi^j(k) \begin{bmatrix} -x(k) \\ 1 \end{bmatrix} & \text{if } j = i(x(k)) \\ \varphi^j(k) \begin{bmatrix} x(k) \\ -1 \end{bmatrix} & \text{if } j \neq i(x(k)),\ j \in I(k) \\ 0 \text{ otherwise} \end{cases}$

   1.1.4. **set**
   $$\frac{\partial \ell(\xi(k), x(k))}{\partial \begin{bmatrix} \omega^j \\ \gamma^j \end{bmatrix}} \leftarrow \lambda \begin{bmatrix} \omega^j(k) \\ \gamma^j(k) \end{bmatrix} + \frac{1}{\pi_{i(x(k))}} \times \psi^j(k);$$

   1.2. **compute**
   $$\begin{aligned} \nu(k) &\leftarrow \nu_0 (1 + \nu_0 \lambda k)^{-\frac{3}{4}}; \\ \mu(k) &\leftarrow 1/\max\{1, k - n_x, k - n_\xi\}; \\ \xi(k+1) &\leftarrow \xi(k) - \nu(k) \nabla_\xi \ell(\xi(k), x(k)); \\ \bar{\xi}(k+1) &\leftarrow \bar{\xi}(k) + \mu(k)(\xi(k+1) - \bar{\xi}(k)); \end{aligned}$$

   1.3. **retrieve** $\omega^i(k), \gamma^i(k)$, $i = 1, \ldots, s$, from $\bar{\xi}(k)$;

2. **end**.

Output: Coefficients $\{\omega^i(k), \gamma^i(k)\}_{i=1}^s$, defining the separator ϕ in (2.3)–(2.4) at each step $k = 0, 1, \ldots$.

of the off-line execution of Algorithm 3 or as zero (or any value in $\mathbb{R}^{n_\xi}$). Otherwise, it can be chosen as any value.

The coefficients π_i used in step 1.1.4 can be estimated from off-line data, namely $\pi_i = \frac{m_i}{N}$ and they can be further updated while Algorithm 4 is running. However, numerical experiments have shown that constant

and uniform coefficients $\pi = \frac{1}{s}$ work equally well.
Algorithm 4 can be used instead of Algorithm 3, when the computation of the partition has to be computed on-line. Alternatively, Algorithm 4 can be employed in addition to Algorithm 3 to refine ϕ on-line, based on streaming data.

2.3 Case studies

The approach for off-line PWA regression (the partition os computed using the Piecewise-smooth Newton approach summarized in Algorithm 3) is tested against two simulation examples and an experimental dataset. In the simulation examples, the output used for training is supposed to be corrupted by an additive zero-mean white noise with Gaussian distribution and the effect of the noise on the output is quantified through the Signal-to-Noise Ratio (SNR), defined for the i-th channel as

$$\mathrm{SNR}_i = 10 \log \frac{\sum_{k=1}^{N} \left(y_i(k) - e_{\mathrm{o},i}(k)\right)^2}{\sum_{k=1}^{N} e_{\mathrm{o},i}^2(k)}. \tag{2.15}$$

Both in the simulation experiments and the experimental test, the estimated PWA functions are validated on a sequence of data not used for training. The quality of the identified model is assessed through the *Best Fit Rate* (BFR) indicator

$$\mathrm{BFR} = 100 \cdot \max \left\{ 1 - \frac{\|y_\mathrm{o} - \hat{y}\|_2}{\|y_\mathrm{o} - \bar{y}_\mathrm{o}\|_2}, 0 \right\} \%, \tag{2.16}$$

with y_o and $\hat{y}$ denoting the vector stacking the measured and simulated outputs, respectively, and $\bar{y}_\mathrm{o}$ representing the vector stacking the sample mean of the measured output.

All computations are carried out on an i7 2.40-GHz Intel core processor with 4GB of RAM running MATLAB R2014b.

2.3.1 Identification of a mono-dimensional PWA map

Define $\tilde{x} = \begin{bmatrix} x & 1 \end{bmatrix}'$, with $x \in \mathbb{R}$. Let the data used for training be generated by the following (unknown) PWA function

$$f_{\mathrm{o}}(x) = \begin{cases} \begin{bmatrix} 1 & 0.2 \end{bmatrix} \tilde{x} & \text{if } x \in (-\infty, -1) \\ \begin{bmatrix} -1 & 2 \end{bmatrix} \tilde{x} & \text{if } x \in [-1, 2) \\ \begin{bmatrix} 1 & -1 \end{bmatrix} \tilde{x} & \text{if } x \in [2, 4) \\ \begin{bmatrix} 0 & 2 \end{bmatrix} \tilde{x} & \text{if } x \in [4, 6) \\ \begin{bmatrix} 2 & -10 \end{bmatrix} \tilde{x} & \text{if } x \in [6, +\infty), \end{cases} \tag{2.17}$$

already proposed as an example for PWA regression in the Hybrid Identification Toolbox (HIT) [43]. The map (2.17) is characterized by $s_{\mathrm{o}} = 5$ affine sub-models. The regressor $x(k) \in \mathbb{R}$ is a white noise sequence with uniform distribution in the interval $[-4,\ 8]$ and length $N = 1000$. The white noise sequence additively corrupting the output, *i.e.*,

$$y(k) = f_{\mathrm{o}}(x(k)) + e_{\mathrm{o}}(k),$$

is $e_{\mathrm{o}}(k) \sim \mathcal{N}(0, 0.01)$, corresponding to SNR equal to 27 dB.
We run Algorithm 2 with $s = s_{\mathrm{o}} = 5$, $\Lambda_e = 1$, no forgetting factor ($\kappa = 1$) and $\delta = 10^3$. The parameters $\{\theta_i\}_{i=1}^{s}$ are initialized in (2.2), while the initial guesses for the cluster centroids and covariance matrices are computed by running an instance of Algorithm 2 without the first term in (S1.1).
Algorithm 2 is then run again 5 times, with the full criterion (S1.1), initializing θ_i, c_i and R_i with the output of the previous run. The clusters are then separated solving problem (2.7) with the regularized Piecewise-smooth Newton method (RPSN) summarized in Algorithm 3, with parameters: $K = 300$, $\sigma = 0.4$, $\lambda = 10^{-5}$, $\zeta = 10^{-4}$, $g_{\mathrm{tol}} = \delta_{\mathrm{tol}} = 10^{-6}$ and $\xi_0 = 0$.

The quality of the estimated PWA map is assessed with respect to a noiseless validation dataset with $N_{\mathrm{V}} = 200$ samples. The obtained BFR is 97.05 %, with only 1 misclassified point out of 200 samples, *i.e.*, only the 0.5% of points in the validation set are misclassified. The total CPU

Table 1: Case study 1. True vs estimated θ_i, $i = 1, \dots, 5$

	θ_1		θ_2		θ_3		θ_4		θ_5	
True	1	0.2	-1	2	1	-1	0	2	2	-10
Algo 2–3	1.01	0.22	-0.99	2.00	1.00	-1.00	0.00	2.01	2.00	-9.97
Algo [45]	1.03	0.28	-0.95	1.98	1.20	-1.72	-0.02	2.10	2.00	-10.00

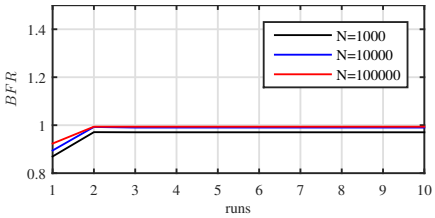


Figure 3: Case study 1. BFR vs M

time for solving the regression problem is 0.137 s, of which 0.006 s are taken to compute the polyhedral partition through Algorithm 3.

Comparison with the method presented in [45]

For comparison, the same regression problem is solved with the algorithm proposed in [45][3], using a linear *Support Vector Classifier* (SVC) [112] to compute the partition. The obtained BFR with the approach of [45] is 95.04 %, with the CPU time needed to solve the regression problem being around 63 s. The method of [45] has thus proven to be 459x slower than our approach (required CPU time around 0.14 s). To provide an additional element of comparison, in Table 1 we report the parameters estimated with the proposed approach and the one of [45]. In the considered case, our method has proven to perform better than the one presented in [45] both in terms of CPU time and of accuracy in estimating the parameters $\{\theta_i\}_{i=1}^{s}$.

Convergence properties

We expect that both the CPU time required to solve the regression problem and the accuracy of the estimated PWA map are influenced by the

[3]The Hybrid Identification Toolbox [43] has been used.

number M of runs of Algorithm 2 and the dimension N of the training set. The obtained BFR as a function of M is plotted in Figure 3 for different lengths of the training set. Algorithm 2 converges after 2 runs.

On the performance of the multi-category discrimination algorithm

To assess the performance of the proposed approaches for linear multi-category discrimination, four multi-category discrimination algorithms are used to generate the polyhedral partition of the domain $\mathcal{X}$:

1. *robust linear programming* (RLP) [17][4];
2. *regularized piecewise-smooth Newton* (RPSN) method (Algorithm 3), using $K = 300$, $\sigma = 0.4$, $\lambda = 10^{-5}$, $\zeta = 10^{-4}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$ and $\xi_0 = 0.$;
3. *averaged stochastic gradient descent* (ASGD) method (Algorithm 4). The weights π_i and ξ_0 computed executing Algorithm 3 on the first 50 training samples. Then, the remaining data-points are processed recursively, $\lambda = 10^{-5}$ and $\nu_0 = 0.01$;
4. *multi-category support vector machines* (MSVM) with linear kernels [68][5].

The CPU time required by the different methods to compute the polyhedral partition is reported in Table 2. The performance of the MSVM approach is evaluated only in relation to the smaller training set, as larger data sets take too long to be processed. It is worth noticing that, for N=100000, Algorithms 3 and 4 are about 234x and 1126x faster, respectively, than the robust linear programming method of [17].

Considering the accuracy of the PWA map obtained when the Voronoi diagram induced by the clusters' centroids is used as the final partition of the space $\mathcal{X}$, the BFRs for the different approaches are reported in Table 3.

The results in Table 3 show that the use of robust linear program-

[4]The solver *Gurobi* has been used to compute the solution of the multi-category discrimination problem.

[5]MSVM has been implemented using the MSVMpack 1.5 toolbox [67].

Table 2: Case study 1. CPU time [s] vs N.

	N= 10^3	N= 10^4	N= 10^5
RLP [17]	0.031 s	1.053 s	53.615 s
RPSN (Algorithm 3)	0.006 s	0.016 s	0.229 s
ASGD (Algorithm 4)	0.003 s	0.008 s	0.048 s
MSVM [68]	62.203 s	–	–

Table 3: Case study 1. BFR vs N.

	N= 10^3	N=10^4	N=10^5
RLP [17]	97.05 %	99.59 %	99.83 %
RPSN (Algorithm 3)	97.05 %	99.01 %	99.38 %
ASGD (Algorithm 4)	94.91 %	99.07 %	99.65 %
MSVM [68]	54.46 %	–	–
Voronoi	95.74 %	86.27 %	95.62 %

ming (RLP), the piecewise-smooth Newton method (RPSN) and average stochastic gradient descent (ASGD) lead to an accurate estimate of the true function in terms of output prediction, with BFRs larger then 95 %. Lower performance are achieved when MSVM is employed. Moreover, the accuracy of the estimated models increases with the number N of training samples when RPL, RPSN and ASGD are used. This is not true when the Voronoi diagram induced by the clusters' centroids is used as a partition. As a matter of fact, the Voronoi diagram only depends on the clusters' centroids and it does not account for how the points are spread around the centroids, thus causing the computed partition to be less accurate.

2.3.2 Learning a static three-dimensional nonlinear function

Suppose that the data are generated by the (unknown) function

$$f_o(x) = \begin{cases} h(x) & \text{if } -0.5 \leq h(x) \geq 0.5 \\ 0.5 & \text{if } h(x) \geq 0.5 \\ -0.5 & \text{if } h(x) \leq -0.5 \end{cases} \tag{2.18}$$

with $h : \mathbb{R}^3 \to \mathbb{R}$ and $h(x) = 0.6 \sin\left(x_1 + x_2^2 - x_3\right)$. The regressor $x(k) \in \mathbb{R}^3$ is a white noise sequence with uniform distribution in the box $[-1,\ 1]^3$

and length $N = 1250$. The output of the function f_o is corrupted by an additive zero-mean white noise $e_o(k)$, *i.e.*,

$$y(k) = f_o(x(k)) + e_o(k),$$

with $e_o \in \mathbb{R} \sim \mathcal{N}(0, 0.02^2)$. This yields to a SNR of 25 dB.

To learn the PWA function approximating f_o, Algorithm 2 is run choosing the same parameters used in the case study presented in Section 2.3.1, *i.e.*, $\Lambda_e = 1$, $\kappa = 1$ and $\delta = 10^3$. The parameters θ_i are initialized as in (2.2) and the initial guesses for the centroids and the covariance matrices are computed running a first instance of Algorithm 2, without the first term in (S1.1). Algorithm 2 is run 25 times, initializing θ_i, c_i and R_i with the output of the previous iteration. Then, to compute the polyhedral partition, problem (2.7) is solved via the Piecewise-smooth Newton method described in Algorithm 3, with parameters $K = 300$, $\lambda = 10^{-5}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$, $\sigma = 0.1$, $\zeta = 10^{-5}$ and an initial guess $\xi_0 = 0$.
Using a set of 250 samples neither used for training nor validation, we have performed a sensitivity analysis with respect to the parameters σ and ζ. As shown by the BFRs plotted in Figure 4 as functions of the tuning parameters ζ and σ, the method summarized in Algorithm 3 is basically insensitive to σ and ζ. An additional sensitivity analysis is performed with respect to the ℓ_2 hyper-parameter λ, using the same 250 calibration data points. The obtained BFR as a function of λ is reported in Figure 5, which shows that, for $\lambda \leq 0.1$, the final estimate is fairly insensitive to the choice of λ.
The number s is not known a priori and it is thus chosen by means of cross validation. Specifically, using a calibration set of 250 samples not used for training, a PWA function approximating f_o is reconstructed for different values of s. The number of affine sub-models is then chosen as the one providing the highest BFR. This is achieved for $s = 12$.

The quality of the estimated PWA function is assessed w.r.t. a validation dataset of $N_V = 200$ noiseless samples. The obtained BFR is 85.19 % and the total CPU time for solving the regression problem is 3 s, of which 0.257 s are taken to compute the polyhedral partition.

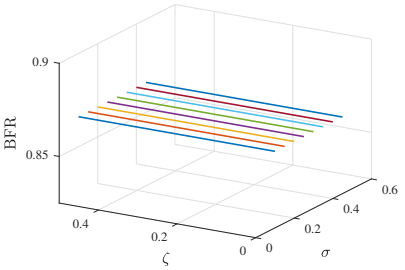


Figure 4: Case study 2. BFR vs σ and ζ.

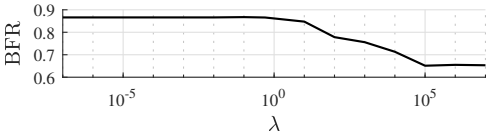


Figure 5: Case study 2. Algorithm 3: BFR vs λ

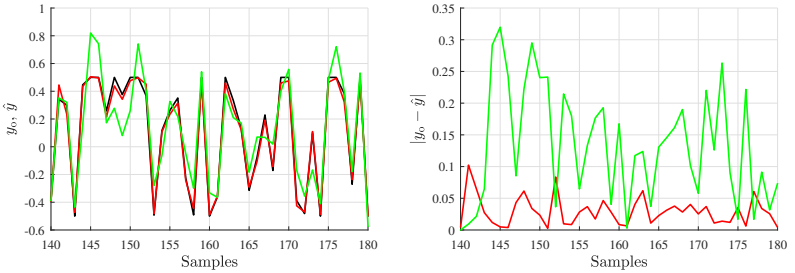
Comparison with the method presented in [45]

For the sake of comparison, the same regression problem is solved with the regression method presented in [45][6]. A Proximal Support Vector Classifier (PSVC) [48] is used to compute the partition[7]. The obtained BFR with [45] is 53.40 %, while BFR $= 85.19\%$ with the proposed approach. The CPU time needed to solve the regression problem with the approach of [45] is around 149 s (*i.e.*, 49x slower than the piecewise-smooth Newton method). To provide another element of comparison between the method proposed in this chapter and the one of [45], the outputs of the functions estimated with the two methods are plotted in Figure 6 against the true output y_o, along with the absolute value of the error $y_o(k) - \hat{y}(k)$. For a better visualization, only the samples in from 140 to 180 are reported.

Convergence properties

The quality of the estimated PWA map is also evaluated with respect to the number M of iterations of Algorithm 2 and the dimension N of

[6] We have used the Hybrid Identification Toolbox [43]
[7] Among the available classifier in HIT, PSVC is the one providing the best results.



(a) y_o vs $\hat{y}$ (b) $|y_o - \hat{y}|$

Figure 6: Case study 2. True vs simulated output. Black: true, red: Algo 2+3, green: Algo [45]+[48]

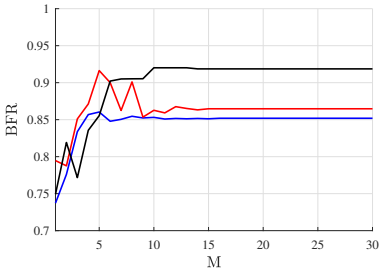


Figure 7: Case study 2. BFR vs M and N. Blue: N=1250, red: N=12500, black: N=125000

the training set. The obtained BFR as a function of the iterations M is reported in Figure 7 for different lengths of the training set. The BFR converges after 15 runs.

On the performance of the multi-category discrimination algorithm

To evaluate the performance of the proposed methods for linear multi-category discrimination, *i.e.*, the piecewise-smooth Newton method (see Algorithm 3) and average stochastic gradient descent (see Algorithm 4), we compute the polyhedral partition of space $\mathcal{X}$ with four methods: robust linear programming [17], RPSN, ASGD and MSVM [68]. The parameters used with RPSN are: $K = 300$, $\lambda = 10^{-5}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$, $\sigma = 0.1$, $\zeta = 10^{-5}$ and an initial guess $\xi_0 = 0$.
When average stochastic gradient descent is tested, the weights π_i and

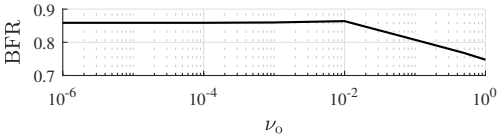


Figure 8: Case study 2. BFR vs ν_o.

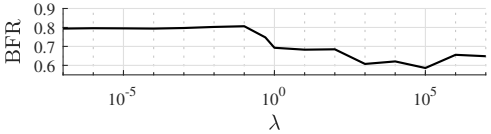


Figure 9: Case study 2. Algorithm 4: BFR vs λ.

the initial estimate ξ_0 are computed executing the batch Algorithm 3 on the first 50 samples, with the remaining training points processes recursively. Furthermore, we perform a sensitivity analysis with respect to the tunable parameters ν_o and λ, using a set of 250 samples neither used for training nor validation. Figures 8-9 report the obtained BFR as a function of the tunable parameters ν_o and λ, respectively. The performance of Algorithm 4 deteriorates when ν_o increases, following the typical behavior of stochastic gradient descend methods where convergence to the global optimum improves as the parameter ν_o decreases, at the price of a lower convergence speed. Instead, Algorithm 4 seems fairly insensitive to the choice of λ, if $\lambda \leq 0.1$. We thus choose $\lambda = 10^{-5}$ and $\nu_0 = 0.01$.

The CPU time required to compute the partition using the four approaches is given in Table 4. The performance of MSVM is evaluated only for the small/medium size training sets, as large datasets take too long to be processed. For a large training set ($N = 125000$), Algorithm 3 and Algorithm 4 are about 454x and 65200x faster, respectively, than the robust linear programming method of [17]. The obtained BFRs are reported in Table 5, along with the BFR obtained when the Voronoi diagram induced by the clusters' centroids is used to compute the partition of space $\mathcal{X}$. The results in Table 5 show that RPSN and ASGD lead to an accurate estimate of the true function in terms of output prediction, with BFRs larger than 80 % also in the case of small training set ($N = 1250$). This indicates that the training samples are accurately clustered at Stage S1. Robust lin-

Table 4: Case study 2. CPU time [s] vs N

	$N = 1250$	$N = 12500$	$N = 125000$
RLP [17]	1.336 s	125 s	8541 s
RPSN (Algorithm 3)	0.257 s	1.762 s	18.8 s
ASGD (Algorithm 4)	0.0014 s	0.018 s	0.131 s
MSVM [68]	6.545 s	3870 s	–

Table 5: Case study 2. BFR vs N

	$N = 1250$	$N = 12500$	$N = 125000$
RLP [17]	86.56 %	87.41 %	93.88 %
RPSN (Algorithm 3)	85.19 %	86.47 %	91.86 %
ASGD (Algorithm 4)	84.78 %	82.30 %	92.99 %
MSVM [68]	80.91 %	80.02 %	–
Voronoi	81.75 %	83.96 %	86.60 %

ear programming (RLP), the piecewise-smooth Newton method (RPSN) and average stochastic gradient descent (ASGD) lead to BFRs larger than 90 % for a large training set ($N = 125000$), while the Voronoi diagram does not achieve similar performance. This suggests that the Voronoi diagram is not flexible enough in partitioning the domain $\mathcal{X}$. As already remarked, the Voronoi diagram only depends on the clusters' centroids, and it does not take into account how the points are spread around the centroids.

Monte Carlo simulation

A Monte Carlo simulation with 100 runs, with new realizations of both the input u and the measurement noise e_{o} is used at each run, is carried out. Considering a training set of length $N = 12500$ and validating the results on a set of $N_{\mathrm{V}} = 200$ noiseless samples, this test allows us to assess the robustness of the estimation algorithm with respect to different realizations of the training data. The mean and the standard deviation of the BFR over the Monte Carlo simulations are reported in Table 6 for both the off-line (see Algorithm 3) and the on-line (see Algorithm 4) proposed to compute the multi-category linear separator.

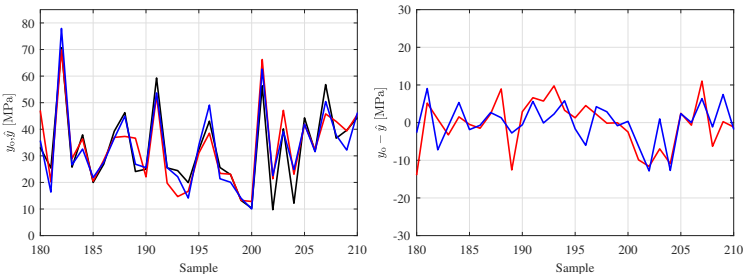
Table 6: Case study 2. Monte Carlo simulation, BFR (mean $\pm$ std).

	RPSN (Algorithm 2)	ASGD (Algorithm 3)
BFR	(87.01 ± 3.07) %	(86.88 ± 2.59) %

2.3.3 Modelling of concrete strength

Our approach for PWA regression is used to estimate from experimental data a function modeling the concrete compressive strength as a function of age, super-plasticizer, fine aggregate content and cement-to-water ratio. The dataset used for this test is taken from the UCI machine learning database repository [71] and it consists of 1030 records gathered from laboratory tests (see [116] for details). After shuffling he 1030 available data, of the 1030 available samples, 50 data-points are used to tune the number of affine sub-models through cross-validation and 730 samples are used for training. The remaining 250 samples are used for validation. The regressor/output pairs are clustered through Algorithm 2 and the polyhedral partition is computed using Algorithm 3, with $K = 300$, $\lambda = 10^{-5}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$, $\sigma = 0.1$, $\zeta = 10^{-5}$ and an initial guess $\xi_0 = 0$. The number of local models is chosen in cross-validation and it is $s = 9$.

We compare the results obtained using Algorithm 2-3 with the model for concrete compressive strength retrieved with a *Neural Network* (NN) with



(a) y_o vs $\hat{y}$ (b) $y_o - \hat{y}$

Figure 10: Case study 3. Black: true, blue: PWA regression, red: Neural Network

a single hidden layer of size 15[8]. Hyperbolic tangent sigmoid activation functions are used.
The actual concrete compressive strength is plotted in Fig. 10, along with the output of the estimated PWA map and the neural network. For the sake of visualization, only 30 out of 250 samples are plotted. The BFR achieved in validation with the PWA function is 52.4 %, while a BFR equal to 56.0 % is obtained with the NN. The resulting BFRs show that the NN achieves a slightly better accuracy than the PWA function, but this comes at the price of a more complex structure.

2.4 Identification of PWARX and LPV systems

In this Section, we show how the PWA regression approach presented in Section 2.2 can be used to identify PWA dynamical systems (specifically, PWA systems with exogenous inputs in the autoregressive form) and Linear Parameter Varying (LPV) systems. When PWA autoregressive systems with exogenous inputs (PWARX) have to be identified, the addressed problem can naturally be recast as a PWA regression problem. Instead, LPV systems are identified approximating their p-dependent coefficients through PWA functions of the scheduling variable p estimated from data.

Identification of PWARX systems

In discrete time, a Piecewise Autoregressive system with exogenous inputs (PWARX) is a Multi-Input Multi-Output (MIMO) dynamical system. The output $y(k) \in \mathbb{R}^{n_y}$ of a PWARX system at sampling time $k \in \mathbb{N}$

[8]The size of the hidden layer tuned through cross-validation.

is given by the following PWA function

$$y(k) = \begin{cases} \theta_1 \begin{bmatrix} 1 \\ x(k) \end{bmatrix} & \text{if } x(k) \in \mathcal{X}_1, \\ \vdots & \\ \theta_s \begin{bmatrix} 1 \\ x(k) \end{bmatrix} & \text{if } x(k) \in \mathcal{X}_s. \end{cases} \tag{2.19}$$

The regressor $x(k) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ is a collection of the past input and output observations, *i.e.*,

$$x(k) = \left[y'(k-1)\ y'(k-2)\ \cdots\ y'(k-n_a)\ u'(k-1)\ u'(k-2)\ \cdots u'(k-n_b) \right], \tag{2.20}$$

with $u(k) \in \mathbb{R}^{n_u}$ denoting the measured exogenous input at time k.

Identification of LPV systems

Linear Parameter Varying (LPV) systems can be seen as an extension of Linear Time-Invariant (LTI) systems, as their dynamic input/output relation is linear, but it changes over time accordingly to the measurable time-varying signal $p(t) \in \mathbb{R}^{n_p}$, the so-called *scheduling variable*. LPV models can thus accurately describe the dynamic behavior of a large class of nonlinear and time-varying systems, while preserving a linear relation between the input and the output signals. Practical use of LPV models is stimulated by the fact that the theory of LPV control is well established, mainly based on the extensions of the results of optimal and robust LTI control theory [3, 85, 101] and its direct use in Model Predictive Control (MPC) [16]. Consider the following MIMO LPV-ARX, modeling the input/output relation of the system to be identified:

$$y(k) = \bar{a}_0(p(k)) + \sum_{j=1}^{n_a} \bar{a}_j(p(k)) y(k-j) + \sum_{j=1}^{n_b} \bar{a}_{j+n_a}(p(k)) u(k-j), \tag{2.21}$$

where $p(k) \in \mathcal{P} \subseteq \mathbb{R}^{n_p}$ is the value of the scheduling variable at time k. The coefficients of the model (2.21), *i.e.*, $\bar{a}_j(p(k))$, $j = 0, \ldots, n_a + n_b$,

are unknown functions of $p(k)$. We approximate the coefficients $\bar{a}_j(p(k))$ through the following PWA functions of p:

$$a_j(p(k)) = \begin{cases} A_1^j(p(k)) & \text{if } p(k) \in \mathcal{P}_1, \\ \vdots & \\ A_s^j(p(k)) & \text{if } p(k) \in \mathcal{P}_s, \end{cases} \tag{2.22}$$

where the dependence of each entry matrix $A_i^j(p(k))$ on $p(k)$ is affine and the scheduling variable space $\mathcal{P}$ is partitioned (instead of space $\mathcal{X}$). The LPV identification problem thus reduces to reconstructing the PWA mapping of the p-dependent coefficients $\bar{a}_j(p(k))$, which is defined over a polyhedral partition $\{\mathcal{P}_i\}_{i=1}^s$ of $\mathcal{P}$ estimated from data. Specifically, the N-length sequence of observations $\{x(k), y(k)\}_{k=1}^N$ is used, with $x(k)$ given by

$$x(k) = \begin{bmatrix} y'(k-1) & \dots & y'(k-n_a) & u'(k-1) & \dots u'(k-n_b) \end{bmatrix}' \otimes \begin{bmatrix} 1 \; p'(k) \end{bmatrix}'.$$

It is worth to further remark that the polyhedral partition $\{\mathcal{P}_i\}_{i=1}^s$ of the scheduling variable space $\mathcal{P}$ is not fixed a priori. The underlying dependencies of $a_j(p(k))$ on the scheduling variable $p(k)$ (see (2.22)) are thus reconstructed from data. This represents one of the main advantages with respect to widely used parametric LPV identification approaches (see [6, 111]), which in turn require to parametrize $a_j(p(k))$ as a linear combination of some a-priori specified basis function (*e.g.*, polynomial or trigonometric functions). Indeed, the choice of the number and type of basis functions is a critical issue in the identification procedure, as an inaccurate selection of the set of basis functions could lead to a structural bias. To capture the underlying dependence of the coefficients $a_j(p(k))$ on $p(k)$, a large set of basis function is often used, with consequent numerical problems in the implementation of the identification schemes and the increasing possibility of overfitting the training data.

Remark 1 *If the entries of the sub-model matrices $A_i^j(p(k))$ ($i = 1, \dots, s$, $j = 0, \dots, n_a + n_b$) do not depend on $p(k)$, i.e., $a_j(p(k))$ are approximated by piecewise constant functions, the LPV model becomes a switched linear model, where each sub-system is LTI and the switching behavior depends on the scheduling signal $p(k)$.* ■

2.4.1 Identification of a MIMO PWARX system

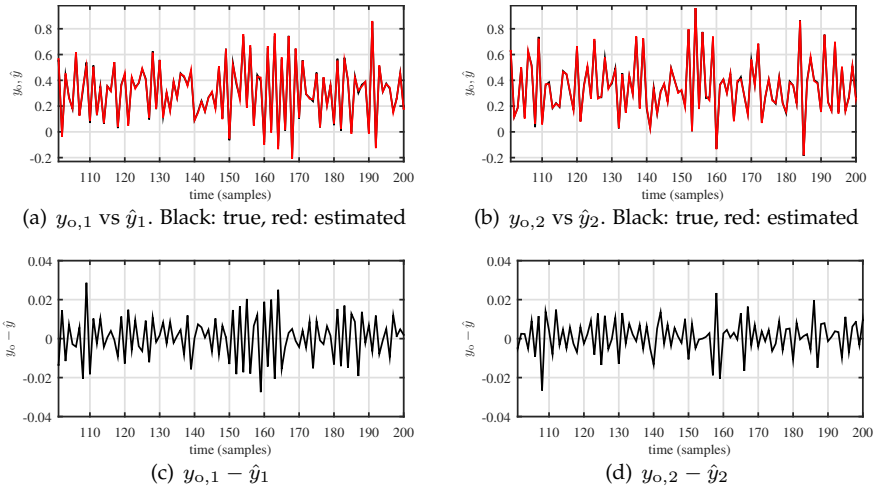
Let the data be generated by the MIMO PWARX described by the difference equation

$$\begin{aligned}\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = &\begin{bmatrix} -0.83 & 0.20 \\ 0.30 & -0.52 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \begin{bmatrix} -0.34 & 0.45 \\ -0.30 & 0.24 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix} + \begin{bmatrix} 0.20 \\ 0.15 \end{bmatrix} + \\ &+ \max \left\{ \begin{bmatrix} 0.20 & -0.90 \\ 0.10 & -0.42 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \begin{bmatrix} 0.42 & 0.20 \\ 0.50 & 0.64 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix} + \right. \\ &\left. + \begin{bmatrix} 0.40 \\ 0.30 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} + e_{\mathrm{o}}(k) \end{aligned} \tag{2.23}$$

System (2.23) is characterized by $\bar{s} = 4$ operating modes, given by the possible combinations of sign of the components of the first vector argument of the "max" operator. The excitation input $u(k)$ is a white noise sequence with uniform distribution in the box $[-1.0,\ -0.4] \times [-1.0,\ -0.4]$ and length $N = 4000$. The output in (2.23) is supposed to be corrupted by an additive noise sequence $e_{\mathrm{o}} \sim \mathcal{N}(0, \Lambda_e)$, with $\Lambda_e = \left[\begin{smallmatrix} 0.02 & 0.02 \\ 0.02 & 0.02 \end{smallmatrix}\right]$ corresponding to SNRs equal to $\mathrm{SNR}_1 = 8.7$ dB and $\mathrm{SNR}_2 = 6.9$ dB on the first and second output channels, respectively.

Algorithm 2 is run with $s = \bar{s} = 4$. Using a training set of length $N = 4000$, the initial guess for the parameters $\theta_1, \ldots, \theta_s$ is obtained as in (2.2), while, as for the other case studies, the clusters' centroids and covariance matrices are initialized running an instance of Algorithm 2 without the first term in (S1.1).
Algorithm 2 is then run for a total of 15 times with the full criterion (S1.1) and, finally, the clusters obtained running Algorithm 2 are separated using the approach for batch linear multi-category discrimination (see Algorithm 3), with $K = 300$, $\sigma = 0.4$, $\lambda = 10^{-5}$, $\zeta = 10^{-4}$, $g_{\mathrm{tol}} = \delta_{\mathrm{tol}} = 10^{-6}$ and the parameters ξ_0 set to zero.

The quality of the estimated PWA model is assessed with respect to a validation dataset of $N_{\mathrm{V}} = 500$ noiseless samples. The true and the open-loop simulated output sequences are shown in Figure 11, along with the simulation error $y_{\mathrm{o}-\hat{y}}$. For the sake of visualization, we show y_{o}, $\hat{y}$ and



(a) $y_{o,1}$ vs $\hat{y}_1$. Black: true, red: estimated

(b) $y_{o,2}$ vs $\hat{y}_2$. Black: true, red: estimated

(c) $y_{o,1} - \hat{y}_1$

(d) $y_{o,2} - \hat{y}_2$

Figure 11: PWARX identification. True vs simulated output

$y_{o-\hat{y}}$ for a window of 100 samples. Accordingly to the simulated output in Figure 11, the BFRs are 96.2% and 96.3%, for the first and the second output channel, respectively. The estimated polyhedral partition of the regressor space $\mathcal{X}$ is such that only 12 out of 500 samples (i.e., the 2.4% of the whole validation set) are misclassified. The CPU time required to identify the complete PWARX model is 0.76 s, of which 0.016 s are taken to execute Algorithm 3.

Convergence properties

As already remarked, the accuracy of the estimated final model and the total CPU time are influenced by the number of runs of Algorithm 2 and the number of samples N available for training. The performance of the used approach (Algorithm 3 is used to compute the partition) has been tested increasing the number of runs M and the dimension of the training set N. Figure 12 shows the obtained BFR as a function of M for different values of N. Clearly, there is no improvement in BFR on the two output channels after about 8 runs.

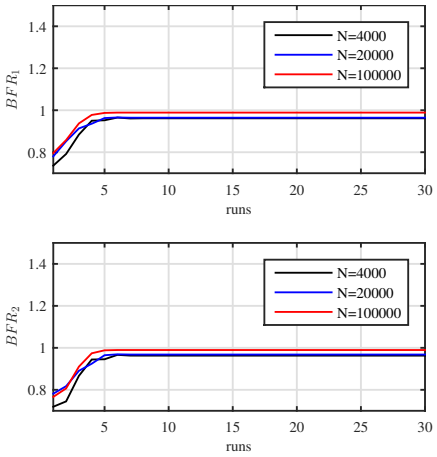


Figure 12: PWARX identification. BFR vs M and N

On the performance of the multi-category discrimination algorithm

The regularized Piecewise-smooth Newton (RPSN) method summarized in Algorithm 3, is compared with: robust linear programming (RLP) [17], the method proposed for the on-line update of the partition (average stochastic gradient descent, see Algorithm 4). When the piecewise-smooth Newton method is employed to compute the polyhedral partition of $\mathcal{X}$, we choose $K = 300$, $\sigma = 0.4$, $\lambda = 10^{-5}$, $\zeta = 10^{-4}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$ and the parameters ξ_0 are set to zero. If average stochastic gradient descent (ASGD) is used, the weights π_i and the initial estimate ξ_0 are computed by executing the batch Algorithm 3 on the first 3000 training samples, $\lambda = 10^{-5}$ and $\nu_0 = 10^{-3}$. The remaining training samples are processed recursively.

The estimated models are validated on a set of 500 samples not used for training, equal to the dataset used to evaluate the performance of the approach and the convergence of Algorithm 2. The obtained BFRs on the two outputs are shown in Table 7, while the percentages of misclassified data points for each method and for different values of N are reported in Table 8. Results in Tables 3–8 show that all the three discrimination algorithms used to compute the polyhedral partition of $\mathcal{X}$ lead to an accurate

Table 7: PWARX identification. BFR vs N

		$N = 4000$	$N = 20000$	$N = 100000$
BFR_1	RLP [17]	96.0 %	96.5 %	99.0 %
	RPSN	96.2 %	96.4 %	98.9 %
	ASGD	86.7 %	95.0 %	96.7 %
BFR_2	RLP	96.2 %	96.9 %	99.0 %
	RPSN	96.3 %	96.8 %	99.0 %
	ASGD	87.4 %	95.2 %	96.4 %

Table 8: PWARX identification. Percentage of misclassified points

	$N = 4000$	$N = 20000$	$N = 100000$
RLP [17]	1.6 %	1.0 %	0.2 %
RPSN	2.4 %	1.2 %	0.4 %
ASGD	5.4 %	2.0 %	1.4 %

Table 9: PWARX identification. CPU time vs N

	$N = 4000$	$N = 20000$	$N = 100000$
RLP [17]	0.308 s	3.227 s	112.435 s
RPSN	0.016 s	0.086 s	0.365 s
ASGD	0.013 s	0.023 s	0.067 s

estimate of the underlying PWA system, both in terms of output prediction (Table 3) and partition of the domain $\mathcal{X}$. Specifically, we obtain BFRs larger then 95 % and a percentage of misclassified points smaller than 2.5 % (except when $N = 4000$ and Algorithm 4 is used). As expected, the accuracy of the models increases with the number of training samples. The CPU time required by the three discrimination algorithms is reported in Table 9. It is worth noting that, for a large training set (*i.e.*, N=100000), Algorithms 3 and 4 are about 300x and 1600x faster, respectively, than the robust linear programming method presented in [17].

On-line learning of a PWARX model

The performance of the proposed approach for PWA regression is further assessed when a PWARX model have to be reconstructed in real-time, while the data are acquired. Consider a set of N= 100000 samples. Among the available data-points, 10000 samples are supposed to

be generated by the PWARX system defined in (2.23), while the remaining 90000 samples are assumed to be generated by the following PWARX system:

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} -0.8579 & 0.1672 \\ 0.2076 & -0.4962 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \begin{bmatrix} -0.2958 & 0.4612 \\ -0.3085 & 0.2807 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix} + \\ + \begin{bmatrix} 0.2102 \\ 0.1323 \end{bmatrix} + \max \left\{ \begin{bmatrix} 0.1884 & -0.9455 \\ 0.0808 & -0.3677 \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \right. \\ \left. + \begin{bmatrix} 0.4486 & 0.1172 \\ 0.4093 & 0.5946 \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix} + \begin{bmatrix} 0.4272 \\ 0.2787 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\} + e_{\mathrm{o}}(k). \quad (2.24)$$

The identification of the PWARX system is performed processing the first 3000 samples, which are all generated by the PWARX in (2.23), in a batch mode. Algorithm 3 is thus used to compute an initial estimate of the polyhedral partition of $\mathcal{X}$. The remaining 97000 data points are processed on-line, with the polyhedral partition of the regressor space $\mathcal{X}$ computed through Algorithm 4.

To assess the performance of the approach we consider the true (noise-free) output $y_{\mathrm{o}}(k)$ and the one-step ahead prediction of the model output $\hat{y}(k)$. The Relative Mean Square Error (ReMSE) is used as performance index, by providing the ratio between the mean square error and the power of the actual output signal, *i.e.*,

$$\mathrm{ReMSE}_i = \frac{\sum_{k=1}^{N} \left(y_{\mathrm{o},i}(k) - \hat{y}_i(k) \right)^2}{\sum_{k=1}^{N} y_{\mathrm{o},i}^2(k)}. \quad (2.25)$$

The obtained values of ReMSE_1 and ReMSE_2 are 0.8 % and 0.1 %, respectively.

Concerning the computational complexity of the method, the average CPU time required to cluster the observed regressor and update the model parameters (Algorithm 2) at each time instant is around 324 μs, while Algorithm 4 requires 34 μs (on average) to update the partition of space $\mathcal{X}$. Based on these results, the proposed method for PWA regression seems to be suited for applications with sampling times down to the order of milliseconds (*e.g.*, adaptive control applications).

2.4.2 Identification of a MIMO LPV system

Let the data-generating system be the following (unknown) MIMO Linear Parameter Varying (LPV) system:

$$\begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} \bar{a}_{1,1}(p(k)) & \bar{a}_{1,2}(p(k)) \\ \bar{a}_{2,1}(p(k)) & \bar{a}_{2,2}(p(k)) \end{bmatrix} \begin{bmatrix} y_1(k-1) \\ y_2(k-1) \end{bmatrix} + \begin{bmatrix} \bar{b}_{1,1}(p(k)) & \bar{b}_{1,2}(p(k)) \\ \bar{b}_{2,1}(p(k)) & \bar{b}_{2,2}(p(k)) \end{bmatrix} \begin{bmatrix} u_1(k-1) \\ u_2(k-1) \end{bmatrix},$$

where

$$\begin{aligned}
\bar{a}_{1,1}(p(k)) =& \begin{cases} -0.3 & \text{if } 0.4\,(p_1(k)+p_2(k)) \leq -0.3 \\ 0.3 & \text{if } 0.4\,(p_1(k)+p_2(k)) \geq 0.3 \\ 0.4\,(p_1(k)+p_2(k)) & \text{otherwise} \end{cases} \\
\bar{a}_{1,2}(p(k)) =& 0.5\,(|p_1(k)|+|p_2(k)|)\,, \\
\bar{a}_{2,1}(p(k)) =& p_1(k)-p_2(k), \\
\bar{a}_{2,2}(p(k)) =& \begin{cases} 0.5 & \text{if } p_1(k) < 0 \\ 0 & \text{if } p_1(k) = 0 \\ -0.5 & \text{if } p_1(k) > 0 \end{cases} \\
\bar{b}_{1,1}(p(k)) =& 3p_1(k)+p_2(k), \\
\bar{b}_{1,2}(p(k)) =& \begin{cases} 0.5 & \text{if } 2\left(p_1^2(k)+p_2^2(k)\right) \geq 0.5 \\ 2\left(p_1^2(k)+p_2^2(k)\right) & \text{otherwise} \end{cases} \\
\bar{b}_{2,1}(p(k)) =& 2\sin\left\{p_1(k)-p_2(k)\right\}, \\
\bar{b}_{2,2}(p(k)) =& 0
\end{aligned}$$

The input $u(k)$ and the scheduling variable $p(k)$ are independent white noise sequences of length $N = 11000$ with uniform distribution in the boxes $[-0.5,\ 0.5] \times [-0.5,\ 0.5]$ and $\mathcal{P} = [-1,\ 1] \times [-1,\ 1]$, respectively, with $\mathcal{P}$ indicating the scheduling variable space. The noise covariance matrix of $e_{\mathrm{o}}(k) \in \mathbb{R}^2$ is $\Lambda_e = \left[\begin{smallmatrix} 0.25 & 0 \\ 0 & 0.25 \end{smallmatrix}\right]$, which yields to SNRs equal to $\mathrm{SNR}_1 = 4$ dB and $\mathrm{SNR}_2 = 7$ dB on the first and the second output channel, respectively.

Choice of the number of modes

The number s of affine sub-models (*i.e.*, the number of polyhedral regions defining the partitions of$\mathcal{P}$) is chosen by means of cross validation. The 11000-length training data set is split into two disjoint sets, where

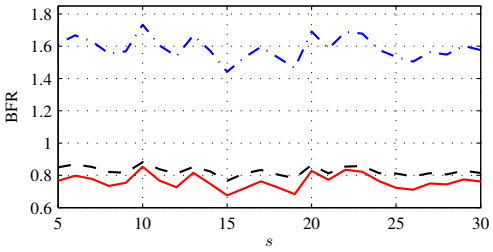


Figure 13: LPV identification. Black dashed line: BFR_1, red solid line: BFR_2, blue dashed-dot line: BFR_T

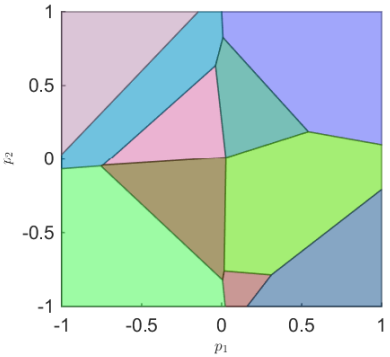
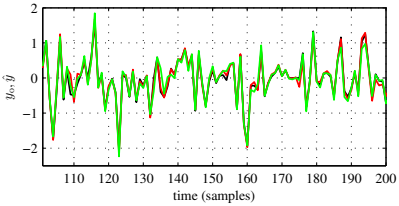


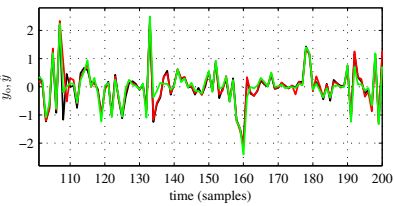
Figure 14: LPV identification. Polyhedral partition of the scheduling variable space $\mathcal{P}$ with $s = 10$

the first 10000 samples are used for calibration purposes. Specifically, the PWA regression problem is solved for different values of s in the range 5–30. For each value of s Algorithm 2 is iterated 10 times. The remaining 1000 samples are used to assess the quality of the identified LPV models. Specifically, for each value of s, we compute the BFR on the two output channels. The obtained BFR_i (with $i = 1, 2$), as a function of s, along with the aggregate BFR, i.e., $\mathrm{BFR}_\mathrm{T} = \mathrm{BFR}_1 + \mathrm{BFR}_2$, are reported in Figure 13. Among the identified LPV models, the one providing the largest aggregated $\mathrm{BFR}_\mathrm{T} = \mathrm{BFR}_1 + \mathrm{BFR}_2$ is selected, which corresponds to $s = 10$. The computed polyhedral partition is plotted in Figure 14[9].

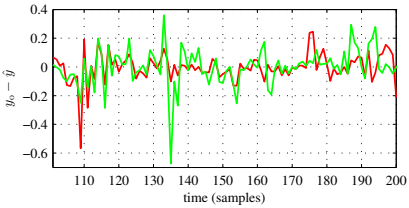
[9]The Hybrid Toolbox for MATLAB [9] is used to compute the partition.



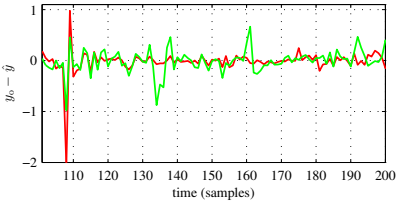
(a) $y_{\mathrm{o},1}$ vs $\hat{y}_1$. Black: true, red: PWA regression, green: [6]



(b) $y_{\mathrm{o},2}$ vs $\hat{y}_2$. Black: true, red: PWA regression, green: [6]



(c) $y_{\mathrm{o},1} - \hat{y}_1$. Red: PWA regression, green: [6]



(d) $y_{\mathrm{o},2} - \hat{y}_2$. Red: PWA regression, green: [6]

Figure 15: LPV identification. True vs estimated output

Model quality assessment

The quality of the estimated model is assessed with respect to a validation dataset, consisting of a sequence of $N_{\mathrm{V}} = 2000$ noiseless samples not used for calibration nor training. For the sake of comparison, the nonlinear coefficient functions $\bar{a}_{i,j}(p(k))$ and $\bar{b}_{i,j}(p(k))$ at k are also estimated through the parametric LPV identification approach in [6], approximating the nonlinear coefficients $\bar{a}_{i,j}(p(k))$ and $\bar{b}_{i,j}(p(k))$ as fourth-order polynomials in the two-dimensional scheduling variable $p(k)$. Figure 15 reports the true and the simulated sequence, along with the estimation error obtained with both the approaches. For the sake of visualization, only the samples in the interval $[100,\ 200]$ are reported. The proposed PWA regression method allows us to accurately reconstruct the output. To provide an additional mean of comparison, the BFRs obtained with the two considered approaches for LPV identification are reported in Table 10. The obtained results show that the proposed PWA-LPV identification approach, based on the PWA approximation of the coefficient functions $\bar{a}_{i,j}(p(k))$ and $\bar{b}_{i,j}(p(k))$, outperforms the paramet-

Table 10: LPV identification. BFR vs approach

	BFR_1	BFR_2
PWA regression (Algo 2+3)	87 %	84 %
parametric LPV [6]	80 %	70 %

ric LPV identification approach in [6], with the method in [6] based on the parametrization of the nonlinear functions $\bar{a}_{i,j}(p(k))$ and $\bar{b}_{i,j}(p(k))$. This results from the flexibility of the PWA model, *i.e.*, the fact that the polyhedral partition is computed from data and not fixed a priori, while the method presented in [6] is based on a priori specified polynomial parametrization of the nonlinear functions $\bar{a}_{i,j}(p(k))$ and $\bar{b}_{i,j}(p(k))$.
Concerning the computational time required to evaluate the output of the LPV model, given the current value of p and the past input/output observations, is about 120 μs, 40 μs of which required to assign p to a polyhedral region. Note that the assignment is performed computing the maximum of the $s = 10$ affine functions $\{\phi_i(p)\}_{i=1}^{s}$ defining the PWA separator $\phi(p)$ in (2.3).
The relatively "low" computational time required to reconstruct the output is mainly due to structure of the estimated model, which can thus be used in applications requiring a "fast" on-line determination of the operating mode (*e.g.*, gain scheduling or LPV Model Predictive Control).

On the performance of the multi-category discrimination algorithms

As in the previous case studies, the performance of different multi-category discrimination method are assessed. In particular, robust linear programming [17], the Newton-like approach (Algorithm 3) and the averaged stochastic gradient descent method (Algorithm 4) are compared. When computing the partition with Algorithm 3, the chosen parameters are: $K = 300$, $\sigma = 0.4$, $\lambda = 10^{-5}$, $\zeta = 10^{-4}$, $g_{\text{tol}} = \delta_{\text{tol}} = 10^{-6}$, initial guess for ξ_{o} set to zero. In running Algorithm 4, λ and ν_0 are set to 10^{-5} and 10^{-3}, respectively. The weights π_i and the initial estimate ξ_0 for Algorithm 4 are computed by executing the batch Algorithm 3 on the first 1000 training samples. The remaining 9000 training samples are

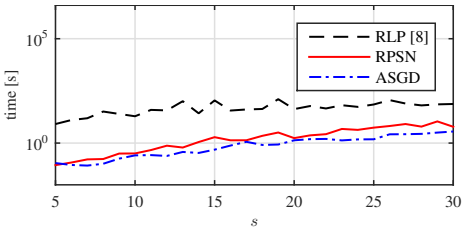


Figure 16: LPV identification. CPU time vs s

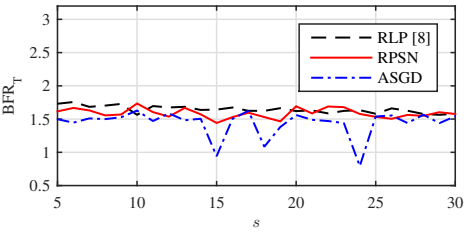


Figure 17: LPV identification. BFR_T vs s

processed recursively.

The CPU time required to compute the partition is reported in Figure 16, as a function of the number of modes s. Note that the CPU time reported in Fig. 16 also includes the time required to initialize Algorithm 4. In particular, for $s = 10$, the estimate of the LPV model is computed in 14 s, 0.4 s of which required to compute the partition, using algorithm 2 and Algorithm 3. On the other hand, the RLP method proposed in [17] takes 4.2 s, *i.e.*, it is almost 10x slower than the proposed extension of the PWA regression method.
The performance are further assessed considering the aggregate BFR, reported in Figure 17.

The results reported in Figure 16, 17 show that the CPU time required to compute the partition of space $\mathcal{P}$ increase with s, independently from the used method. This behavior can be explained considering that the number of parameters ξ defining the PWA separator ϕ increases linearly with s. Furthermore, the piecewise-smooth Newton method and average stochastic gradient descent are faster (from 6x to 20x) than robust linear

programming [17]. On the one hand, RLP and RPSN achieve similar performance in term of BFR. See Figure 17. On the other hand, for $s = 11$, $s = 14$ and $s = 20$, ASGD does not provide an accurate partition of the scheduling variable space, leading to an aggregate BFR smaller than 1.1. This means that, for $s = 11, 14, 20$, ASGD fails to converge to the batch solution of the discrimination problem (2.7).

Chapter 3

Learning Hybrid Models
Discrete Hybrid Automata

In this chapter the problem of black-box identification of Discrete Hybrid Automata (DHA) is addressed. Discrete hybrid automata result from the connection of a Finite State Machine (FSM) and a switched affine system, that are used to describe the discrete and continuous dynamics of the system, respectively. We present a two-stage approach for identification of DHA, which is an extension of the method for Piecewise PWA regression introduced in chapter 2 and allows us to identify both the switched affine system and the FSM.
The chapter is organized as follows. Discrete hybrid automata and the addressed identification problem are introduced in Sections 3.1–3.2. The two-stage method proposed for DHA identification from data is described in Section 3.3 and, finally, two case studies are presented in Section 3.4 to show the effectiveness of the developed identification method.

3.1 Description of Discrete Hybrid Automata

As a model class, Discrete Hybrid Automata (DHA) are quite general and include, among others, Mixed Logical Dynamical (MLD) models [15], PWA models [104] and max-min-plus-scaling models [56]. To frame

the addressed identification problem, we describe each of the elements composing a DHA, according to the definitions introduced in [110]. Furthermore, we introduce the assumptions made on the DHA in designing the identification method.

A schematic of the functional blocks constituting a DHA is shown in Figure 18, where the information shared among different blocks is underlined.

3.1.1 Switched Affine Systems

A Switched Affine System (SAS), $\mathcal{S}$, is a collection of affine dynamical systems [110]

$$\begin{bmatrix} z_1(k) \\ w_1(k) \end{bmatrix} = \begin{cases} \begin{bmatrix} A_1 x_r(k) + B_1 u(k) + f_1 \\ C_1 x_r(k) + D_1 u(k) + g_1 \end{bmatrix} & \text{if } m(k) = 1, \\ 0_{n_x+n_y} \text{ otherwise} \end{cases}$$

$$\vdots$$

$$\begin{bmatrix} z_s(k) \\ w_s(k) \end{bmatrix} = \begin{cases} \begin{bmatrix} A_s x_r(k) + B_s u(k) + f_s \\ C_s x_r(k) + D_s u(k) + g_s \end{bmatrix} & \text{if } m(k) = s, \\ 0_{n_x+n_y} \text{ otherwise} \end{cases}$$

$$x_r(k+1) = \sum_{i=1}^{s} z_i(k),$$

$$y(k) = \sum_{i=1}^{s} w_i(k).$$

where $k \in \mathbb{N}$ is the discrete time index, $u \in \mathbb{R}^{n_u}$ is the exogenous continuous input vector and $y \in \mathbb{R}^{n_y}$ is the continuous output vector. The index $m(k) \in \{1, \dots, s\}$ specifies the affine state update dynamics at time k and it can only take a finite number of integer values, with s denoting the number of local affine models.

Alternatively, using a state space representation, a SAS is given by

$$\begin{cases} x_r(k+1) = A_{m(k)} x_r(k) + B_{m(k)} u(k) + f_{m(k)} \\ y(k) = C_{m(k)} x_r(k) + D_{m(k)} u(k) + g_{m(k)}, \end{cases} \tag{3.1}$$

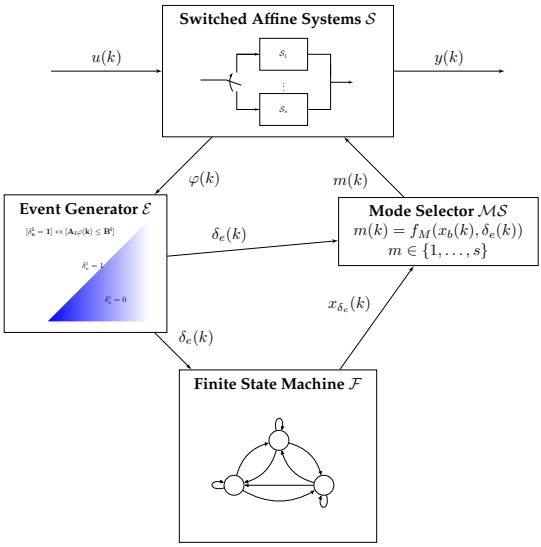


Figure 18: Discrete Hybrid Automata. Scheme of the functional blocks, with $y(k)$ either equal to $y(k)$ or $X(k)$.

For fixed model orders n_a and n_b, a switched affine system $\mathcal{S}$ can also be represented in the input/output form:

$$y(k) = \theta'_{m(k)}\tilde{X}(k), \tag{3.2}$$

where $\theta_{m(k)}$, with $m(k) = 1, \ldots, s$, are the parameter matrices describing the affine dynamical sub-models and $\tilde{X}(k) \in \mathbb{R}^{n_{\tilde{x}}}$ is the extended regressor vector defined as $\tilde{X}(k) = \begin{bmatrix} X(k) & 1 \end{bmatrix}'$, with $X(k) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$ being the collection of past input and output measurements, *i.e.*,

$$X(k) = [y'(k-1) \ldots y'(k-n_a)\; u'(k-1) \ldots u'(k-n_b)]'. \tag{3.3}$$

We focus on switched affine systems represented in the autoregressive input/output form.

3.1.2 Event Generator

An Event Generator (EG) $\mathcal{E}$ generates a Boolean vector $\delta_e(k) \in \mathcal{D} \subseteq \{0,1\}^{n_e}$ according to the satisfaction of affine constraints. Instead of considering

the general definition provided in [110], where the Event Generator is a function of the continuous state $x_r(k)$, the input $u(k)$ and the discrete-time index k, we assume that the EG is characterized by the following relationship

$$\delta_e(k) = f_H(\varphi(k)), \tag{3.4}$$

where $f_H : \Phi \to \mathcal{D} \subseteq \{0,1\}^{n_e}$ is a vector of descriptive functions of a linear hyperplane and $\varphi(k) \in \Phi \subseteq \mathbb{R}^{n_\varphi}$ is either equal to the input $y(k) \in \mathcal{Y} \subseteq \mathbb{R}^{n_y}$ or to the regressor $X(k) \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$. The number of possible events described through the EG is equal to 2^{n_e}, which corresponds to the values that can be taken by the Boolean vector $\delta_e(k)$.
Even though the event generator can be used to model both *time events*[1] and *threshold events*, we only consider EG modelling this second class of as we assume that transitions between different operating conditions are due to the signal φ crossing a threshold. In particular, given matrix $A \in \mathbb{R}^{n_e \times n_\varphi}$ and vector $B \in \mathbb{R}^{n_e}$, threshold events are represented as: $[\delta_{e,i}(k) = 1] \leftrightarrow [A_i\varphi(k) \leq B_i]$, where $\varphi(k) \in \Phi \subseteq \mathbb{R}^{n_\varphi}$ is equal either to the output $y(k)$ or the regressor $X(k)$ and $\delta_{e,i}(k)$ is the i-th component of $\delta_e(k)$.

3.1.3 Finite State Machine

A Finite State Machine (FSM) $\mathcal{F}$ is a discrete dynamic process that evolves according to a logic state update function [110]

$$x_b(k+1) = f_B(x_b(k), u_b(k), \delta_e(k)),$$

where $x_b \in \mathcal{X}_b \subseteq \{0,1\}^{n_{x_b}}$ is the discrete (or logic) state, $u_b \in \mathcal{U}_b \subseteq \{0,1\}^{n_{u_b}}$ is an exogenous Boolean input, $\delta_e(k)$ is the endogenous input coming from the event generator $\mathcal{E}$, and $f_B : \mathcal{X}_b \times \mathcal{U}_b \times \mathcal{D} \to \mathcal{X}_b$ is a deterministic logic function. Only synchronous FSM will be considered, so that transitions may happen only at sampling instants. We assume that no exogenous Boolean input $u_b(k)$ is present, *i.e.*,

$$x_b(k+1) = f_B(x_b(k), \delta_e(k)), \tag{3.5}$$

[1] A time event is modelled as: $[\delta_{e,i}(k) = 1] \leftrightarrow [kT_s \geq t_o]$, with T_s indicating the sampling time, and where t_o is a given time and $\delta_{e,i}(k)$ is the i-th component of $\delta_e(k)$.

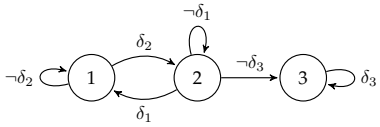


Figure 19: Finite State Machine with 3 states and no exogenous Boolean inputs $u_b(k)$.

namely the FSM is entirely driven by the output of the event generator δ_e. An example of a finite state machine, with 3 states and that is driven by the events δ_i, $i = 1, 2, 3$, is shown in Figure 19. The logic state update function associated with the FSM in Figure 19 is

$$x_b(k+1) = \begin{cases} 1 \text{ if } ((x_b(k) = 1) \wedge \neg\delta_2) \vee ((x_b(k) = 2) \wedge \delta_1) \\ 2 \text{ if } ((x_b(k) = 1) \wedge \delta_2) \vee ((x_b(k) = 2) \wedge \neg\delta_1) \\ 3 \text{ if } ((x_b(k) = 2) \wedge \neg\delta_3) \vee ((x_b(k) = 3) \wedge \delta_3), \end{cases}$$

where the following *connectives* are introduced: "$\wedge$" (and), "$\vee$" (or) and "$\neg$' (not). Note that the conditions associated with the edges of the oriented graph in Figure 19 are mutually exclusive, as (3.5) is a deterministic function.

3.1.4 Mode Selector

The Mode Selector (MS) $\mathcal{MS}$ is a Boolean function $f_M : \mathcal{X}_b \times \mathcal{D} \to \{1, \ldots, s\}$:

$$m(k) = f_M(x_b(k), \delta_e(k)), \tag{3.6}$$

where $m(k)$ is the selected dynamic sub-model of the SAS, called *active mode*. We assume that the mode selector $\mathcal{MS}$ is only a function of the current logic state $x_b(k)$. This means that one and only one mode $m(k)$ is associated to the logic state $x_b(k)$ and, as a consequence, that the active mode $m(k)$ is driven by the logic state update function f_B in (3.5). Because of this assumption, with some abuse of notation, the terms *logic/discrete state* and *active mode* will be interchangeably used.

3.2 Problem statement

Based on the definitions introduced in Section 3.1, the problem of black-box identification of DHA can be formally stated as follows:

Problem 1 Discrete Hybrid Automaton identification
Consider an unknown data-generating Discrete Hybrid Automaton:

$$\mathcal{H} = \{\mathcal{S}, \mathcal{E}, \mathcal{F}, \mathcal{MS}\}, \tag{3.7}$$

where $\mathcal{S}$ is the correspondent SAS, $\mathcal{E}$ is the Event Generator, $\mathcal{F}$ is the associated FSM and $\mathcal{MS}$ is the Mode Selector.
Given a set of N input/output pairs $\{u(k), y(k)\}_{k=1}^{N}$ generated by the "true"system $\mathcal{H}$ in (3.7), the *DHA identification problem* aims at estimating a DHA model describing the input/output relationship between the data $\{u(k), y(k)\}_{k=1}^{N}$, under the hypothesis that the measured outputs $y(k)$ may be noisy and the sequence of active modes $m(k)$ is not directly observable. ■

To solve problem 1 one is required to:

- **(i)** choose the number of local affine sub-models s, *i.e.*, the number of different values of the logic state x_b defining the FSM $\mathcal{F}$. In the design of the identification strategy, the value of s is assumed to be fixed by the user. If the number of sub-models is not known a priori, s can be chosen through cross-validation, with a possible upper-bound dictated by the maximum tolerable complexity of the estimated DHA model. It is worth remarking that the number of possible logic states should be selected to trade-off between data fitting and model complexity.
- **(ii)** estimate the parameter matrices $\{\theta_i\}_{i=1}^{s}$ associated to each affine dynamical system of $\mathcal{S}$ (see eq. (3.2));
- **(iii)** identify, from the regressor/output pairs $\{X(k), y(k)\}_{k=1}^{N}$, the transitions between the active modes (or equivalently, based on our assumptions, between the discrete states x_b) and derive the conditions leading to such transitions.

3.3 DHA identification algorithm

A two-step strategy is used to tackle the DHA identification problem. Specifically, the algorithm consists of the following stages:

S1. Simultaneous *clustering* of the data $\{X(k), y(k)\}_{k=1}^{N}$ and *estimation* of the parameter matrices $\theta_1, \ldots, \theta_s$. This is performed recursively, by processing the training samples $\{X(k), y(k)\}_{k=1}^{N}$ sequentially.
In this stage we thus estimate the parameters $\{\theta_i\}_{i=1}^{s}$ and we associate each data pair $\{X(k), y(k)\}$ with a cluster $\mathcal{C}_i$, $i \in \{1, \ldots, s\}$, where $\mathcal{C}_i$ represents the set of points associated to the i-th logic state of the system. Furthermore, the results of the clustering procedure allow us to reconstruct the sequence of *active modes* $\mathcal{M} = \{m(k)\}_{k=1}^{N}$.

S2. *Learning* s different partitions $\{\Phi_j\}_{j=1}^{s}$ of the space Φ. it is worth remarking that Φ is the space where φ is defined, with $\varphi(k)$ either equal to $y(k)$ ($\Phi = \mathcal{Y}$) or the regressor $X(k)$ ($\Phi = \mathcal{X}$). Each partition Φ_j implicitly defines the function $f_H(\varphi(k))$ in equation (3.4), describing the event generator $\mathcal{E}$, and the logic state update rule $x_b(k+1) = f_B(x_b(k), \delta_e(k))$ given that the active mode $m(k)$ at the current time k is $m(k) = j$.
More specifically, after estimating the sequence of *active modes* $\mathcal{M} = \{m(k)\}_{k=1}^{N}$ at stage S1, s different clusters $\{\mathcal{C}_j^{i^+}\}_{i^+=1}^{s}$ are constructed, one for each possible mode $j = 1, \ldots, s$. Each cluster $\mathcal{C}_j^{i^+}$ contains the samples $\{\varphi(k)\}_{k=1}^{N-1}$ such that $m(k) = j$ and $m(k+1) = i^+$. The j-th partition Φ_j of the space Φ is thus obtained by separating the clusters $\{\mathcal{C}_j^{i^+}\}_{i^+=1}^{s}$ through a multi-class linear separator, computed using the Newton-like method presented in chapter 2, Section 2.2.2.

As already remarked $\varphi \in \Phi$ is supposed to be either the measured output or the regressor. In particular, among these two possibilities, the signal φ driving the transition between different logic states can be selected either through cross-validation or on the basis on prior knowledge on the true system.

Before providing any technical detail, it is worth remarking that the introduced two-stage algorithm is based on a proper extension of the method for PWA regression presented in chapter 2. The main difference with respect to the method described in chapter 2 is that we account for the time evolution of the logic states, through the construction of s different partitions $\{\Phi_j\}_{j=1}^{s}$ of the space Φ.

Remark 2 *The order in which the samples are collected plays an important role in the estimation of the switching laws. As a consequence, the order of the data must not be altered through permutations of the sequence* $\{X(k), y(k)\}_{k=1}^{N}$. ■

3.3.1 S1. Iterative clustering and parameter estimation

The iterations of Stage S1 are summarized in Algorithm 5. To recursively update the values of the parameters θ_i and to generate the clusters $\mathcal{C}_i$, we use an approach similar to the one presented in chapter 2 (see Algorithm 2) for recursive clustering and parameter estimation. The methods summarized in Algorithm 5 and Algorithm 2 are based on different clustering polices. In particular, in Step 3.1 of Algorithm 5 we consider the modelling error, while in Algorithm 2 we account for both the fitting error and the spatial distribution of the data.
The main idea of Algorithm 5 is to compute, at each time instant k, the estimation error e_i ($i \in \{1, \dots, s\}$) provided by all the s local affine submodels and select the one that "best fits" the current output observation $y(k)$ (Steps 3.1 and 3.2). Once the "best" sub-model is chosen, the sequence of *active modes* $\mathcal{M}$ is updated as in Step 3.4 and, at Step 3.5, the parameter matrix $\theta_{m(k)}$ associated to the selected model is updated using the recursive least-squares algorithm in [2], based on inverse QR decomposition.

As pointed out in chapter 2, due to the greedy nature of Algorithm 5, the estimates θ_i and the clusters $\mathcal{C}_i$, with $i = 1, \dots, s$, will be influenced by the initialization of the parameters θ_i. A possible choice for the initial values of the matrices θ_i is to take $\theta_1, \dots, \theta_s$ all equal to the best linear

Algorithm 5 Recursive clustering and parameter estimation algorithm

Input: Sequence of observations $\{X(k), y(k)\}_{k=1}^{N}$, desired number s of logic states, initial condition for the parameter matrices θ_i, $i = 1, \ldots, s$.

1. **let** $\mathcal{M} \leftarrow \mathbf{0}_{N\times 1}$;
2. **let** $\mathcal{C}_i \leftarrow \emptyset$, $i = 1, \ldots, s$;
3. **for** $k = 1, \ldots, N$ **do**
   - 3.1. **let** $e_i(k) \leftarrow y(k) - \theta_i' \tilde{X}(k)$, $i = 1, \ldots, s$;
   - 3.2. **let**
   $$m(k) \leftarrow \arg \min_{i=1,\ldots,s} e_i'(k) e_i(k); \tag{3.8}$$
   - 3.3. **let** $\mathcal{C}_{m(k)} \leftarrow \mathcal{C}_{m(k)} \cup \{y(k), X(k)\}$;
   - 3.4. **let** $\mathcal{M}(k) \leftarrow m(k)$;
   - 3.5. **update** $\theta_{m(k)}$ using the recursive least-squares Algorithm [2];
4. **end for**;
5. **end**.

Output: Estimated matrices $\theta_1, \ldots, \theta_s$, clusters $\mathcal{C}_1, \ldots, \mathcal{C}_s$ and sequence of active modes $\mathcal{M}$.

model, *i.e.*,

$$\theta_i \equiv \arg\min_{\theta} \sum_{k=1}^{N} \|y(k) - \theta' X(k)\|_2^2, \ \forall i = 1, \dots, s. \tag{3.9}$$

As suggested in Chapter 2, the accuracy of the estimates can be enhanced reiterating Algorithm 5 multiple times, using its output as an initial condition for the following iteration.

Dealing with logic states with equal continuous dynamics

Suppose that Algorithm 5 is used to identify a DHA system with two or more logic states sharing the same continuous dynamics (3.1). In this setting, it might be difficult to accurately reconstruct the discrete behavior of the DHA, since data points originated from different logic states may be associated with the same cluster.
Considering that s is fixed, due to errors in the clustering procedure (Step 3.8) it might happen that a clusters $\mathcal{C}_\emptyset$ is "almost empty", *i.e.*, the points associated to $\mathcal{C}_\emptyset$ are outliers. The remaining clusters might not be linearly separable and, consequently, the function f_H characterizing the event generator $\mathcal{E}$ may not be correctly retrieved.

We introduce an heuristic to overcome this problem, extending Algorithm 5 as summarized in Algorithm 6. It is worth underlining that Algorithm 6 has to be run after Algorithm 5.

The number of modes s is supposed to be known a priori and, consequently, none of the clusters obtained with Algorithm 5 should be "empty". At step 2 of Algorithm 6, we find the number $c_\emptyset$ of "almost empty" clusters, with $\mathcal{C}_i$ ranked as "almost empty" if it contains less than εN samples. The tunable parameter ε is used to select the minimum dimension of the cluster for it not to be classified as "almost empty"and it allows us to account for the fact that outliers can be assigned to "empty" clusters. When $c_\emptyset \neq 0$, we assume that data points belonging to the "almost empty" clusters have been erroneously assigned to other logic states. As the clustering policy used in is Algorithm 5 is only based on

Algorithm 6 Splitting clustering algorithm

Input: Sequence of training samples $\{X(k), y(k)\}_{k=1}^{N}$, desired number s of logic states; parameter matrices θ_i and clusters $\mathcal{C}_i$ ($i = 1, \ldots, s$) provided by Algorithm 5; threshold $\varepsilon \geq 0$.

1. **let** $\mathrm{c}_\emptyset \leftarrow 0$;
2. **for** $i = 1, \ldots, s$ **do**
   2.1. **if** $\frac{|\mathcal{C}_i|}{N} \leq \varepsilon$ **let** $\mathrm{c}_\emptyset \leftarrow \mathrm{c}_\emptyset + 1$;
3. **end for**;
4. **let** $\bar{s} = s - \mathrm{c}_\emptyset$;
5. **compute** new clusters $\bar{\mathcal{C}}_i$ and parameter matrices θ_i $i = 1, \ldots, \bar{s}$ through Algorithm 5;
6. **compute** the clusters' sample covariance matrices R_i, $i = 1, \ldots, \bar{s}$;
7. **let** $\Sigma_i \leftarrow \mathrm{tr}(R_i)$, $i = 1 : \ldots, \bar{s}$;
8. **let** $\mu \leftarrow \bar{s}$ and $\beta \leftarrow \bar{s}$;
9. **while** $\mu < s$ **do**
   9.1. **let** $j^\star \leftarrow \arg\max_{j \in \{1,\ldots,\beta\}} \Sigma j$;
   9.2. **let** $l^\star \leftarrow \arg\min_{l \in \{1,\ldots,s-\mu+1\}} DB(l)$;
   9.3. **divide** $\bar{\mathcal{C}}_{j^\star}$ into $l^\star$ clusters through K-means [55];
   9.4. **associate** the parameter vector $\theta_{j^\star}$ to the obtained clusters;
   9.5. **let** $\beta \leftarrow \beta - 1$ and **remove** $\bar{\mathcal{C}}_{j^\star}$ from the set of clusters that can be partitioned;
   9.6. **update** $\mu \leftarrow \mu + (l^\star - 1)$;
   9.7. **let** $\mathcal{C}_i$, $i = 1, \ldots, \mu$ **be** the new collection of clusters and θ_i the associated parameter matrices;
10. **end while**;

Output: Estimated parameters $\{\theta_i\}_{i=1}^{s}$, clusters $\{\mathcal{C}_i\}_{i=1}^{s}$ and updated sequence of active modes $\mathcal{M}$.

the modeling error, we further suppose that the samples associated with an "almost empty" cluster have been assigned to a mode characterized by the same continuous dynamics of the "undetected" one. Based on these considerations, Algorithm 5 is then run again with a reduced number of modes $\bar{s} = s - c_{\emptyset}$ (Steps 4-5) to improve the accuracy of the estimated the sub-models, *i.e.*, the quality of the estimates θ_i, $i = 1, \ldots, \bar{s}$. Nonetheless, as the number of logic states s is supposed to be known a priori, also the parameter matrices associated with the discarded $s - \bar{s}$ logic states have to be estimated. Steps 6-10 allow us to accomplish this task. At Step 7, we calculate the trace Σ_i of the covariance matrix R_i (computed at step 6) of the clusters $\bar{\mathcal{C}}_i$, with $i = 1, \ldots, \bar{s}$. The quantities $\{\Sigma_i\}_{i=1}^{\bar{s}}$ are used to evaluate the *dispersion* of the clusters $\{\mathcal{C}_i\}_{i=1}^{\bar{s}}$. Starting from the cluster with maximum dispersion $\bar{\mathcal{C}}_{j^\star}$, K-means [55] is then iteratively applied (step 9.3) until the s clusters are retrieved. In particular, at step 9, $\bar{\mathcal{C}}_{j^\star}$ is partitioned into $l^\star$ sub-clusters, with $l^\star$ (computed at stage 9.2) set as the value corresponding to the clustering solution minimizing the Davies-Bouldin index, *i.e.*,

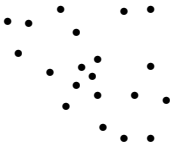
$$l^\star = \underset{l \in \{1 \ldots, \bar{s}\}}{\operatorname{argmin}} \, DB(l),$$

with the Davies-Bouldin index [41] defined as

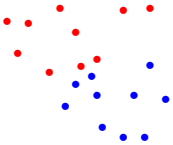
$$DB(l) = \frac{1}{l} \sum_{i=1}^{l} \max_{j \neq i} \left(\frac{\bar{d}_i + \bar{d}_j}{d_{ij}} \right). \tag{3.10}$$

In equation (3.10), l is the number of sub-clusters, $\bar{d}_i$ is the i-th cluster within-cluster distance and d_{ij} is the distance between the centroids of the i-th and the j-th cluster. Once $\bar{\mathcal{C}}_{j^\star}$ is partitioned, the parameter vector $\theta_{j^\star}$ is associated with all the obtained sub-clusters (see Step 9.4) and $\bar{\mathcal{C}}_{j^\star}$ is removed from the set of clusters that can further be divided (Step 9.5). This procedure is repeated until the desired number s of cluster is retrieved.

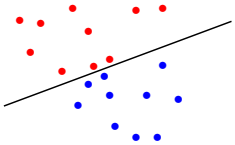
Algorithm 6 allows us to retrieve the s clusters, the parameter matrices $\{\theta_i\}_{i=1}^{s}$ and the resulting sequence of active modes $\mathcal{M}$.



(a) Points assigned to $\mathcal{C}_1$ at Stage S1



(b) Result of Step 2.1. Set of points assigned to: $\mathcal{C}_1^1$ (red) and $\mathcal{C}_1^2$ (blue)



(c) Result of Step 2.3. Linear separator between $\mathcal{C}_1^1$ and $\mathcal{C}_1^2$

Figure 20: Learning the discrete dynamics of a DHA with 2 modes. Results obtained performing Step 2 of Algorithm 7 for $j = 1$, based on $\mathcal{C}_1$ computed at Stage S1.

3.3.2 Learning the discrete dynamics

Although Stage S1 does not provide a direct estimation of the law driving the discrete dynamics, the underlying FSM can be identified from the estimated sequence of *active modes* $\mathcal{M}$.

For each sample $\varphi(k)$, with $\varphi(k)$ supposed to be equal either to $y(k)$ or $X(k)$, sequence $\mathcal{M}$ gives both the estimated active mode $m(k)$ and the one-step ahead logic state $m(k+1)$.

Algorithm 7 summarize the procedure used to reconstruct the logic state

dynamics. Specifically, we compute s different polyhedral partitions $\{\Phi_j\}_{j=1}^{s}$ of the space Φ, with each partition Φ_j computed independently from the others. At step 2.1, s different clusters $\{\mathcal{C}_j^{i^+}\}_{i^+=1}^{s}$ are constructed for each $j = 1, \ldots, s$. Each cluster $\mathcal{C}_j^{i^+}$ contains the samples $\{\varphi(k)\}_{k=1}^{N-1}$ such that $m(k) = j$ and $m(k+1) = i^+$. The j-th partition Φ_j of space Φ is thus obtained by separating the clusters $\{\mathcal{C}_j^{i^+}\}_{i^+=1}^{s}$ through the multi-class linear separation method presented in Chapter 2 and summarized in Algorithm 3 (step 2.3).
When applied to a DHA with 2 discrete states, the results obtained performing Steps 2.1-2.3 of Algorithm 7 (for $j = 1$) are summarized in Figure 20.

Algorithm 7 Logic state dynamics identification algorithm

Input: Samples $\{\varphi(k)\}_{k=1}^{N}$; identified sequence of logic states $\mathcal{M} = \{m(k)\}_{k=1}^{N}$.

1. **let** $\mathcal{C}_j^{i^+} \leftarrow \emptyset, \quad j, i^+ = 1, \ldots, s$;
2. **for** $j = 1, \ldots, s$ **do**
   - 2.1. **for** $i^+ = 1, \ldots, s$ **do**
     - 2.1.1. **for** $k = 1, \ldots, N-1$ **do**
       - 2.1.1.1. **if** $m(k) = j$ and $m(k+1) = i^+$
       - 2.1.1.2. **let** $\mathcal{C}_j^{i^+} \leftarrow \mathcal{C}_j^{i^+} \cup \{\varphi(k)\}$;
       - 2.1.1.3. **end if**;
     - 2.1.2. **end for**;
   - 2.2. **end for**;
   - 2.3. for all $j = 1, \ldots, s$, **compute** the polyhedral partition Φ_j of the space Φ by separating the clusters $\mathcal{C}_j^{i^+}$ (with $i^+ = 1, \ldots, s$) through the multi-class linear separation method in Algorithm 3.
3. **end for**;

Output: Polyhedral partitions Φ_j, $j = 1, \ldots, s$.

3.4 Case studies

To show the effectiveness of the proposed identification technique, a simulation example and a simple experimental case study are considered. In both the examples, it is assumed to be known a priori that $\varphi = y$ and Algorithm 5 is executed repeatedly until convergence.
The estimated models are validated over datasets not used for training, assuming that the initial logic state of the data-generating system is known a priori. The quality of the learned DHA is evaluated comparing the (open-loop) simulation output $\hat{y}$ and the measured output y, with the time evolution of the logic state is simulated through the polyhedral partitions computed by Algorithm 7. To quantitatively assess the accuracy of the estimated models we use the the Best Fit Rate (BFR) indicator, defined as:

$$\text{BFR} = \max\left\{1 - \frac{||\hat{y} - y||_2}{||y - \bar{y}||_2}, 0\right\} \cdot 100\% \tag{3.11}$$

where $\bar{y}$ is the sample mean of y.

All computations are carried out on a 2.8-GHz Intel Core i7 with 16 GB of RAM running MATLAB R2015a.

3.4.1 Simulation Example: Identification of a 4 modes DHA

Consider the DHA with 4 modes depicted in Figure 21, with continuous dynamics given by the following SAS

$$y(k) = \begin{cases} 0.9y(k-1) + 6 & \text{if } m(k) = 1 \\ 0.8y(k-1) + 20 & \text{if } m(k) = 2 \\ 0.9y(k-1) + 6 & \text{if } m(k) = 3 \\ 0.7y(k-1) & \text{if } m(k) = 4. \end{cases} \tag{3.12}$$

Each sub-system in (3.12) is described by a first-order difference equation, with no exogenous input signal and the local models $\mathcal{S}_1$ and $\mathcal{S}_3$, *i.e.*,

$$\mathcal{S}_1 : y(k) = 0.9y(k-1) + 6,$$
$$\mathcal{S}_3 : y(k) = 0.9y(k-1) + 6,$$

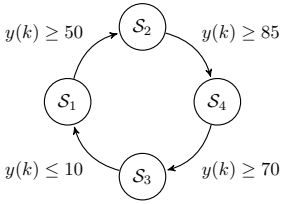


Figure 21: Simulation Example. DHA to be identified.

sharing the same continuous dynamics. Note that, in this example, the regressor $X(k)$ is equal to the output $y(k-1)$ at step $k-1$ and $\Phi = \mathcal{Y} = \mathbb{R}$. The system is estimated using a set of $N = 5000$ outputs, with $y(k)$ generated as

$$y(k) = \theta_{m(k),1} y(k-1) + \theta_{m(k),2} + e_{\mathrm{o}}(k),$$

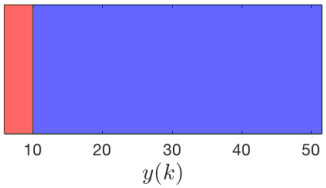
and e_{o} being a zero-mean white noise sequence with Gaussian distribution $e_{\mathrm{o}} \sim \mathcal{N}(0, 0.04)$. The effect of the noise on the output is assessed through the Signal-to-Noise Ratio (SNR)

$$\mathrm{SNR} = 10 \log \left\{ \frac{\sum_{k=1}^{N} \left(y(k) - e_{\mathrm{o}}(k) \right)^2}{\sum_{k=1}^{N} e_{\mathrm{o}}^2(k)} \right\} = 34.9 \text{ dB}.$$

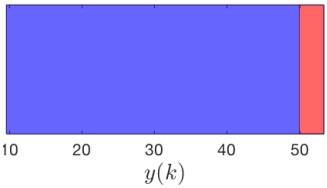
To assess the performance of the heuristic summarized in Algorithm 6, Stage S1 is performed using only Algorithm 5 and then running Algorithms 5-6. Algorithm 5 is always run 10 times, with the number of modes $s = 4$ known a priori. The threshold ε in Algorithm 6 is selected through cross-validation on a set of 100 noisy samples not employed in the training phase. The chosen threshold is $\varepsilon = 0.05$, meaning that "almost empty" clusters are the ones containing less than $250 = N\varepsilon$ samples. The tests performed using different values of ε have shown that the data-generating system is not correctly identified if $\varepsilon < 0.05$ or $\varepsilon > 0.2$. The estimated parameters are reported in Table 11, showing that only Algorithm 6 allows us to retrieve an accurate estimates for θ_1 and θ_3. On the other hand, θ_3 is not correctly estimated when only Algorithm 5. Based on the active mode sequence $\mathcal{M}$ estimated with Algorithm 6, the $s = 4$ partitions Φ_j ($j = 1, \ldots, 4$) of space Φ are computed through Algorithm 7. On average, the CPU time required to compute each partition

Table 11: Simulation Example. θ_i vs $\hat{\theta}_i$, $i = 1, \ldots, s$.

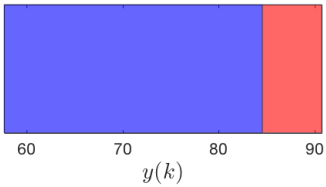
	S_1		S_2		S_3		S_4	
	$\theta_{1,1}$	$\theta_{1,2}$	$\theta_{2,1}$	$\theta_{2,2}$	$\theta_{3,1}$	$\theta_{3,2}$	$\theta_{4,1}$	$\theta_{4,2}$
true	0.9	6	0.8	20	0.9	6	0.7	0
Algorithm 5	0.90	5.96	0.79	20.64	-5.59	263.95	0.70	0.01
Algorithm 6	0.90	5.96	0.79	20.64	0.90	5.96	0.70	0.01



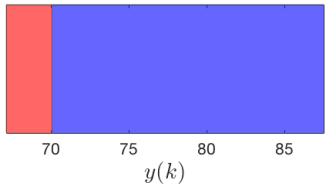
(a) $m(k) = 1$. Blue: $m^+(k) = 1$, red: $m^+(k) = 2$.



(b) $m(k) = 2$. Blue: $m^+(k) = 2$, red: $m^+(k) = 3$.



(c) $m(k) = 3$. Blue: $m^+(k) = 3$, red: $m^+(k) = 4$.



(d) $m(k) = 4$. Blue: $m^+(k) = 4$, red: $m^+(k) = 1$.

Figure 22: Simulation example. Partitions computed through Algorithm 7, with $m^+(k) = m(k+1)$.

Φ_j ($j = 1, \ldots, s$) is 0.024 s, while the entire identification procedure takes 1.215 s to train the whole model.

The estimated partitions, reported in Figure 22, are all characterized by two polytopes. We thus accurately model the underlying DHA, where each logic state can be followed only by two other states (including itself). The computed linear separators allows us to retrieve estimates for the switching conditions characterizing the logic dynamics of the considered data-generating system. To further assess the performance of the approach, Figure 23 shows the oriented graphs on whose edges are re-

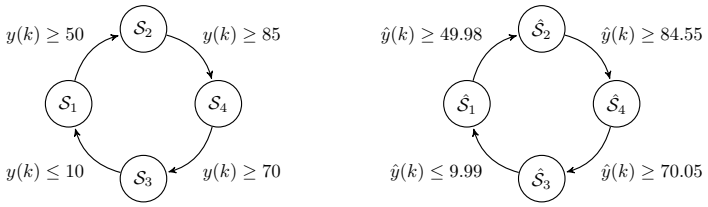


Figure 23: Simulation example. True (left panel) vs estimates (right panel) DHA.

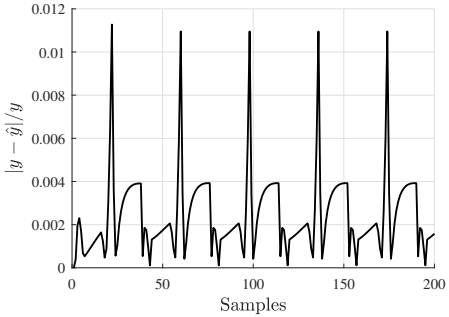


Figure 24: Simulation Example. Relative error $\frac{|y-\hat{y}|}{y}$

ported the actual and estimated switching conditions, respectively. As we assume to have no prior knowledge on the aforementioned conditions, the negligible difference between the true and estimated guards is due to the observed data-points.

The identified model is further validated on a noiseless dataset of length $N_V = 200$. The obtained BFR is 99.51%, showing the effectiveness of the proposed identification method, even in the challenging situation of two modes sharing the same continuous dynamics. The relative error between the actual and simulated output is shown in Figure 24, with the maximum error obtained in correspondence of mode transitions.

3.4.2 Modelling of switching RC circuit

The proposed algorithm is also tested against an experimental dataset, consisting on the output voltage of the RC circuit with switching load, whose schematic is depicted in Figure 25. The oriented graph describing

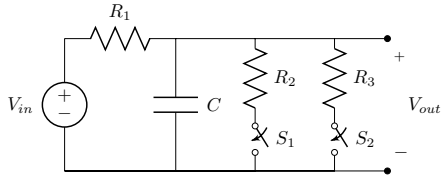


Figure 25: Schematic of the switching RC circuit.

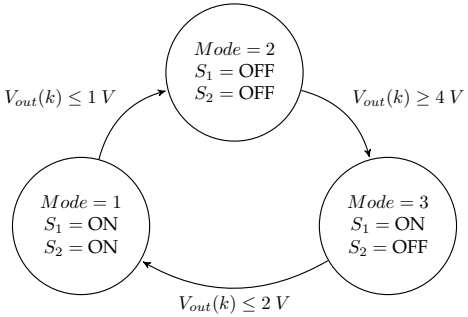
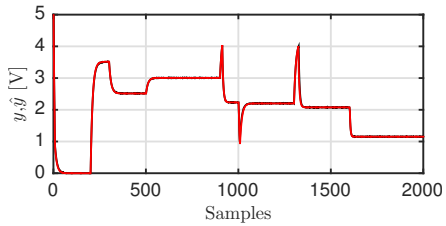


Figure 26: Scheme of the discrete state dynamics of the system

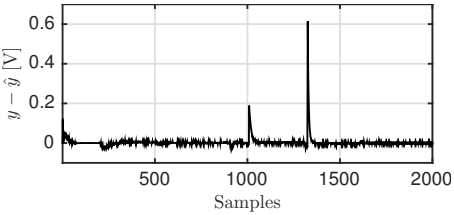
the switching logic of the circuit is represented in Figure 26 where, as in the previous example, only the conditions leading to a change in the state are reported. If the conditions on the edge are not satisfied, the logic state does not change.

A 10 μF capacitor C and three 10 kΩ resistors R_1, R_2 and R_3 are used. The ON/OFF switches are implemented using MOSFETs. An Arduino UNO board is used for: (i) measuring the output voltage $y(k) = V_{out}(k)$; (ii) generating the input voltage $u(k) = V_{in}(k)$ and (iii) driving the switches according to the logic behavior, represented in Figure 26. Two different piecewise-constant signals are applied as input voltage V_{in} to generate the training and the validation datasets. Both the input signals V_{in}^{train} (training) and V_{in}^{val} (validation) have length 2000. The output voltage $V_{out}(k)$ is measured, at a sampling time of T_s = 100 ms, with an analog-to-digital (A/D) converter available on the Arduino board[2].
The number of logic states is set to $s = 3$ and all the local models are

[2]The A/D converter used in the experiment has an input rage of 0-5 V and a resolution of 10 bits.



(a) V_{out} vs $\hat{V}_{out}$. Black: true (y), red: simulated ($\hat{y}$)



(b) V_{out} vs $\hat{V}_{out}$. $y - \hat{y}$

Figure 27: True V_{out} vs estimated $\hat{V}_{out}$ output voltage

described by a first-order linear difference equation, relating $y(k)$ to the regressor $X(k) = [y(k-1)\, u(k-1)]'$. As the same parameters are retrieved using Algorithm 5 or Algorithms 5-6 ($\varepsilon = 0.05$), we perform Stage S1 through Algorithm 5 (which is iterated 10 times). The overall computational time required to identify a DHA model of the circuit is 0.078 s, of which 0.033 s are taken to compute the 3 partitions of space Φ.

To assess the quality of the estimated model, the (open-loop) simulated output $\hat{y}$ of the estimated DHA model is plotted in Figure 27, along with the actual output y. Note that the plots refer to the validation dataset. As the estimated and actual output overlaps (see Figure 3.27(a)), the simulation error $y(k) - \hat{y}(k)$ is also shown for the sake of a better visualization of the performance (see Figure 3.27(b)). Only once the error exceeds 0.4 V.
A BFR of 98.64% is achieved and the 0.4% of the data-points in the validation set are assigned to the wrong mode, *i.e.*, only 8 times out of 2000

the mode is not correctly reconstructed. The error in the prediction of the logic state depends on (unavoidable) mismatch between the true and the estimated switching conditions. Nevertheless, this discrepancy vanishes after few steps.

Chapter 4

Learning Jump Models

This chapter is dedicated to the description of two algorithms designed for learning Jump Models (JM). This class of models allows one to describe systems characterized by multiple operating conditions, like piecewise affine functions or Discrete Hybrid Automata (DHA), but it is more general as concerns the mechanism regulating the transition between different modes. Piecewise affine functions (chapter 2) and Discrete Hybrid Automata(chapter 3) can be seen particular instances of this class.
In this chapter we focus on the problem of identifying two specific JMs, Rarely Jump Models (RJMs) and Markov Jump Models (MJMs). The method presented to tackle these problems rely on the assumption that neither the local models nor the laws governing the transitions between different models are known a priori.
The chapter is organized as follows. Section 4.1 is devoted to the formulation of the addressed learning problem. In Section 4.2 the algorithms for learning Rarely Jump and Markov Jump Models are then described in detail. The method proposed to learn MJMs from data is inspired by the approaches for PWA regression and DHA proposed in chapter 2. A set of case studies, two of which involving experimental data, are reported in Section 4.3.

4.1 Problem formulation

Consider a dataset $\mathcal{D}_T = \{X_t, y_t\}_{t=1}^T$ of regressor/output pairs $\{X_t, y_t\}$, generated by an unknown nonlinear function f^{o}. The measured output is supposed to be obtained as

$$y_t = f^{\text{o}}(X_t) + e_t. \tag{4.1}$$

with e_t being an additive noise sequence and $t \in \mathbb{N}$ denoting the time index. We aim at finding a function f that approximates the relation between the regressor X_t and the output y_t,

$$y_t \approx f\left(X_t\right), \tag{4.2}$$

The function f is parametrized as a collection of K local models, each of them describing the behavior of the underlying data-generating function f^{o} at a given mode. To account for changes in the operating conditions, we introduce the discrete state $s(t) \in \mathcal{K} = \{1, \ldots, K\}$ to indicate which of the K models is active at time t. The model f to be learned from data (see (4.2)) is then given by

$$f\left(X_t\right) = f_{s(t)}(X_t, \theta_{s(t)}). \tag{4.3}$$

where $f_{s(t)}$ is the local model active at time t and $\theta_{s(t)} \in \mathbb{R}^{n_\theta}$ is the associated vector of unknown parameters to be estimated from data.

The problem of fitting K models is formulated as the minimization of the cost function

$$J(X_t, y_t, \Theta, \mathcal{S}) = \frac{1}{T}\sum_{t=1}^{T} \ell(X_t, y_t, \theta_{s(t)}) + \gamma \sum_{k=1}^{K} r(\theta_k) \tag{4.4}$$

with respect to the parameters $\Theta = \{\theta_k\}_{k=1}^K$ and the sequence $\mathcal{S}$ of discrete states, *i.e.*, $\{s(t)\}_{t=1}^T$. The loss function ℓ in (4.4) accounts for the fitting error, while r is a regularization term introduced to reduce model complexity, thus avoiding over fitting. The tuning hyper-parameter $\gamma \in \mathbb{R}^+$ is used to balance the trade-off between data fitting and model complexity.

It is worth remarking that the sequence $\mathcal{S}$ of active modes is not directly observable and must be retrieved from $\mathcal{D}_T$ along with the model parameters Θ.

When the cost in (4.4) is minimized, the ordering imposed by the label t does not affect the accuracy of the estimated model. The order of the data can thus be permuted. However, if the state $s(t)$ evolves accordingly to a certain time pattern the ordering of the data is relevant and must be taken into account in estimating the model f. In this chapter we consider two different modeling frameworks that allow us to exploit the information provided by the temporal order of the data:

$\mathcal{M}1$. *Rarely Jump Model* (RJM): the discrete state $s(t)$ is supposed to change rarely over time, so that $s(t) = s(t-1)$ for most of the time instants $t = 2, \dots, T$;

$\mathcal{M}2$. *Markov Jump Model* (MJM): the transitions of the discrete state are modelled through a discrete-time Markov Chain. The jumps thus satisfy the Markov property

$$P\{s(t) = \sigma_t | \{s(\tau) = \sigma_\tau\}_{\tau=1}^{t-1}\} = P\{s(t) = \sigma_t | s(t-1) = \sigma_{t-1}\}, \tag{4.5}$$

with $\sigma_t \in \mathcal{K}$ for $t = 1, \dots, T$, and $P(\mathbb{E})$ indicating the probability associated to the event $\mathbb{E}$.
The probability of obtaining a certain sequence $\{\sigma_t\}_{t=1}^T$ is equal to

$$P\{s(1) = \sigma_1, \dots, s(T) = \sigma_T\} = \prod_{t=2}^{T} \pi_{\sigma_{t-1},\sigma_t} P\{s(1) = \sigma_1\},$$

where $\pi_{\sigma_{t-1},\sigma_t} = P\{s(t) = \sigma_t | s(t-1) = \sigma_{t-1}\}$ and $P\{s(1) = \sigma_1\}$ is the probability associated with the initial discrete state σ_1. The transition matrix of the underlying Markov chain describing the time evolution of the discrete state is denoted by Π and its (i,j)-entry is $\pi_{i,j}$, that is the transition probability from state $s(t-1) = i$ to state $s(t) = j$.

If the regressor X_t contains past values of input and output signals of a dynamical system, that is

$$X_t = \left[y'_{t-1} \ \dots \ y'_{t-n_a} u'_{t-1} \ \dots \ u'_{t-n_b} \right]', \tag{4.6}$$

and the sub-models are linear, the approximated models $\mathcal{M}_1$ and $\mathcal{M}_2$ have the form of a Jump Linear System (JLS) [40].

4.2 Learning algorithms

This section is devoted to the description of the algorithms developed for Jump Model learning, focusing on the identification of Rarely Jump and Markov Jump Models. Before providing further technical details, it is worth underlining that we consider the general convex cost functions J as in (4.4) when learning RJMs, while we limit to quadratic costs

$$J(X, y, \Theta, \mathcal{S}) = \frac{1}{T} \sum_{t=1}^{T} \underbrace{\|y_t - \theta'_{s(t)} X_t\|_2^2}_{\ell(X_t, y_t, \theta_{s(t)})} + \gamma \sum_{k=1}^{K} \underbrace{\theta'_k W_k \theta_k}_{r(\theta_k)}, \tag{4.7}$$

with W_k being a known positive-semidefinite weight matrix of proper dimensions when identifying MJMs. Such a restriction on the cost functions for Markov Jump Models identification allows us to develop a strategy that can be applied both for batch (off-line) and recursive (on-line) learning.

The number K of local sub-models is not chosen by the algorithm, but it must be selected by cross-validation, unless it is known a priori, with an upper-bound on K dictated by the maximum tolerated complexity of the model.

4.2.1 $\mathcal{M}_1$. Learning Rarely Jump Models

The problem of learning Rarely Jump Models is tackled through a two-stage strategy, alternating minimization over the model parameters Θ and over the sequence of discrete state $\mathcal{S}$ of the following modification of

the cost (4.4):

$$\frac{1}{T}\left[\sum_{t=1}^{T-1}\left(\ell(X_t, y_t, \theta_{s(t)}) + \lambda \mathbf{1}_{[s(t+1)\neq s(t)]}\right) + \ell(x_T, y_T, \theta_{s(T)})\right] + \gamma \sum_{k=1}^{K} r(\theta_k). \tag{4.8}$$

The additional term

$$\mathbf{1}_{[s(t+1)\neq s(t)]} = \begin{cases} 1 & \text{if } s(t+1) \neq s(t) \\ 0 & \text{otherwise}, \end{cases} \tag{4.9}$$

multiplied by the scalar weight $\lambda \geq 0$, is introduced to penalize the changes of discrete state according to the modeling hypothesis that the discrete state rarely changes over time.

The cost in (4.8) is alternatively minimized with respect to the model parameters Θ (for a fixed state sequence $\mathcal{S}$), and with respect to the discrete-state sequence $\mathcal{S}$ (for fixed model parameters Θ).
On the one hand, if the state sequence $\mathcal{S}$ is fixed, the minimization of (4.8) over the parameters Θ is a convex problem provided that ℓ and r are convex functions. On the other hand, when the parameters Θ are fixed, the minimization of (4.8) over the state sequence $\mathcal{S}$ is performed via dynamic programming (DP), as described next.

Define $L(i,t) \triangleq \ell(X_t, y_t, \theta_i)$ and $R(i,j) = \lambda \mathbf{1}_{[i\neq j]}$, with $i, j \in \mathcal{K}$. The cost to be minimized over $s(t)$ can be written in the compact form

$$\sum_{t=1}^{T-1} [L(s(t), t) + R(s(t), s(t+1))] + L(s(T), T), \tag{4.10}$$

which corresponds to the original cost function (4.8) without regularization on θ.

Minimizing (4.10) over $\mathcal{S}$ amounts at finding the shortest path through a graph $\mathcal{G}$ with KT vertexes. The edges connecting the vertexes (i,t) and $(j,t+1)$ (with $i, j \in \mathcal{K}$, $t \in 1, \ldots, T-1$) represent the time transition from state i to state j. The cost to cross vertex (i,t), which weights the condition $s(t) = i$, is given by $L(i,t)$, while the cost associated to the transition from state i to state j is given by $R(i,j)$. In other words, the weight associated to the edge connecting the vertexes (i,t) and $(j,t+1)$

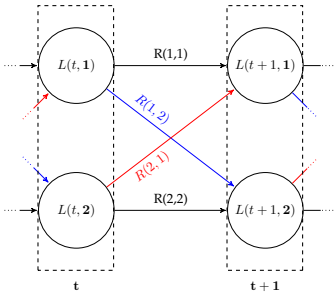


Figure 28: Vertexes corresponding to the consecutive instants t and $t+1$ of the graph over which the shortest path $\mathcal{S} = \{s(t)\}_{t=1}^{T}$ is computed, for $K = 2$.

is λ if $i \neq j$, 0 otherwise. The described graph is represented in Figure 28 for the case $K = 2$ and two consecutive time instants t and $t+1$.

The shortest path in the graph $\mathcal{G}$ is thus computed through dynamic programming as follows. Define $V(k,T) = L(k,T)$, $k \in \mathcal{K}$, and

$$V(i,t) = L(i,t) + \min_{j \in \mathcal{K}} (V(i,t+1) + R(i,j)), \quad i = 1,\dots,K, \quad t = T-1, T, \dots, 1 \tag{4.11}$$

with $V(i,t)$ representing the cost to be at state i at time t, going backward from time T by following the shortest path. The shortest path for the whole graph is then given by

$$s(t) = \underset{k \in \mathcal{K}}{\operatorname{argmin}} \{V(k,t) + R(s(t-1),k)\}, \quad t = 1,\dots,T. \tag{4.12}$$

The computation of the state sequence $\mathcal{S}$ thus requires $O(TK^2)$ operations (see Eq. (4.11)), not counting the evaluation of $L(k,t)$ at each instant $t = 1,\dots,T$ and for each $k = 1,\dots,K$, which is likely to dominate the computations.

The alternating minimization strategy described above is summarized in Algorithm 8, where the computation of $\mathcal{S}$ via DP is carried out at Steps 2- 4. The quality of the final estimates Θ and $\mathcal{S}$ may be improved by repeating Algorithm 8 iteratively, using the previously computed state sequence $\mathcal{S}$ as the initial condition for the following execution, until a

maximum number of iterations N_{max} is reached or when the solutions of two consecutive runs remain the same.

Remark 3 *Once the sub-model parameters in Θ have been estimated, if a new set of data $\{X_t, y_t\}_{t=1}^{T_\mathrm{v}}$ is available the underlying sequence of discrete states $\{s(t)\}_{t=1}^{T_\mathrm{v}}$ can be retrieved by minimizing the following cost*

$$\sum_{t=1}^{T_\mathrm{v}-1} \left[L_y(s(t),t) + R(s(t),s(t+1))\right] + L_y(s(T_\mathrm{v}),T_\mathrm{v}) \tag{4.13}$$

via DP, with

$$L_y(k,t) = \min_{y\in\mathcal{Y}} \ell\left(X_t, y, \theta_k\right), \quad k = 1,\ldots,K, \tag{4.14}$$

with $y \in \mathcal{Y}$ (e.g., in binary classification $\mathcal{Y} = \{-1, 1\}$). Note that the minimizer of $L_y(k,t)$ provides the "best" output predictor of the k-th model. ■

Algorithm 8 RJM learning algorithm

Input: Sequence of observations $\{X_t, y_t\}_{t=1}^{T}$, number of local sub-models K, initial state sequence $\mathcal{S}_\mathrm{o} = \{s(t)\}_{t=1}^{T}$.

---

1. **find** $\Theta = \{\theta_k\}_{k=1}^{K}$ minimizing (4.8) given $\mathcal{S}$;
2. **let** $V(k,T) \leftarrow L(k,T)$, $k = 1,\ldots,K$;
3. **for** $t = T,\ldots,2$ **do**
   - 3.1. **for** $k = 1,\ldots,K$ **do**
     - 3.1.1. $V(i,t-1) \leftarrow L(i,t-1) + \min_{j\in\mathcal{K}}(V(j,t) + R(i,j))$;
4. **for** $t = 1,\ldots,T$ **do**
   - 4.1. $s(t) \leftarrow \operatorname{argmin}_{k\in\mathcal{K}} \{V(k,t) + R(s(t-1),k)\}$;
5. **end**.

Output: Estimated parameters Θ and sequence $\mathcal{S}$ of active modes.

4.2.2 $\mathcal{M}_2$. Learning Markov Jump Models

In this section, we tackle the problem of learning Markov Jump Models, extending the approach developed in [25, 28] for piecewise-affine regression and identification of discrete hybrid automata. We propose to learn MJMs in two steps:

S1. Each data pair (X_t, y_t) is processed iteratively and assigned to the local model that most likely has generated it. Simultaneously, the parameters of the associated sub-model are updated via recursive least-squares. The outputs of this step are (i) the sub-model parameters $\{\theta_k\}_{k=1}^K$, (ii) the initial estimates of the discrete-state sequence $\mathcal{S}$ and (iii) an initial approximation for the transition matrix Π of the Markov Chain driving discrete-state transitions. An approach to identify Markov Jump Linear Systems relying on this scheme is also presented in [36].

S2. The state sequence $\mathcal{S}$ is estimated again by exploiting the probabilistic information on the state transition embedded in Π.

When working off-line, the hidden Markov Chain governing the mode transitions is assumed to be homogeneous, *i.e.*, the transition probabilities are supposed to be constant over time. Nonetheless, by designing the approach so that it can be used on-line, we can also account for time varying transition probabilities.

Before providing a detailed description of steps $S1$ and $S2$, we remark that both of them can be carried out either in a batch mode (off-line) or recursively (on-line).

S1. Recursive sub-model assignment and parameter estimation

The first stage S1 of the proposed MJM learning algorithm is detailed in Algorithm 9, which is based on the ideas presented in [28]. The rationale behind Algorithm 9 is to process the data pairs (X_t, y_t) iteratively and compute, at each time instant t, the estimation error $\epsilon_k(t)$ (Step 1.1) for all the possible models, $k = 1, \ldots, K$, using the parameters estimated at the previous step $t - 1$. The current data pair (X_t, y_t) is then assigned

to the model that "best fits" the observation at time t (Step 1.2), thus updating the estimated state sequence $\mathcal{S}$. Then, only the parameter vector $\theta_{s(t)}$ associated to the selected model is updated (Step 1.3) using the numerically efficient recursive least-squares method based on inverse QR factorization of [2], while the parameters of the other sub-models are not modified. An approximation of the transition matrix Π is computed iteratively (Step 1.4), based on the empirical jumping frequencies.

Algorithm 9 Recursive clustering and parameter estimation algorithm

Input: Sequence of observations $\{X_t, y_t\}_{t=1}^T$, number of local submodels K, initial guess for the parameter vectors $\{\theta_k\}_{k=1}^K$.

1. **for** $t = 1, \ldots, T$ **do**
   - 1.1. **let** $\epsilon_k(t) \leftarrow y_t - \theta_k' X_t$, $k = 1, \ldots, K$;
   - 1.2. **let** $s(t) \leftarrow \underset{k \in \mathcal{K}}{\operatorname{argmin}}\, \epsilon_k'(t)\epsilon_k(t)$;
   - 1.3. **update** $\theta_{s(t)}$ using recursive least-squares [2];
   - 1.4. **if** $t > 1$ **do**
     - 1.4.1. $N_i(t) \leftarrow \sum_{\tau=1}^{t} \mathbf{1}\left(s(\tau) = i\right)$, $i = 1, \ldots, K$;
     - 1.4.2. $N_{ij}(t) \leftarrow \sum_{\tau=2}^{t} \mathbf{1}\left(s(\tau-1) = i, s(\tau) = j\right)$, $i, j = 1, \ldots, K$;
     - 1.4.3. $\pi_{ij}(t) \leftarrow \dfrac{N_{ij}(t)}{N_i(t-1)}$, $i, j = 1, \ldots, K$;
2. $\Pi_{ij} \leftarrow \pi_{ij}(T)$, $i, j = 1, \ldots, K$;
3. **end**.

Output: Estimated parameter vectors Θ, sample transition matrix Π, sequence of active modes $\mathcal{S} = \{s(t)\}_{t=1}^T$.

As the other algorithms for recursive clustering and parameter estimation, Algorithm 9 requires an initial guess for the matrix Θ of model parameters. Because of the greedy nature of the algorithm, the selected

initial guess influences the final estimate of both Θ and the state sequence $\mathcal{S}$. If Algorithm 9 is executed on a batch of data, a reasonable initial guess is given by solving the least squares problem

$$\theta_k \equiv \arg\min_{\theta} \sum_{t=1}^{T} \|y_t - \theta' X_t\|_2^2, \ \forall k = 1, \dots, K. \tag{4.15}$$

As for RJM learning, the quality of the estimated MJM may be further improved by executing Algorithm 9 multiple times, using its output as initial condition for the next iteration.

S2. Refinement of the discrete-state sequence estimate

Algorithm 9 does not rely on the hypothesis that the discrete state $s(t)$ evolves according to a Markov chain. To exploit this information, at stage S2, the discrete-state sequence $\{s(t)\}_{t=1}^{T}$ is refined iteratively by using the probabilistic information on the state transition embedded in the sampled transition matrix Π obtained with Algorithm 9, as described below.

Let $\mathcal{I}^t$ be the *information vector* containing the regressor/output pairs until time t, and let $\{\alpha_k(t|t)\}_{k=1}^{K}$ be the probability of being at state k at time t given $\mathcal{I}^t$, *i.e.*,

$$\alpha_k(t|t) = P(s(t) = k|\mathcal{I}^t) \quad k = 1, \dots, K.$$

We have that

$$\begin{aligned}\alpha_k(t|t) =& P(s(t) = k|\mathcal{I}^{t-1}, X_t, y_t) = \frac{P\left(y_t, s(t) = k|\mathcal{I}^{t-1}, X_t\right)}{P\left(y_t|\mathcal{I}^{t-1}, X_t\right)} = \\ =& \frac{P\left(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t\right) P\left(s(t) = k|\mathcal{I}^{t-1}, X_t\right)}{P\left(y_t|\mathcal{I}^{t-1}, X_t\right)}.\end{aligned} \tag{4.16}$$

Let $\alpha_k(t|t-1) = P(s(t) = k|\mathcal{I}^{t-1}, X_t) = p(s(t) = k|\mathcal{I}^{t-1})$ be the probability of being at state k at time t, given the observations up to time $t-1$. By the total probability theorem we have that

$$\alpha_k(t|t-1) = \sum_{i=1}^{K} p(s(t) = k|s(t-1) = i, \mathcal{I}^{t-1}) p(s(t-1) = i|\mathcal{I}^{t-1}). \tag{4.17}$$

Note that $p(s(t) = k|s(t-1) = i, \mathcal{I}^{t-1})$ corresponds to the (i,k) entry $\pi_{i,k}$ of the transition matrix Π.

Using equation (4.17), the probabilities in (4.16) can be rewritten as

$$\alpha_k(t|t) = \frac{p(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t)\alpha_k(t|t-1)}{c}, \tag{4.18}$$

where

$$c = p(y_t|\mathcal{I}^{t-1}, X_t) = \sum_{j=1}^{K} p(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t)\alpha_k(t|t-1)$$

is a normalization constant.

Under the assumption that the noise e_t corrupting the output y_t (see equation (4.1)) is white, zero-mean and Gaussian, the likelihood function of mode k at time t $p(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t)$, is given by:

$$p(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t) = \frac{1}{\sqrt{(2\pi)^{n_y} \det(\Lambda)}} \exp\left\{-0.5\|y_t - \theta_k' X_t\|^2_{\Lambda^{-1}}\right\} \tag{4.19}$$

where Λ is the covariance matrix of the error e_t. In case the refinement is performed in a batch mode, a possible choice for Λ is the sample covariance matrix obtained on the basis of the state sequence $\mathcal{S}$ and the model parameters $\{\theta\}_{k=1}^{K}$ estimated in $S1$, *i.e.*,

$$\Lambda = \frac{1}{T-1} \sum_{t=1}^{T} \left(y_t - \theta_{s(t)}' X_t\right)' \left(y_t - \theta_{s(t)}' X_t\right). \tag{4.20}$$

When the proposed method is used for on-line learning, the inverse of Λ is updated at each time step, using the iterative formulation for (4.20) and the matrix inversion lemma

$$Q = \Lambda^{-1} - \frac{\Lambda^{-1}\left(y_t - \theta_{s(t)}' X_t\right)\left(y_t - \theta_{s(t)}' X_t\right)'\Lambda^{-1}}{t - 2 + \left(y_t - \theta_{s(t)}' X_t\right)'\Lambda^{-1}\left(y_t - \theta_{s(t)}' X_t\right)}$$

$$\Lambda^{-1} = \frac{t-1}{t-2}\left(Q - \frac{Q\delta_e\delta_e' Q}{\frac{t-2}{t-1} + \delta_e' Q \delta_e}\right),$$

Algorithm 10 Probabilistic clustering

Input: Sequence of observations $\{X_t, y_t\}_{t=1}^T$, number of local submodels K, parameters $\Theta = \{\theta_k\}_{k=1}^K$, error covariance matrix Λ, initial probabilities $\{\alpha_k(1|0)\}_{k=1}^K$, sample transition matrix Π.

1. **for** $t = 1, \ldots, T$ **do**
   1.1. **compute** $p(y_t|s(t) = k, \mathcal{I}^{t-1}, X_t)$, $k = 1, \ldots, K$, as in (4.19);
   1.2. **compute** $\alpha_k(t|t)$, $k = 1, \ldots, K$, as in (4.18);
   1.3. **let** $s(t) \leftarrow \underset{k=1,\ldots,K}{\operatorname{argmax}}\, \alpha_k(t|t)$;
   1.4. **if** $t > 1$ **do**
      1.4.1. $N_i(t) \leftarrow \sum_{\tau=1}^{t} \mathbf{1}\left(s(\tau) = i\right)$, $i = 1, \ldots, K$;
      1.4.2. $N_{ij}(t) \leftarrow \sum_{\tau=2}^{t} \mathbf{1}\left(s(\tau-1) = i, s(\tau) = j\right)$, $i, j = 1, \ldots, K$;
      1.4.3. $\pi_{ij}(t) \leftarrow \dfrac{N_{ij}(t)}{N_i(t-1)}$, $i, j = 1, \ldots, K$;
   1.5. **update** $\alpha_k(t+1|t)$, $k = 1, \ldots, K$ as in (4.17);
2. **end**.

Output: Estimated sequence of states $\mathcal{S} = \{s(t)\}_{t=1}^T$.

where, at time t, $\delta_e = \frac{1}{t}\left(y_t - \theta'_{s(t)} X_t\right)$.

The iterative procedure to refine the estimated state sequence $\mathcal{S}$ is summarized in Algorithm 10. Note that, it requires the initialization of $\alpha_k(1|0)$, $k = 1, \ldots, K$, which represent the probabilities of each possible initial state $s(1)$ (*e.g.*, $\alpha_k(1|0) = P(s(1) = k)$). If no information on the probability of the initial state is available, $P(s(1) = k)$ is set to $\dfrac{1}{K}$ for all $k \in \mathcal{K}$.

As Algorithm 10 is designed to estimate $\mathcal{S}$, it can be also used to improve the estimate of the sampled transition matrix Π.

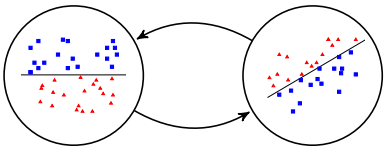


Figure 29: Two-state binary RJM classifier.

Remark 4 *Once the transition matrix* Π *is estimated by Algorithm 9 and refined by Algorithm 10, if a new set* $\{X_t, y_t\}_{t=1}^{T_v}$ *of data is given the computation of the* Maximum A Posteriori *(MAP) estimate of the hidden discrete-state sequence*

$$\{s^\star(t)\}_{t=1}^{T_v} = \underset{\{\sigma_t \in \mathcal{K}\}_{t=1}^{T_v}}{\operatorname{argmax}} P(s(1) = \sigma_1, \ldots, s(T) = \sigma_{T_v} | \mathcal{I}^{T_v}), \tag{4.21}$$

reduces to a well-known problem in the Hidden Markov Model literature and it can be solved via Viterbi algorithm [92, 93]. ■

4.3 Examples

The effectiveness of the proposed learning algorithms in evaluated on a set of examples, using both synthetic and experimental data. All computations are carried out on an i7 2.80-GHz Intel core processor with 16 GB of RAM running MATLAB R2016b. The reported examples can be replicated using the MATLAB source codes available in [30].

4.3.1 Learning Rarely Jump Models

Time-varying linear classifier

We start with a toy numerical example, in which a binary-label dataset is generated by the two-state RJM depicted in Figure 29. The training dataset consists of $T = 400$ two-dimensional samples and is plotted in Figure 30.

The aim is to estimate a binary linear classifier. Figure 30 clearly shows that the two classes are not linearly separable. However, as shown in Figure 4.31(a), the first 262 data points are linearly separable, as well

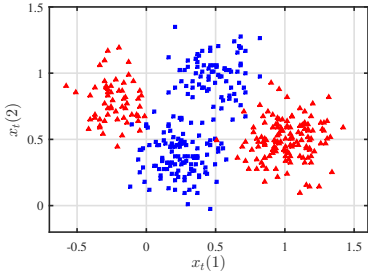
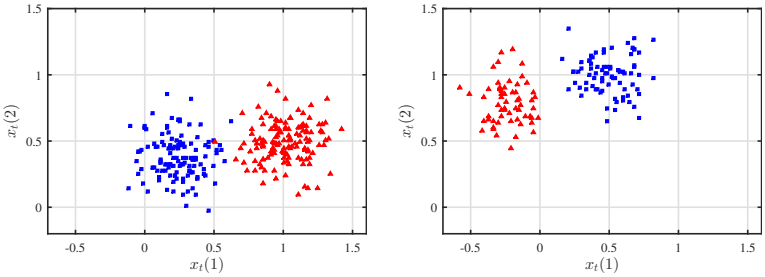


Figure 30: RJM Classifier: training dataset. Blue: $y_t = 1$; red: $y_t = -1$.



(a) $\{X_t\}_{t=1}^{262}$ (b) $\{X_t\}_{t=263}^{400}$

Figure 31: Rarely Jump Classifier: temporal pattern in the training dataset. Label $y_t = 1$ (blue); label $y_t = -1$ (red).

as the remaining samples (see Figure 4.31(b)). Thus, if the samples are properly split into $K = 2$ subsets, a linear classifier for each subset can thus be computed. Moreover, it is apparent that jumps between the possible states are quite rare (a transition happens only once in the given dataset), which motivates the use of the RJM modeling framework.

A Rarely Jump Model with $K = 2$ discrete states is estimated. Although in this example the underlying discrete-state sequence $\{s(t)\}_{t=1}^{T}$ generating the training set can be easily inferred by looking at the data flow, this information is *not* provided as an input to Algorithm 8 but the discrete-state sequence is estimated via DP (Steps 2-4 of Algorithm 8).

Due to the nature of the problem, we consider the following loss function

$$\ell(X_t, y_t, \theta_k) = \max\{0, 1 - y_t \theta_k' X_t\}, \tag{4.22}$$

with $y_t = \pm 1$, along with the ℓ_2-regularization term

$$r(\theta_k) = \|\theta_k\|_2, \quad k = 1, 2. \tag{4.23}$$

Unlike the original cost in (4.8), time-varying weights $\lambda(s(t))$ on the state transitions are considered, so that the term $R(i, j)$ in (4.10) is equal to $\lambda(s(t))\mathbf{1}_{[s(t+1)\neq s(t)]}$.

The optimization problems characterizing Step 1 of Algorithm 8 are solved using the CVX package for MATLAB [52, 53].

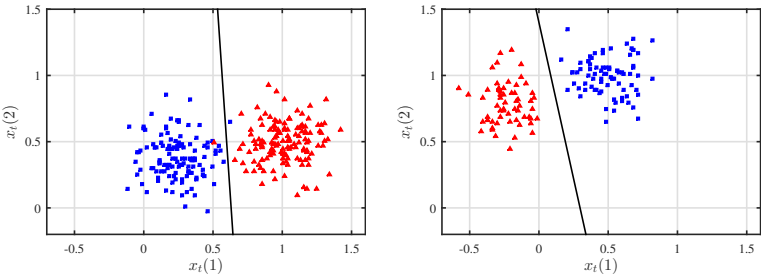
As Algorithm 8 is iterated multiple times, the hyper-parameter λ_k is set to a constant λ_o (for $k = 1, 2$) for the first $n_{min} = 4$ iterations and then it is replaced with the empirical probability of permanence in the k-th mode from time t to time $t+1$. The hyper-parameter γ and λ_o are chosen through cross-validation, and they are set to 5 and 0.99, respectively.

Algorithm 8 is iterated until the sequence $\{s(t)\}_{t=1}^{T}$ remains equal over two consecutive runs or the maximum number of runs $N_{max} = 30$ is reached. The state sequence is initialized by solving the K-model fitting problem (*i.e.*, $\lambda_k = 0$, $k = 1, 2$ in (4.8)) for different randomly generated sequences. Among the latter, the mode sequence achieving the lowest cost is used as initial value of $\{s(t)\}_{t=1}^{T}$. The $K = 2$ binary linear separators computed by Algorithm 8 are reported in Figure 32, while Figure 4.32(a) and Figure 4.32(b) show the samples associated by Algorithm 8 to the discrete state $s(t) = 1$ and $s(t) = 2$, respectively. The obtained results show that we are able to obtain two linear classifiers properly separating the two classes for both the operating modes.

The performance of the estimated RJM classifier is assessed on a validation dataset of $T_v = 200$ samples. The underlying sequence of discrete states is retrieved by DP as outlined in Remark 3. Note the labels y_t associated to the data points are not available for learning and they are only used to compare estimated and actual labels.

We use two indexes to evaluate the quality of the estimated classifier:

- *Percentage of Mislabeled Points* (*PMP*), representing the percentage of misclassified points (points labeled as 1 when the actual label



(a) Binary linear separator: points associated to the discrete state $s(t) = 1$.

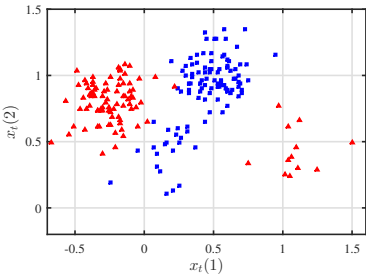
(b) Binary linear separator: points associated to the discrete state $s(t) = 2$.

Figure 32: RJM classifier: binary jumping linear separator trained through Algorithm 8.

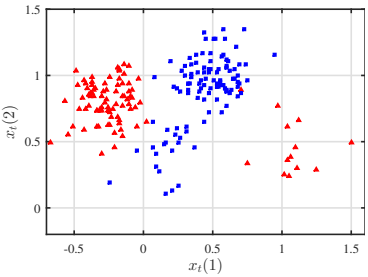
is -1, or viceversa). The obtained value of the PMP index is 1.5% and it is equal to the PMP index achieved by a Support Vector Machine (SVM) classifier with quadratic kernels, trained using MATLAB's Statistic and Machine Learning Toolbox [59]. The quadratic kernel is chosen as it is the one providing the least number of misclassified points in validation. Figure 4.33(a), Figure 4.33(b) and Figure 4.33(c) show the actual labels of the points in the validation dataset and the estimated labels with our approach and SVM, respectively. Note that, even though the estimated RJM classifier and the SVM classifier achieve the same performance in terms of PMP, the SVM classifier cannot account for the temporal pattern of the data.

- *Percentage of Mismatched Modes* (*PMM*), representing the percentage of points assigned to the wrong discrete state (points assigned to mode $s(t) = 1$ when the actual generating state is $s(t) = 2$, or viceversa). The obtained value of PMM is 2.5%. Figure 34 shows the actual and estimated discrete states for each sample, highlighting the capability of the approach to retrieve the hidden sequence of active modes.

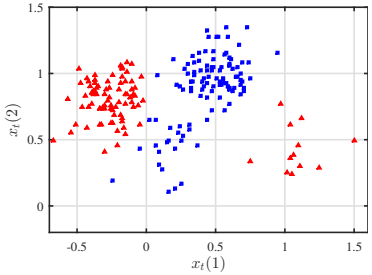
Figure 35 plots the minimum of the cost function in (4.8) as a function of the number of runs of Algorithm 8, showing that it practically



(a) True labels: blue=$y_t = 1$; red= $y_t = -1$.

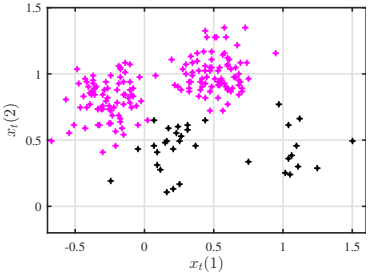


(b) Estimated labels (jumping classifier): blue=$y_t = 1$; red= $y_t = -1$.

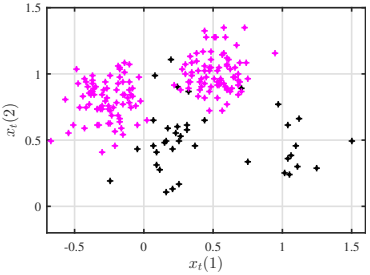


(c) Estimated labels (SVM): blue=$y_t = 1$; red= $y_t = -1$.

Figure 33: Validation set: true vs estimated labels.



(a) True mode: black=$s(t) = 1$, magenta=$s(t) = 2$.



(b) Estimated mode: black=$s(t) = 1$, magenta=$s(t) = 2$.

Figure 34: Rarely Jump Classifier. Validation set: true vs estimated discrete-state sequence $\{s(t)\}_{t=1}^{T_{\text{v}}}$.

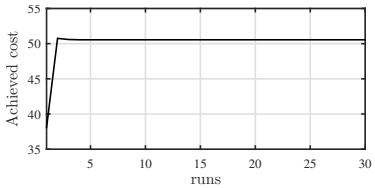


Figure 35: Rarely Jump Classifier. Achieved minimum cost (4.8) vs runs of Algorithm 8.

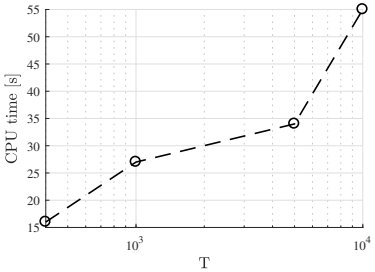


Figure 36: Rarely Jump Classifier. CPU time required to estimate the classifier vs length T of the training set.

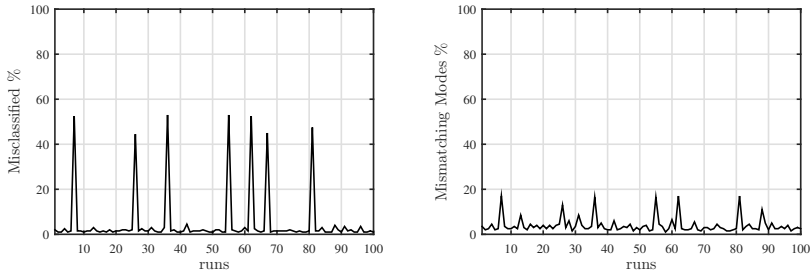
converges after 4 iterations.

To analyze the computation complexity of the algorithm, training sets of different lengths are used. The total CPU time required to estimate the Rarely Jump Model, performing 15 iterations, is reported in Figure 36. Note that, when $T = 10000$ instead of $T = 400$ is considered, the CPU time required to train the classifier is 3.4 times bigger.

Finally, a Monte Carlo study with 100 runs, with new realizations of the training dataset at each run, is performed to assess the robustness of the learning method. Training sets of length $T = 1000$ are used, and the performance of the estimated model is assessed on the validation dataset already presented in Figure 4.33(a) and Figure 4.34(a). The performance indexes PMP and PMM obtained in each simulation are reported in Figure 37, while their mean and standard deviations over the Monte Carlo simulation are given in Table 12. The results in Figure 4.37(a) show that in most of the considered cases less then 10 points (5% of the validation set) are mislabeled. However, in 7 simulations more then 40% of the data are not correctly classified. In all these simulation (except one)

Table 12: Rarely Jump Classifier. Monte Carlo study: Percentage of Mislabeled Points (PMP) and Percentage of Mismatched Modes (PMM) (mean $\pm$ std) %.

PMP	$(4.98 \pm 12.39)\%$
PMM	$(3.97 \pm 3.48)\%$



(a) PMP index vs Monte Carlo runs. (b) PMM index vs Monte Carlo runs.

Figure 37: Rarely Jump Classifier. Monte Carlo simulation.

more than 12% of the points are associated with the wrong discrete state (see Figure 4.37(b)), showing the importance of accurately reconstructing the underlying state sequence. By changing the sequences of discrete states used to initialize Algorithm 8, the PMP index for the 7 trials showing more than 40% of misclassified points varies in the interval between 1% and 4%, while the PPM index is always less than 4%. This shows that the performance of learning algorithm can be further improved by initializing Algorithm 8 with different state sequences $\mathcal{S}$.

Electric power consumption

Consider experimental data taken from the AMPDs dataset [74], which consists of the electric power consumption readings of different appliances taken every minute in a single house located in Canada from April 1st 2012 to March 31st 2013. In particular, among the available measures, we consider power reading collected from the clothes dryer (CDE), the dishwasher (DWE), the kitchen fridge (FGE), and the heat pump (HPE)[1].

[1]The cloths dryer, dishwasher, kitchen fridge and heat pump are labeled as, $i = 1, 2, 3, 4$, respectively.

The goal is to train a RJM for each appliance i and then use the estimated RJMs to figure out the appliance i that consumes electricity from measurements of total power consumption.

The training set consists of one week of power consumption records ($T = 10080$ samples) for each appliance. To have informative enough data sets for each appliance, we consider different weeks: from day 74 to day 80 form the training set for CDE, from day 170 to day 176 for DWE, from day 23 to day 29 for FGE and HPE.

Looking at the available data, it can be noticed that the appliances change their operating mode rarely over time, thus motivating the use of RJM to model their consumption behavior. For each appliance, the power consumption pattern is described by an RJM with K_i static submodels

$$y_t^i = \theta^i_{s_i(t)} \;\; s_i(t) \in \{1, \dots, K_i\}, \tag{4.24}$$

with $\theta^i_k \in \mathbb{R}$, for $i = 1, \dots, 4$. Each parameter $\theta^i_{s_i(t)}$ represents the power consumed by the i-th appliance at the operating condition $s_i(t)$. Cross-validation is used to choose the number of modes K_i for each appliance, that results in $K_1 = K_2 = 3$ (clothes dryer and dishwasher) and $K_3 = K_4 = 2$ (fridge and heat pump).

In the cost function (4.8) minimized by Algorithm 8, the squared Euclidean norm of the fitting error

$$\ell(y_t, \theta_k) = (y_t - \theta_k)^2 \tag{4.25}$$

is used as a loss function and no regularization is considered. Time-varying weights $\lambda(s(t))$ on the state transitions are used, leading to the following cost

$$J = \frac{1}{T}\left(\sum_{t=1}^{T-1} (y_t - \theta_k)^2 + \lambda(s(t))\mathbf{1}_{[s(t+1)\neq s(t)]} + (y_T - \theta_k)^2\right). \tag{4.26}$$

As in the example of Section 4.3.1, the hyper-parameter λ_k is set to a constant $\lambda_o = 0.9$ (for $k = 1, \dots, K_i$) for the first $n_{min} = 4$ iterations of Algorithm 8, and then replaced with the empirical probability of permanence in the k-th mode from time t to time $t+1$. The parameters $\{\theta^i_k\}_{k=1}^{K_i}$

Table 13: Rarely Jump Models learning: experimental example. Estimated parameters for each appliance.

#	Appliance	θ_1	θ_2	θ_3
1	CDE	251.8	0.4	4644.3
2	DWE	0.1	781.2	146.6
3	FGE	1.1	132.8	–
4	HPE	33.7	1822.1	–

are estimated by computing analytically the minimum of (4.26) for each $i = 1, 2, 3, 4$. Algorithm 8 is iterated until the state sequence $\{s_i(t)\}_{t=1}^{T}$ remains equal over two consecutive runs or the maximum number of runs $N_{max} = 30$ is reached. The state sequence is initialized by solving the K-model fitting problem (*i.e.*, $\lambda_k = 0$, $k = 1, 2$ in (4.26)) for 10 different randomly generated sequences. The mode sequence achieving the lowest cost is used as the initial value of $\{s_i(t)\}_{t=1}^{T}$.
The obtained values of θ_k^i for each appliance are reported in Table 13 and, because of the chosen model structure (4.24), each parameter θ_k^i represents the estimated power consumption (in Watt) of the i-th appliance at the operating mode k. The average CPU time required to train a three-state model (cloth dryer, dishwasher) is 1.2 s, while the time to train the two-state model (fridge, heat pump) is 1.1 s.

The performance of the four estimated Rarely Jump Models is assessed on the following one-day ($T_{\text{v}} = 1440$ samples) validation sets: $\mathcal{V}_1$ = day 170 for CDE, $\mathcal{V}_2$ = day 45 for DWE, $\mathcal{V}_3$=day 234 for FGE, and $\mathcal{V}_4$=day 80 for HPE. The quality of the estimated models is quantified in terms of the *Best Fit Rate* (BFR)

$$\text{BFR} = 100 \max\left\{1 - \frac{\sqrt{\sum_{t=1}^{T_{\text{v}}} (y_t - \hat{y}_t)^2}}{\sqrt{\sum_{t=1}^{T_{\text{v}}} (y_t - \bar{y})^2}}, 0\right\} \%, \tag{4.27}$$

with $\bar{y}$ denoting the sample mean of the output and $\hat{y}_t = \theta_{s(t)}$ being the output predicted by the model.

The outputs of the four estimated models are reported in Figure 38, along with the actual output in the corresponding validation sets. Only the results obtained for 240 samples are plotted to simplify visualization. The reported estimated sequences $\{s_i(t)\}_{t=1}^{T_{\text{v}}}$ of active modes show that

Table 14: Rarely Jump Models learning: experimental example. Computed BFR for the four estimated models of the appliances on the validation datasets $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4$.

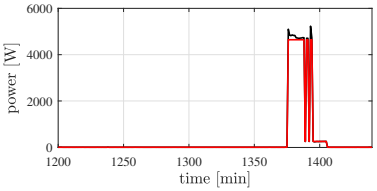
Set \ Model	CDE	DWE	FGE	HPE
$\mathcal{V}_1$	95.48%	15.76%	2.20%	37.08%
$\mathcal{V}_2$	30.53%	98.04%	15.19%	0%
$\mathcal{V}_3$	0%	81.81%	91.44%	0%
$\mathcal{V}_4$	10.02%	40.06%	3.25%	91.28%

the behavior of each appliance is accurately described by the corresponding estimated RJM.

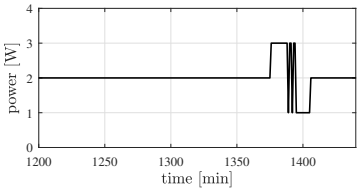
The BFRs computed with respect to the validation datasets $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, $\mathcal{V}_4$ are reported in Table 14. Observe that the BFRs reported in the diagonal entries of the table are always higher than 90%, and they are the highest one of the corresponding row. This means that the estimated models can automatically detect which of the considered four appliances has generated a given power consumption pattern. The BFR achieved by the model of the dishwasher (DWE) is relatively high (81.8%) on the validation set $\mathcal{V}_3$, which instead consists of the sequence of power consumption readings of the fridge (FGE). This is mainly due to the fact that the consumption patterns of the two appliances are quite similar, and thus barely distinguishable from each other. This can be also deducted looking at the estimated parameters θ^i_k for the DWE and the FGE model (see Table 13).

A different validation procedure is carried out to further assess the performance of the four estimated RJMs in discriminating power consumption from concatenated sequences of readings of multiple appliances. The new validation set $\mathcal{V}_5$ contains 5760 power consumption records obtained by stacking sequentially the measurements of CDE at day 40 and of DWE, FGE and HPE at day 60. The validation set $\mathcal{V}_5$ is then split into 96 non-overlapping windows of 60 samples. For each of the four estimated models and each windows the hidden sequence of the operating condition is retrieved and the corresponding BFR computed.

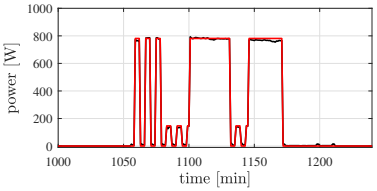
Figure 39 shows which of the four appliances is actually consuming at each time sample, along with the BFRs provided, for each time win-



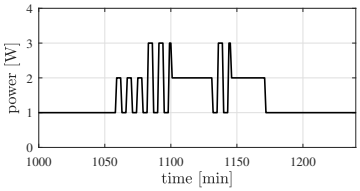
(a) Cloth dryer: output.



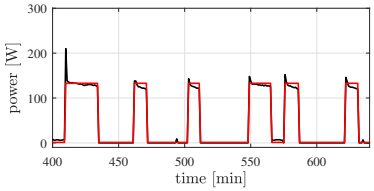
(b) Cloth dryer: discrete-state sequence.



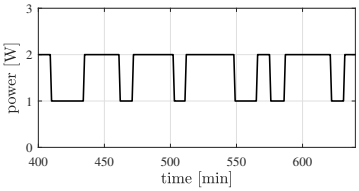
(c) Dishwasher: output.



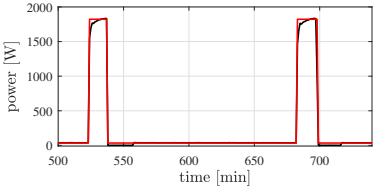
(d) Dishwasher: discrete-state sequence.



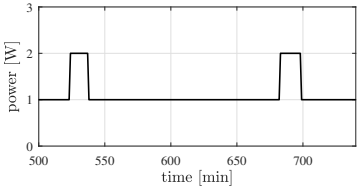
(e) Fridge: output.



(f) Fridge: discrete-state sequence.



(g) Heat pump: output.



(h) Heat pump: discrete-state sequence.

Figure 38: Rarely Jump Models learning: experimental example. Predicted output (red) vs actual output (black) (left plots) and estimated state sequence (right plots).

dow, by the four estimated models. These results show that the model associated to the device that is actually consuming provides the highest BFR. The only exception is the 59th window (see Figure 39 between sam-

ple 3481 and 3541), where the highest BFR is obtained with the model of the dishwasher, while the fridge is actually consuming power. Such an error is due to the fact that, as already pointed out $\theta_1^2 \approx \theta_1^3$ and $\theta_3^2 \approx \theta_2^3$. The models of the dishwasher and the fridge are thus similar, although the DWE model is characterized by an additional mode, corresponding to $\theta_2^2 = 781.2$ W, that is not present in the fridge (see Table 13). Looking at Figure 40 and focusing on the samples in the 59th window, one observes a spike (of about 700 W) in the power consumption readings. This kind of spikes are seldom observed in the fridge consumption pattern used for training, and thus they are not described by the associated model. On the other hand, the model of the dishwasher recognizes this spike as the power consumed in its third state, thus leading to a higher BFR than the one obtained with the model of the fridge. During certain time windows, the BFR is zero for all the estimated models. This corresponds to the case in which all the devices are off (*i.e.*, the power readings are approximately zero over the whole time window), and hence the active appliance is undetectable from power consumption measurements.

4.3.2 Learning Markov Jump Models

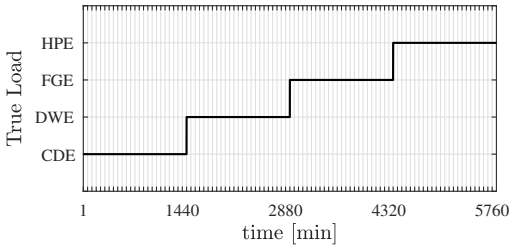
Markov jump linear system

We train a MJM from data generated by the Markov Jump Linear System (MJLS)

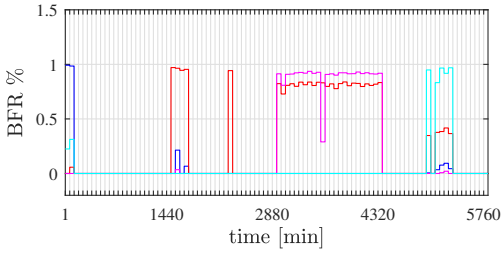
$$y_t = \begin{cases} 0.4y_{t-1} + 0.7u_{t-1} + e_t & \text{if } s(t) = 1 \\ -0.2y_{t-1} + 0.4u_{t-1} + e_t & \text{if } s(t) = 2 \\ 0.7y_{t-1} + 0.5u_{t-1} + e_t & \text{if } s(t) = 3, \end{cases} \tag{4.28}$$

where $u_t \in \mathbb{R}$ is a white sequence of i.i.d. samples with standard normal distribution and e_t is a white noise, with $e_t \sim \mathcal{N}(0, 0.01)$. The effect of noise on the training dataset is quantified by the *Signal-to-Noise Ratio* (SNR)

$$\text{SNR} = 10 \log \frac{\sum_{t=1}^{T} (y_t - e_t)^2}{\sum_{t=1}^{T} e_t^2}. \tag{4.29}$$



(a) Sequence of active appliances



(b) BFRs provided by the 4 estimated models: CDE (blue), DWE (red), FGE (magenta), HPE (cyan).

Figure 39: Rarely Jump Models learning: experimental example. Validation set $\mathcal{V}_5$: active appliance and computed BFRs.

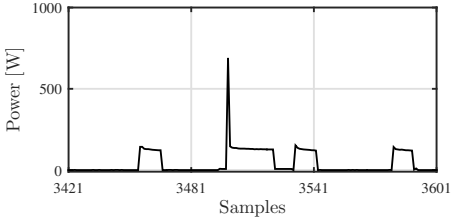


Figure 40: Rarely Jump Models learning: experimental example. Validation set $\mathcal{V}_5$: 58-th, 59-th and 60-th windows.

The Markov Chain characterizing the changes in the discrete state is described by the transition matrix

$$\Pi^{\circ} = \begin{bmatrix} 0.6 & 0.25 & 0.15 \\ 0.25 & 0.5 & 0.25 \\ 0.2 & 0.1 & 0.7 \end{bmatrix}, \tag{4.30}$$

and is depicted in Figure 41. Observe that, because of the relatively large off-diagonal terms in Π°, a RJM would not be an adequate model for the

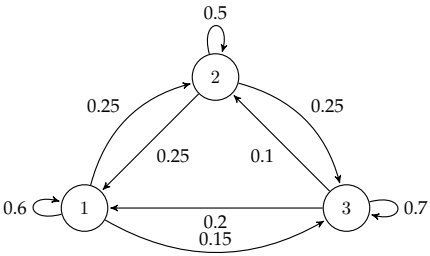


Figure 41: Markov chain governing the mode transitions of the MJLS, with transition probabilities associated with each edge.

Table 15: Markov Jump Linear Models: estimated parameters.

	θ_1		θ_2		θ_3	
true	0.40	0.70	-0.20	0.40	0.70	0.50
estimated	0.40	0.71	-0.20	0.39	0.71	0.49

considered system.

To learn the MJM model we generate numerically a training set of $T = 5000$ samples with SNR$= 16$ dB. We assume that the number of operating modes $K = 3$ and the order of the dynamical system $n_a = 1$ and $n_b = 1$ used to construct the regressor X_t as in (4.6) are known a priori. This assumption could be relaxed by selecting the parameters K, n_a, n_b via cross-validation.

Algorithm 9 is executed in batch mode by initializing the sub-model parameter $\{\theta_k\}_{k=1}^3$ with the best linear model as in (4.15), and iterating the algorithm 15 times, using its output as the initial condition for the next iteration. The estimated sub-models parameters $\{\theta_k\}_{k=1}^3$ are reported in Table 15 and compared to the parameters of the true system in (4.28). Finally, the transition matrix estimated with Algorithm 9 is

$$\Pi = \begin{bmatrix} 0.507 & 0.221 & 0.272 \\ 0.237 & 0.442 & 0.322 \\ 0.302 & 0.128 & 0.570 \end{bmatrix}.$$

To validate the quality of the identified Markov Jump Linear Model we consider a new validation dataset $\{u_t, y_t\}_{t=1}^{T_\text{v}}$, $T_\text{v} = 200$, whose outputs y_t are corrupted by e_t. The sequence of active modes $\{s(t)\}_{t=1}^{T_\text{v}}$ is

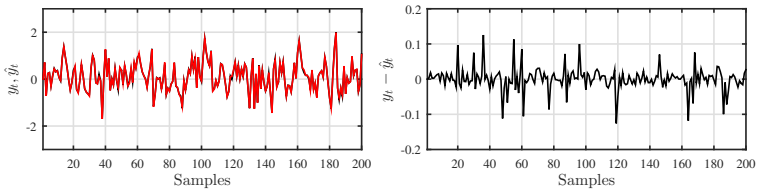


Figure 42: Markov Jump Linear Models: output (left panel): black = true, red = one-step ahead prediction; prediction error (right panel).

Table 16: Markov Jump Linear Models. Monte Carlo simulation, BFR (mean $\pm$ std) %.

BFR	$(90.37 \pm 5.92)\%$

reconstructed, as described in Remark 4, from $\{u_t, y_t\}_{t=1}^{T_v}$, the estimated state transition Π and the estimated model parameters $\{\theta_k\}_{k=1}^{3}$. This sequence is then used to estimate the one-step ahead predicted output $\hat{y}_t$. The actual and predicted output, *i.e.*, y_t, $\hat{y}_t$ are reported in Figure 42, along with the prediction error $y_t - \hat{y}_t$. Since the output samples y_t are available, predicting the output $\hat{y}_t$ has no practical value beyond assessing the quality of the identified model and of the estimated discrete-state sequence $\{s(t)\}_{t=1}^{T_v}$.

The BFR computed on the validation set is plotted in Figure 43 with respect to the iterations of Algorithm 9 and it shows that Algorithm 9 convergences after 2 iterations. Figure 44 quantifies the total CPU time required by the algorithm for training sets of different lengths. Note that large training sets can be efficiently handled by the proposed MJM approach with a computation time smaller than the one obtained running the RJM learning algorithm (see Figure 36).

The robustness of the developed learning method is assessed performing a Monte Carlo simulation over 50 new realizations of length $T = 5000$ of the training dataset. The resulting BFR is reported in Table 16, which shows mean and standard deviation over the 50 runs. Robustness is also analyzed w.r.t. the noise corrupting the data, by performing multiple experiments with different variances of the output noise e_t. See Table 17 for a comparison of the BFRs obtained for different SNRs.

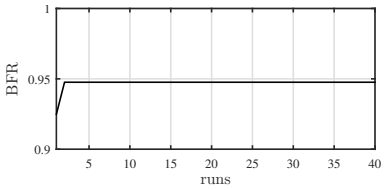


Figure 43: Markov Jump Linear Models. BFR vs number of iteration of Algorithm 9.

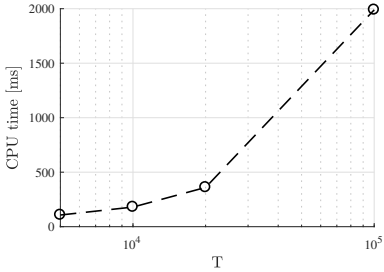


Figure 44: Markov Jump Linear Models. CPU time required to identify the MJLS in (4.28), with transition matrix as in (4.30), vs length T of the training set.

Finally, we evaluate the on-line learning capabilities of the proposed approach in tracking changes of both the sub-model parameters and the transition matrix. We consider a training set of length $T = 30000$, in which the first 6000 samples are generated by the MJLS in (4.28) with transition matrix Π in (4.30), while the remaining 24000 data points are generated by the Markov Jump Linear System

$$y_t = \begin{cases} 0.8y_{t-1} + 0.4u_{t-1} + e_t & \text{if } s(t) = 1 \\ 0.2y_{t-1} + 0.8u_{t-1} + e_t & \text{if } s(t) = 2 \\ -0.1y_{t-1} + 0.2u_{t-1} + e_t & \text{if } s(t) = 3 \end{cases} \tag{4.31}$$

with transition matrix

$$\Pi^o = \begin{bmatrix} 0.8 & 0.2 & 0 \\ 0 & 0.9 & 0.1 \\ 0.3 & 0.2 & 0.5 \end{bmatrix}. \tag{4.32}$$

The SNR on the training set is 16 dB.

The first 5500 samples are processed in batch mode to initialize the

Table 17: Markov Jump Linear Models. BFR and Percentage of Mismatched Modes (PMM) vs Signal-to-Noise Ratio (SNR).

SNR (dB)	30	22	16	13	8
BFR (%)	98.2	96.8	94.8	93.5	79.1
PMM (%)	4.5	10	15	16	28

parameters of the MJLM, while the other 24500 points are processed iteratively, to recursively update the estimated parameters and the transition probabilities. Considering a forgetting factor equal to 0.9975, the time-evolution of the estimated parameters θ_{1-3} is plotted in Figure 45, along with the true parameters. Note that the estimate of θ_3 converges to the actual value of the model parameters slower than the others, due to the fact that mode $\#3$ is activated less frequently than the other two discrete states.

The transition matrix, obtained after processing the last sample, is

$$\begin{bmatrix} 0.6609 & 0.2648 & 0.0744 \\ 0.0610 & 0.8294 & 0.1096 \\ 0.3214 & 0.2571 & 0.4215 \end{bmatrix}. \tag{4.33}$$

Electric power consumption

Algorithm 9 is run to learn a Markov Jump Model with static sub-models from the experimental data already used in Section 4.3.1. Cross-validation suggests taking $K_1 = K_3 = K_4 = 2$ and $K_2 = 3$, which do coincide with the ones used for learning a Rarely Jump Models. The sub-model parameters $\{\theta_k^i\}_{k=1}^{K_i}$ are computed using the best static model as in equation (4.15) to initialize Algorithm 9, which is then run 10 times. The estimates of θ_k for each appliance are reported in Table 18 and they represent the power consumption of each appliance at each operating mode.
The average CPU time to train the three-state model (*i.e.*, the dishwasher) is 198 ms, while the time to train a two-state model (*i.e.*, cloth dryer, fridge, and heat pump) is 170 ms. The CPU time required to learn the MJMs is thus 6x smaller than the one needed to train the Rarely Jump Models, probably because of the lighter initialization procedure of the MJM learning method.

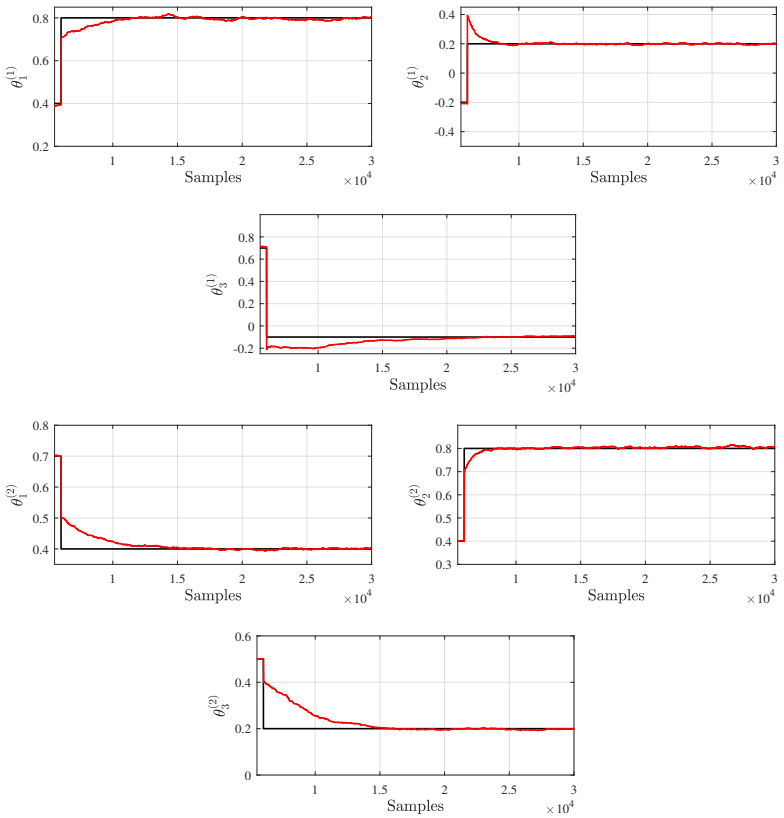


Figure 45: Markov Jump Linear Models, online learning. True (black) vs estimated (red) sub-model parameters θ_k ($\theta_j^{(i)}$ denotes the i-th component of θ_j).

The same analysis as in Section 4.3.1 is repeated over the same one-day validation sets $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$, $\mathcal{V}_4$, to assess the quality of the learned Markov Jump Models. The estimated and actual outputs, reported in Figure 46, show that the behavior of each of the four appliances is well captured by the corresponding estimated MJM. Unlike the RJM case, the power consumption level of 200 W of the cloth dryer (see Figure 4.46(a), around sample 1400) is not considered.
The BFRs of the 4 estimated MJMs over the four validation datasets and

Table 18: Markov Jump Models learning: experimental example. Estimated parameters for each for each appliance.

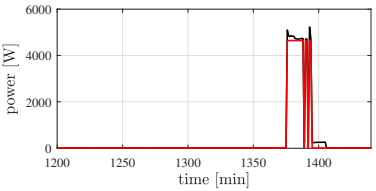
#	Appliance	θ_1	θ_2	θ_3
1	CDE	2.5	4644.3	–
2	DWE	0.1	781.2	146.6
3	FGE	1.1	132.8	–
4	HPE	33.7	1822.1	–

Table 19: Markov Jump Models learning: experimental example. BFR %.

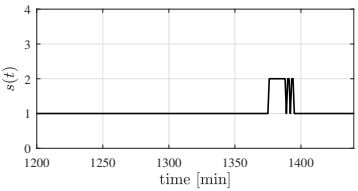
Set \ Model	CDE	DWE	FGE	HPE
$\mathcal{V}_1$	93.61 %	15.76 %	2.20%	37.08%
$\mathcal{V}_2$	0 %	98.04 %	15.19 %	0 %
$\mathcal{V}_3$	0 %	81.81 %	91.44 %	0 %
$\mathcal{V}_4$	0 %	40.06 %	3.25 %	91.28 %

reported in Table 19, showing that the BFRs in the diagonal entries of Table 19 are always higher than 90%, and they are the highest ones in the corresponding row also when Markov Jump Models are trained. Consequently, it is possible to figure out which device has generated a given consumption pattern from the obtained BFRs.

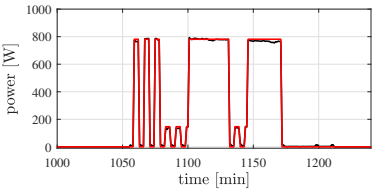
To this end, as in the Rarely Jump Models case, the performance of the estimated models is assessed with respect the validation dataset $\mathcal{V}_5$, to test the ability of the models to discriminate power consumption from concatenated sequences of readings of multiple appliances. Figure 47 shows which of the four appliance is actually consuming at each sample, along with the BFRs provided for each time window by the four estimated Markov Jump Models. Results similar to the ones obtained for the RJM case are achieved. In particular, the model associated to the active device always provides the highest BFR, except for the 59th window.



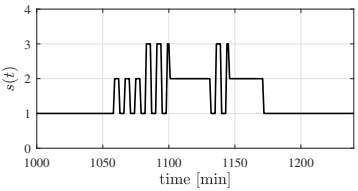
(a) Cloth dryer: output.



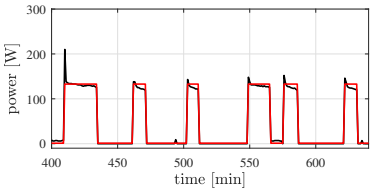
(b) Cloth dryer: discrete-state sequence.



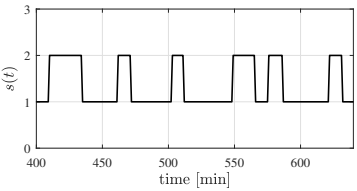
(c) Dishwasher: output.



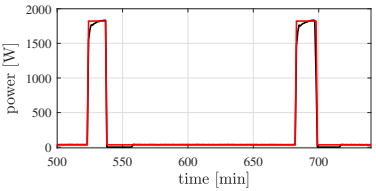
(d) Dishwasher: discrete-state sequence.



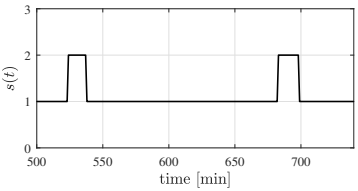
(e) Fridge: output.



(f) Fridge: discrete-state sequence.

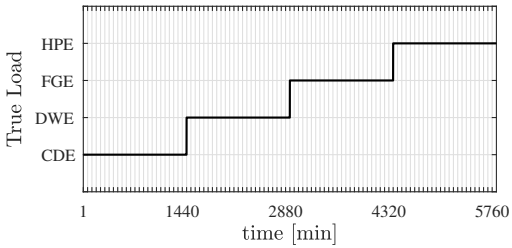


(g) Heat pump: output.

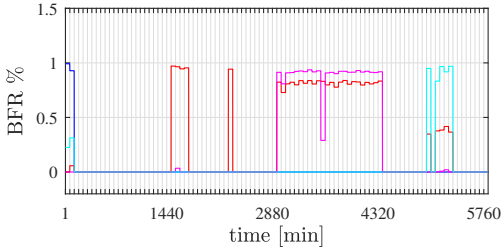


(h) Heat pump: discrete-state sequence.

Figure 46: Markov Jump Models learning: experimental example. Predicted output (red) vs actual output (black) (left panels) and estimated state sequence (right panels).



(a) Sequence of active appliance.



(b) BFRs provided by the 4 estimated models. Model for cloth dryer (blue), dishwasher (red), fridge (magenta), heat pump (cyan).

Figure 47: Markov Jump Models learning: experimental example. Validation set $\mathcal{V}_5$, active appliance and computed BFRs.

Chapter 5

Energy Disaggregation

Two methods for Non-Intrusive Load Monitoring (NILM) are presented in this Chapter. Both the methods are designed assuming the behavior of the appliances to be approximately known. In particular, the energy-use behavior of the devices is modeled with an RJM, obtained with the learning approach introduced in Chapter 4.
In Section 5.1 we formalize the problem of disaggregation and, in Section 5.2, we briefly summarized the techniques used to model the behavior of the appliances. Section 5.2 is devoted to the description of the proposed NILM method. The results obtained testing all the methods against a benchmark on real world power consumption data [74] are reported and discussed in Section 5.4.

Throughout the chapter $\mathbf{1}_{[\mathcal{A}]}$ denotes the indicator function of condition $\mathcal{A}$, *i.e.*,

$$\mathbf{1}_{[\mathcal{A}]} = \begin{cases} 1 & \text{if } \mathcal{A} \text{ is true} \\ 0 & \text{otherwise,} \end{cases}$$

5.1 Problem formulation

Consider N different electrical appliances available in a house and connected to the electric power line. Let $y_i(t)$, with $i \in \{1, \dots, N\}$, be the power demand of the ith appliance at time t and $y(t)$ be the household

aggregate power reading, *i.e.*,

$$y(t) = \sum_{i=1}^{N} y_i(t) + e(t), \tag{5.1}$$

where $e(t)$ is a modelling error, accounting for additional appliances connected to the line and measurement noise on the aggregate power reading.

Given a sequence $\{y(t)\}_{t=1}^{T}$ of observations of the aggregate power signal, energy disaggregation (also known as non-intrusive appliance load monitoring) aims at estimating the power demand $y_i(t)$ of the single appliances at each time sample t.

5.2 Modelling appliance behaviour

The power demand of the i-th appliance is described by K_i sub-models, with $K_i \in \mathbb{N}$, each one representing the consumption behaviour of the appliance at a different operating condition (or mode). The power demand of the i-th appliance at mode j (with $j = 1, \ldots, K_i$) is modelled as:

$$X_i(t)\theta_i^j + e_i(t), \tag{5.2}$$

where θ_i^j and $X_i(t)$ are the model parameter and the feature vector, respectively, characterizing the power demand at mode j, and $e_i(t)$ is an intrinsic modelling error. Let $s_i(t) \in \{1, \ldots, K_i\}$ be the active mode of the i-th appliance at time t. From (5.2), the power consumption of the i-th appliance is then given by:

$$y_i(t) = X_i(t)\theta_i^{s_i(t)} + e_i(t). \tag{5.3}$$

Typical consumption profiles of each device are assumed to be available. This information is essential to model the consumption pattern of the appliances, namely, to estimate the parameter vectors $\Theta_i = (\theta_i^1, \ldots, \theta_i^{K_i})$, $i = 1, \ldots, N$. In practice, we assume that N distinct datasets $\mathcal{M}_i = \{y_i(\tau)\}_{\tau=1}^{\bar{T}}$, $i = 1, \ldots, N$, are available. The dataset $\mathcal{M}_i$ consists of the

power demand of the i-th appliance gathered over a short intrusive training period of length $\bar{T}$, where we suppose to have direct access to the power consumption of the i-th appliance, for instance, by switching off all the other appliances. The length $\bar{T}$ of the training period should be as short as possible to reduce intrusiveness and costs, and it should not be necessarily the same for all devices.

Two different classes of sub-models (5.3) are used to describe the consumption pattern of the appliances at a given operating mode: (*i*) *static models*; (*ii*) *dynamical models*.

5.2.1 Learning static models

In case *static* models are used, the feature vector $X_i(t)$ is set to $X_i(t) = 1$. Therefore,the sub-model in (5.3) becomes

$$y_i(t) = \theta_i^{s_i(t)}. \tag{5.4}$$

The parameter θ_i^j thus represents the power consumption of the i-th appliance at mode j, $j = 1, \dots, K_i$.
Equation (5.4) shows that the estimation of the parameters Θ_i requires also to reconstruct the sequence of active modes $S_i = \{s_i(t)\}_{t=1}^{\bar{T}}$. The sub-model parameters Θ_i and the mode sequence S_i are jointly estimated through the jump model fitting approach in [12], based on the minimization over Θ_i and S_i of the *loss function* $J(\Theta_i, S_i)$ already introduced in chapter 4, which is equal to

$$\sum_{t=1}^{\bar{T}-1} \left(\ell\left(y_i(t), s_i(t), \Theta_i\right) + \lambda_i(s_i(t)) \mathbf{1}_{[s_i(t+1) \neq s_i(t)]} \right) + \ell\big(y_i(\bar{T}), s_i(\bar{T}), \Theta_i\big), \tag{5.5}$$

where $\ell\left(y_i(t), s_i(t), \Theta_i\right)$ is a *fitting cost* penalizing the mismatch between the measured and the model output. Among possible fitting costs, we choose the 2-norm of the fitting error

$$\ell\left(y_i(t), s_i(t), \Theta_i\right) = \frac{1}{\bar{T}} \| y_i(t) - \theta_i^{s_i(t)} \|_2^2. \tag{5.6}$$

The term $\lambda_i(s_i(t)) \mathbf{1}_{[s_i(t+1) \neq s_i(t)]}$ in (5.5) takes into account prior assumptions on the switch between different modes, penalizing by a factor $\lambda_i \geq$

0 a temporal change of the operating condition. This reflects the hypothesis that each device rarely changes its operating regime over time, which is a reasonable assumption in energy disaggregation problems with power readings taken at high-time resolution (*e.g.*, 1 min). The hyper-parameter λ_i is chosen as the empirical probability (with Laplace smoothing) of remaining in the same mode for two consecutive time instants, *i.e.*,

$$\lambda_i(j) = \frac{\sum_{t=1}^{\bar{T}-1} \mathbf{1}_{[s_i(t)=j, s_i(t+1)=j]} + 1}{\sum_{t=1}^{\bar{T}-1} \mathbf{1}_{[s_i(t)=j]} + K_i^2}, \quad j = 1, \ldots, K_i. \tag{5.7}$$

The cost $J(\Theta_i, S_i)$ in (5.5) is minimized through Algorithm 11, a coordinate descent approach, originally proposed in [12], that alternates minimization with respect to the sub-model parameters Θ_i (Step 1.1) and the mode sequence S_i (Step 1.2). Since the fitting cost ℓ is chosen as the quadratic function (5.6), Step 1.1 is solved analytically through least squares, while Step 1.2 is solved by discrete *dynamic programming* (DP). The initial mode sequences S_i^0 are chosen randomly, and the initial values for the hyper-parameters $\lambda_i^0(j)$ are set to 0, and then updated in Step 1.3. Technical details on the implementation of Algorithm 11 can be found in [12].

5.2.2 Learning dynamic models

Since some appliances exhibit a transient behaviour, their power consumption patterns can be more accurately described by dynamical submodels instead of static ones. For instance, a transient behaviour can be clearly observed in the consumption patterns of the fridge and the heat pump (see Figure 48), which show typical dynamics of second and first order *Linear Time-Invariant* (LTI) systems, respectively.

Provided that the sequence of active modes S_i in (5.3) is given, the LTI dynamical sub-model associated to each mode can be estimated through standard system identification techniques [72]. Specifically, the power consumption $y_i(t)$ is modelled as (5.3), with regressor $X_i(t) = [\tilde{X}_i(t) \;\; 1]$,

Algorithm 11 Learning static models

Input: Training data set $\mathcal{M}_i = \{y_i(t)\}_{t=1}^{\bar{T}}$; number K_i of sub-models; initial mode sequence $S_i^0 = \{s_i^0(1), \ldots, s_i^0(\bar{T})\}$; initial hyper-parameters $\lambda_i^0(j)$, $j = 1, \ldots, K_i$.

1. **iterate for** $h = 1, \ldots$

   1.1. $\Theta_i^h \leftarrow \text{argmin}_{\Theta_i} \sum_{t=1}^{\bar{T}} \ell\left(y_i(t), s_i^{h-1}(t), \Theta_i\right)$; (sub-model fitting)

   1.2. $S_i^h \leftarrow \text{argmin}_{S_i} J(\Theta_i^h, S_i)$; (mode sequence fitting)

   1.3. $\lambda_i^h(j) = \frac{\sum_{t=1}^{\bar{T}-1} \mathbf{1}_{[s_i^h(t)=j, s_i^h(t+1)=j]} + 1}{\sum_{t=1}^{\bar{T}} \mathbf{1}_{[s_i^h(t)=j]} + K_i^2}$, $j = 1, \ldots, K_i$.

2. **until** $S_i^h = S_i^{h-1}$.

Output: Estimated model parameters $\Theta_i^\star = \Theta_i^k$ and mode sequences $S_i^\star = S_i^k$.

where $\tilde{X}_i(t)$ consists of past output samples, *i.e.*,

$$\tilde{X}_i(t) = [y_i(t-1) \ \ldots \ y_i(t-n)], \tag{5.8}$$

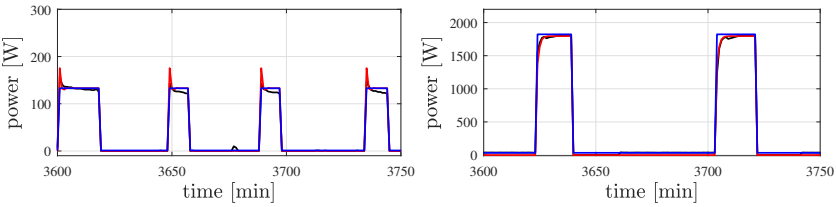
with $n \in \mathbb{N}$ defining the dynamical order of the model. The parameters Θ_i are estimated solving the simulation-error minimization problem

$$\min_{\Theta_i} \sum_{t=n+1}^{\bar{T}} \left(y_i(t) - \hat{y}_i(t, \Theta_i, s_i(t))\right)^2, \tag{5.9}$$

where $\hat{y}_i(t)$ is the simulated output given by

$$\hat{y}_i(t, \Theta_i, s_i(t)) = [\hat{y}_i(t-1, \Theta_i, s_i(t-1)) \ \ldots \ \hat{y}_i(t-n, \Theta_i, s_i(t-n)) \ 1]\theta_i^{s_i(t)}. \tag{5.10}$$

Because of the nested dependence of $\hat{y}_i(t, \Theta_i, s_i(t))$ on the model parameters Θ_i, the optimization problem in (5.9) is non convex and solved through Particle Swarm Optimization [91]. Since the sequence of active mode S_i in (5.9) is actually not known, Algorithm 11 is first run to estimate S_i using static sub-models.



(a) Fridge (b) Heat Pump

Figure 48: Power consumption profiles: true (black), estimated with static sub-models (blue), estimated with dynamic sub-models (red). Black and red lines are almost overlapped.

The power consumption for the fridge and the heat pump estimated using static and second-order dynamical models are plotted in Figure 48, along with the actual consumption profiles. The reported results show the capabilities of dynamical models to reconstruct the transient of the considered appliances. Similar performance is obtained for other electrical devices.

5.3 Disaggregation algorithms

Once the models of each appliance have been estimated, the energy disaggregation problem requires to detect, from the aggregate reading $y(t)$, the active mode $s_i(t)$ for each appliance and at each time instant. In this section, we describe two different algorithms for energy disaggregation, which process data iteratively and thus are suited for an online implementation when data are acquired in real time.

In the following, we refer to the *joint active mode* $s(t) \in \mathbb{N}^N$ as the the vector stacking the appliances' modes at time t, *i.e.*, $s(t) = [s_1(t), \ldots, s_N(t)]$. We denote with $\mathcal{S}$ the set of all possible combinations taken by $s(t)$ and $|\mathcal{S}|$ the cardinality of $\mathcal{S}$, *i.e.*, $|\mathcal{S}| = \prod_{i=1}^{N} K_i$.

5.3.1 Dynamic-programming based disaggregation

The first algorithm for iterative energy disaggregation is based on the minimization of the *loss function*:

$$J(t, s(t)) = \ell(y(1), s(1)) + \left[\sum_{\tau=2}^{t} \ell(y(\tau), s(\tau)) + \lambda(s(\tau-1)) \mathbf{1}_{[s(\tau) \neq s(\tau-1)]} \right], \tag{5.11}$$

which penalizes, similarly to (5.5), the time switch of the joint mode, as well as the fitting error $\ell(y(t), s(t))$ on the aggregate power measurement $y(t)$ defined as:

$$\ell(y(t), s(t)) = \left(y(t) - \sum_{i=1}^{N} \hat{y}_i(t, \Theta_i, s_i(t)) \right)^2.$$

In penalizing transitions on the joint mode we have taken into account the assumption, already used in Section 5.2, that the appliances rarely change their operating mode over time. The hyper-parameter $\lambda(d)$ in (5.11), with $d \in \mathcal{S}$, is proportional to the empirical probability of remaining at mode d for two consecutive time instants. The value of $\lambda(d)$ is computed based on the empirical probabilities $\lambda_i(j)$ (5.7), with $i = 1, \ldots, N$ and $j = 1, \ldots, K_i$. Under the assumption that the appliances change their modes independently from each other, $\lambda(d)$ is chosen as

$$\lambda(d) \propto \prod_{i=1}^{N} \lambda_i(d_i). \tag{5.12}$$

Unlike the minimization of (5.5), the cost (5.11) has not to be optimized with respect to the model parameters Θ_i, but only with respect to the active mode sequence $\{s(\tau)\}_{\tau=1}^{t}$. Since disaggregation should be performed in real time, only the aggregate readings up to time t can be used to reconstruct the active mode $s(t)$. The minimization of the cost $J(t, s(t))$ in (5.11) is performed through dynamic programming, as described in Algorithm 12. For each possible joint mode $h \in \mathcal{S}$, the value of $J(1, h)$ is computed at Step 1, and the joint mode $s(1)$ is selected as the one minimizing $J(1, h)$ over $h \in \mathcal{S}$ (Step 2). At time $t \geq 2$, the optimal costs

Algorithm 12 Dynamic-programming based disaggregation

Input: Aggregate output readings flow $y(1), y(2), \ldots$; model parameters Θ_i, $i = 1, \ldots, N$; hyper-parameters $\lambda(d)$, $d \in \mathcal{S}$.

1. $J^*(1,h) \leftarrow \ell(y(1),h)$; $h \in \mathcal{S}$;
2. $s^*(1) \leftarrow \text{argmin}_{h\in\mathcal{S}}\, J^*(1,h)$;
3. **iterate for** $t = 2, \ldots$
   - 3.1. $J^*(t,h) \leftarrow \ell(y(t),h) + \min_{d\in\mathcal{S}} \left(J^*(t-1,d) + \lambda(d)\mathbf{1}_{[h\neq d]}\right)$, $h \in \mathcal{S}$;
   - 3.2. $s^*(t) \leftarrow \text{argmin}_{h\in\mathcal{S}}\, J^*(t,h)$;

Output: Estimated joint mode sequence $s^*(t)$.

$J^*(t,h)$, for $h \in \mathcal{S}$, are updated based on the previously computed optimal costs $J^*(t-1,d)$ and the current aggregate power measurement $y(t)$ (Step 3.1). The optimal joint mode $s^*(t)$ is finally computed at Step 3.2.

It is worth remarking that, in case static models are used, the dynamic programming approach in Algorithm 12 provides the optimal mode $s^*(t)$ minimizing the cost $J(t,s(t))$ in (5.11), given the information up to time t. If dynamical models are used, the fitting cost $\ell(y(t),h)$ is computed approximating the single appliance consumptions with the previous estimates $\hat{y}_i(t-1), \ldots, \hat{y}_i(t-n)$ obtained at time $t-1, \ldots, t-n$. This implies that the history up to time $t-1$ is embedded into $\hat{y}_i(t-1), \ldots, \hat{y}_i(t-n)$, thus leading to an approximation of the optimum of the objective function $J(t,h)$.

The t-th iteration of Algorithm 12 is schematized in Figure 49. Note that, the update of the cost $J^*(t,h)$ at step 3.1 can be recursively computed based on $J^*(t-1,d)$, $d \in \mathcal{S}$ and the new observation $y(t)$, without the need to store and reprocess past data. This makes Algorithm 12 suited for online disaggregation.

Note that, at each time sample t, Algorithm 12 computes the cost-to-go $J^*(t,h)$ at mode h for all possible values of $h \in \mathcal{S}$, which requires to evaluate the cost $J^*(t-1,d) + \lambda(d)\mathbf{1}\,(h \neq d)$ for all possible values of

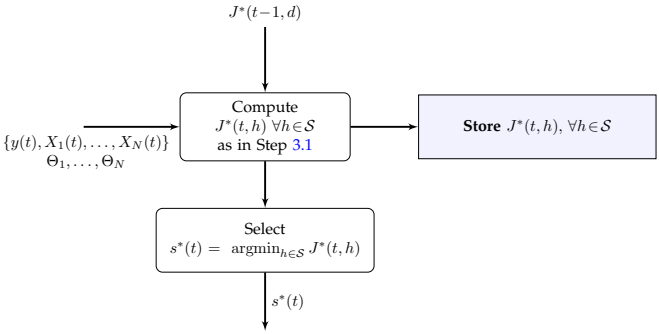


Figure 49: Dynamic-programming based disaggregation: iteration at time t.

$d \in \mathcal{S}$ (Step 3.1). Thus, the computational complexity of Algorithm 12 in detecting the joint mode $s(t)$ is $O(|\mathcal{S}|^2)$. Since the number $|\mathcal{S}|$ of possible combinations of the joint mode $s(t)$ increases exponentially with the number of appliances N and the number of operating conditions K_i, $i = 1, \ldots, N$, implementation of Algorithm 12 might not be practically feasible in case of large N and K_i.

Complexity reduction

A possible solution to reduce the computational complexity of Algorithm 12 from $O(|\mathcal{S}|^2)$ to $O(|\mathcal{S}|)$ is to approximate Step 3.1 with

$$J^*(t,h) \leftarrow \ell(y(t),h) + J^*(t-1, s^*(t-1)) + \lambda(s^*(t-1))\mathbf{1}_{[h\neq s^*(t-1)]},\ h \in \mathcal{S} \tag{5.13}$$

where $s^*(t-1)$ is the estimate of the joint mode at the previous time step $t-1$, given by

$$s^*(t-1) = \operatorname*{argmin}_{h\in\mathcal{S}} J^*(t-1,h). \tag{5.14}$$

The idea behind approximation (5.13), schematized in Figure 50, is to embed the information up to time $t-1$ into the estimated mode $s^*(t-1)$.

To further reduce the complexity of Algorithm 12, the cost $J^*(t,h)$ in Step 3.1 is not computed for all possible modes $h \in \mathcal{S}$, but only for those satisfying at least one of the following conditions:

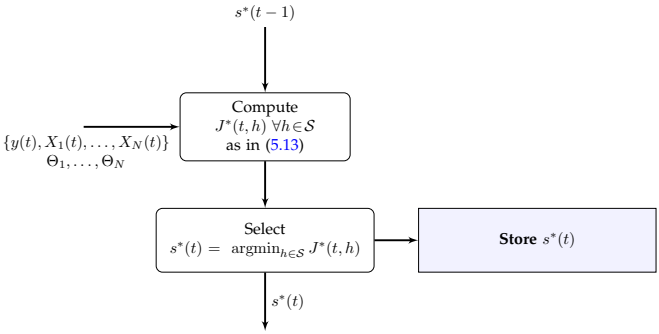


Figure 50: Schematic of the approximation used to reduce complexity of dynamic-programming based disaggregation from $O(|\mathcal{S}|^2)$ to $O(|\mathcal{S}|)$. All the past history up to time $t-1$ is embedded in $s^*(t-1)$.

$\mathcal{C}_1$ at most one appliance changes its operating condition;

$\mathcal{C}_2$ all the appliances are in the operating condition corresponding to their minimal consumption energy (namely, all of them are off);

$\mathcal{C}_3$ only one appliance is on.

These assumptions are realistic in energy disaggregation problems with high-time resolution (*e.g.*, 1 min) power readings. Although $\mathcal{C}_2$-$\mathcal{C}_3$ consider the cases where at most one appliance is on, condition $\mathcal{C}_1$ allows us to handle configurations where multiple appliances are consuming. When two devices are switched on simultaneously at time t, the operating mode of one of the two appliances cannot be correctly detected, as this configuration is not accounted for by $\mathcal{C}_1$-$\mathcal{C}_3$. Nevertheless, the actual operating mode is expected to be retrieved at time $t+1$ thanks to condition $\mathcal{C}_1$. Since only the configurations satisfying $\mathcal{C}_1$, $\mathcal{C}_2$ or $\mathcal{C}_3$ are considered, the computational complexity of the approach is further reduced from $O(|\mathcal{S}|)$ to $O\left(2+2\left(\sum_{i=1}^{N}(K_i-1)\right)\right)$. A schematic of the operations performed at time t is shown in Figure 51.

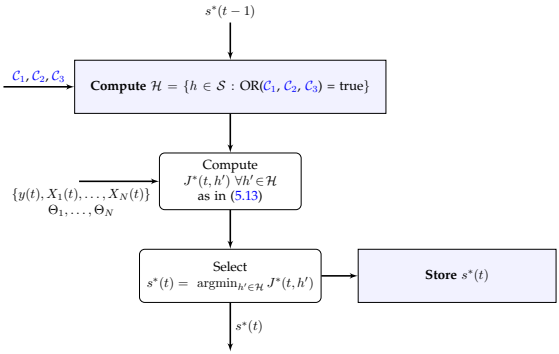


Figure 51: Schematic of the approximation used to further reduce complexity of dynamic-programming based disaggregation from $O(|\mathcal{S}|)$ to $O\left(2 + 2\left(\sum_{i=1}^{N}(K_i - 1)\right)\right)$. Only configurations described by conditions $\mathcal{C}_1$, $\mathcal{C}_2$ or $\mathcal{C}_3$ are considered.

5.3.2 Disaggregation through Kalman Filtering

Alternatively to the dynamic-programming based method described in Section 5.3.1, we propose another iterative approach based on a reformulation of energy disaggregation as a state estimation problem for switching linear dynamical systems. This problem is then solved via multiple-model Kalman filtering

A state-space representation of the switching linear model representing the household power consumption over time is given by:

$$x(t+1) = A[s(t)]x(t) + B[s(t)] + w[s(t)](t) \tag{5.15a}$$

$$y(t) = C[s(t)]x(t) + v[s(t)](t), \tag{5.15b}$$

where $y(t)$ is the measured aggregate power, $w[s(t)](t)$ and $v[s(t)](t)$ are the process and measurement noise, respectively, which are supposed to be white and mutually independent. According to standard hypothesis in Kalman filtering, $w[s(t)](t)$ and $v[s(t)](t)$ are assumed to be generated by zero-mean Gaussian distributions with covariance matrices $Q[s(t)]$ and $R[s(t)]$, *i.e.*, $w[s(t)](t) \sim \mathcal{N}(0, Q[s(t)])$ and $v[s(t)](t) \sim \mathcal{N}(0, R[s(t)])$. Similarly to Section 5.3.1, $s(t) \in \mathcal{S}$ represents the joint mode at time t.

The continuous state $x(t)$ and the matrices $A[s(t)]$, $B[s(t)]$, $C[s(t)]$ are properly defined based on the static/dynamic models of each device estimated as in Section 5.2. For example, in the case each appliance is described by static sub-models (5.4), the state $x(t)$ is the collection of the single appliances' consumptions $x(t) = [\, y_1(t) \ \dots \ y_N(t) \,]'$ and

$$A[s(t)] = 0_{N\times N} \quad B[s(t)] = \begin{bmatrix} \theta_1^{s_1(t)} \\ \vdots \\ \theta_N^{s_N(t)} \end{bmatrix} \qquad C[s(t)] = 1'_N,$$

with 1_N indicating the column vector of ones.
The time evolution of the mode $s(t)$ is described by a stationary *Markov Chain* with transition probabilities

$$P(s(t) = h|s(t-1) = d) \text{ with } h, d \in \mathcal{S}. \tag{5.16}$$

These transitions probabilities are approximated from the results of the training procedure described in Section 5.2.1, using the empirical probabilities with Laplace smoothing:

$$P(s_i(t)=h_i|s_i(t-1)=d_i)=\frac{\sum_{t=2}^{\bar{T}} \mathbf{1}_{[s_i(t)=h_i, s_i(t-1)=d_i]}+1}{\sum_{t=2}^{\bar{T}} \mathbf{1}_{[s_i(t-1)=d_i]} + K_i^2}, \ i=1,\dots,N. \tag{5.17}$$

Under the assumption that the appliances change their mode independently from each other, the transition probabilities in (5.16) are thus computed as

$$P(s(t) = h|s(t-1) = d) = \prod_{i=1}^{N} P(s_i(t) = h_i|s_i(t-1) = d_i). \tag{5.18}$$

Multiple-model Kalman filtering techniques can be used to simultaneously estimate both the joint mode $s(t)$ and the continuous state $x(t)$ of the dynamical system in (5.15). The *first-order generalized pseudo-Bayesian* (GPB_1) algorithm is employed. The main ideas behind GPB_1 are summarized in the rest of this section. Heuristics to reduce the computational complexity in applying GPB_1 to energy disaggregation problems are then described in Section 5.3.2.

Let $\mathcal{I}^t$ be the set of available data up to time t, i.e., $\mathcal{I}^t = \{y(1), \ldots, y(t)\}$. At each time t, GPB$_1$ approximates the state conditional probability density function $p[x(t)|\mathcal{I}^t]$ as:

$$p[x(t)|\mathcal{I}^t] = \sum_{h=1}^{|\mathcal{S}|} p[x(t)|s(t)=h, \mathcal{I}^t] P(s(t)=h|\mathcal{I}^t)$$

$$= \sum_{h=1}^{|\mathcal{S}|} p[x(t)|s(t)=h, y(t), \mathcal{I}^{t-1}] P(s(t)=h|\mathcal{I}^t)$$

$$\approx \sum_{h=1}^{|\mathcal{S}|} p[x(t)|s(t)=h, y(t), \hat{x}(t-1|t-1), P_x(t-1|t-1)] P(s(t)=h|\mathcal{I}^t). \quad (5.19)$$

The idea behind this approximation is to embed the past information $\mathcal{I}^{t-1}$ on the system into the state estimate $\hat{x}(t-1|t-1)$ computed at time $t-1$ (using only information up to $t-1$) and the associated covariance matrix $P_x(t-1|t-1)$.

By assuming that the initial state $x(0)$ is Gaussian distributed with mean $\hat{x}(0|0)$ and covariance $P_x(0|0)$, the probability density function

$$p[x(t)|s(t)=h, y(t), \hat{x}(t-1|t-1), P_x(t-1|t-1)]$$

is Gaussian with mean $\hat{x}[h](t|t)$ and covariance $P_x[h](t|t)$, where $\hat{x}[h](t|t)$ and $P_x[h](t|t)$ are the output of the Kalman filter associated with the linear sub-model (5.15) for $s(t)=h$.

Thus, $p[x(t)|\mathcal{I}^t]$ in (5.19) is a Gaussian mixture, with weights

$$\alpha[h](t|t) = P(s(t)=h|\mathcal{I}^t), \quad (5.20)$$

where $\alpha[h](t|t)$ thus represents the probability of being at mode h at time t, given the information up to time t. These weights can be equivalently expressed as

$$\alpha[h](t|t) = P(s(t)=h|y(t), \mathcal{I}^{t-1}) = \frac{p[y(t)|s(t)=h, \mathcal{I}^{t-1}] P(s(t)=h|\mathcal{I}^{t-1})}{\sum_{j\in\mathcal{S}} p[y(t)|s(t)=j, \mathcal{I}^{t-1}] P(s(t)=j|\mathcal{I}^{t-1})}, \quad (5.21)$$

where $P(s(t) = h|\mathcal{I}^{t-1})$ is the probability of being at mode h at time t given the measurements up to time $t-1$, and it can be computed iteratively as

$$\alpha[h](t|t-1) = P(s(t) = h|\mathcal{I}^{t-1}) = \sum_{d \in \mathcal{S}} P(s(t) = h|s(t-1) = d)\alpha[d](t-1|t-1), \tag{5.22}$$

with $P(s(t) = h|s(t-1) = d)$ given by (5.17) and (5.18).

As in (5.19), the past information $\mathcal{I}^{t-1}$ is embedded into $\hat{x}(t-1|t-1)$ and $P_x(t-1|t-1)$. This allows us to approximate the conditional likelihood $p[y(t)|s(t) = h, \mathcal{I}^{t-1}]$ of the aggregate output $y(t)$ as

$$p[y(t)|s(t) = h, \mathcal{I}^{t-1}] \approx p[y(t)|s(t) = h, \hat{x}(t-1|t-1), P_x(t-1|t-1)]. \tag{5.23}$$

Using the dynamical equations (5.15) and prior assumptions on the distributions of $x(0)$, $w[h](t)$ and $v[h](t)$, the approximated likelihood in (5.23) is Gaussian with mean $C[h]\hat{x}[h](t|t-1)$ and covariance $R[h]+C[h]P_x[h](t|t-1)C[h]^T$, where

$$\begin{aligned} \hat{x}[h](t|t-1) &= A[h]\hat{x}(t-1|t-1) + B[h] \\ P_x[h](t|t-1) &= A[h]^T P_x(t-1|t-1)A[h] + Q[h]. \end{aligned}$$

Summarizing, the weights of the Gaussian mixture $p[x(t)|\mathcal{I}^t]$ in (5.19) are calculated using (5.21) and (5.23). The state estimate $\hat{x}(t|t)$ and the associated covariance $P_x(t|t)$ are then chosen as the expected value and covariance matrix of the random variable $x \sim p[x|\mathcal{I}^t]$, namely:

$$\hat{x}(t|t) = \sum_{h \in \mathcal{S}} \hat{x}[h](t|t)\alpha[h](t), \tag{5.24}$$

$$P(t|t) = \sum_{h \in \mathcal{S}} \alpha[h](t|t) \left\{ P[h](t|t) + [\hat{x}[h](t|t) - \hat{x}(t|t)][\hat{x}[h](t|t) - \hat{x}(t|t)]' \right\}. \tag{5.25}$$

The active mode at time t is finally selected as

$$s^*(t) = \underset{h \in \mathcal{S}}{\operatorname{argmax}}\, \alpha[h](t|t), \tag{5.26}$$

and the final disaggregated power of each appliance is retrieved from the estimated state $\hat{x}[h](t|t)$, for $h = s^*(t)$.

Algorithm 13 Kalman filter based disaggregation

Input: Aggregate output readings flow $y(1), y(2), \ldots$; models $A[h], B[h], C[h]$ and noise covariance matrices $Q[h], R[h]$, $h \in \mathcal{S}$; prior on the initial state $\hat{x}(0|0), P_x(0|0)$, initial mode probabilities $\alpha[h](0|0)$, $h \in \mathcal{S}$.

1. **iterate for** $t = 1, 2, \ldots$
   - 1.1. **update** $\hat{x}[h](t|t), P_x[h](t|t)$, $h \in \mathcal{S}$, using linear Kalman filter;
   - 1.2. **compute** the likelihood $p[y(t)|s(t) = h, \mathcal{I}^{t-1}]$, $h \in \mathcal{S}$, as in (5.23);
   - 1.3. **update** $\alpha[h](t|t)$, $h \in \mathcal{S}$, as in (5.21);
   - 1.4. $s^*(t) \leftarrow \text{argmax}_{h \in \mathcal{S}}\, \alpha[h](t|t)$, $h \in \mathcal{S}$;
   - 1.5. **compute** $\hat{x}(t|t)$ as in (5.24);
   - 1.6. **compute** $P_x(t|t)$ as in (5.25);

Output: Estimated sequence of joint mode $s^*(t)$ and optimal state $\hat{x}[s^*(t)](t|t)$.

Algorithm 13 summarizes the iterations of the Kalman filter based disaggregation approach. If no prior on the initial mode probabilities $\alpha[h](0|0) = P(s(0) = h)$ is available, Algorithm 13 can be initialized by setting $\alpha[h](0|0) = \frac{1}{|\mathcal{S}|}$ for all $h \in \mathcal{S}$. This is equivalent to assume a uniform probability distribution on the initial mode $s(0)$.

The t-th step of Algorithm 13 is also schematized in Figure 52. Note that, at each iteration t, the computations in Steps 1.1- 1.3 can be recursively performed based on $\hat{x}(t-1|t-1), P_x(t-1|t-1)$ and the new observation $y(t)$, without the need to store and reprocess past data. Thus, like Algorithm 12, also Algorithm 13 is suited for online disaggregation.

Since Steps 1.1-1.3 of Algorithm 13 should be performed for each $h \in \mathcal{S}$, its complexity is $O(|\mathcal{S}|)$. As the number $|\mathcal{S}|$ of possible joint modes $s(t)$ increases exponentially with the number of appliances N and the number of corresponding operating conditions K_i, $i = 1, \ldots, N$, the approach is limited to disaggregation problems with few devices and with

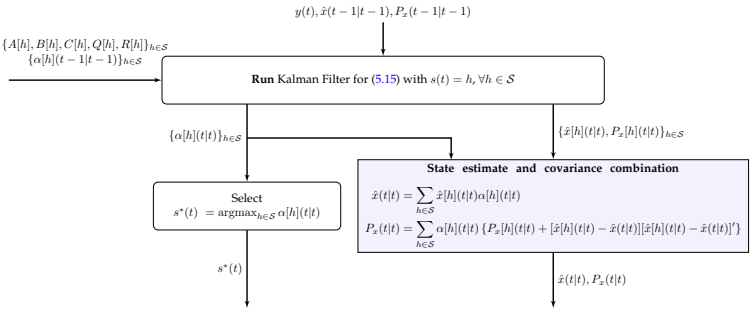


Figure 52: Kalman filter based disaggregation approach: iteration at step t.

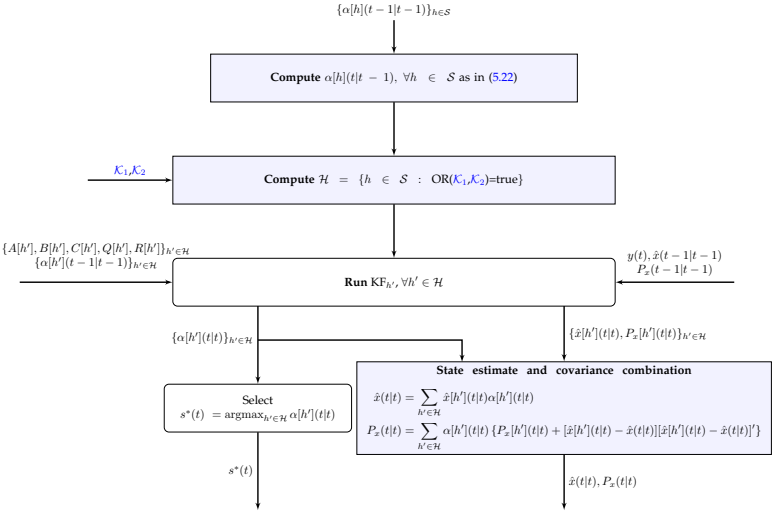


Figure 53: Scheme of the reduced-complexity KF-based approach reporting the operations performed at step t.

a small number of operating regimes for each appliance.

Complexity reduction

To reduce the computational complexity of Algorithm 13, Step 1.1 (which requires to run $|\mathcal{S}|$ Kalman filters in parallel) is performed at time t only

for the modes h' in $\mathcal{S}$ satisfying the following conditions:

$\mathcal{K}_1$ The probability $\alpha[h'](t|t-1) = P(s(t) = h'|\mathcal{I}^{t-1})$ is larger than a threshold ε, *i.e.*,

$$\alpha[h'](t|t-1) \geq \varepsilon.$$

A possible value for ε is $1/|\mathcal{S}|$;

$\mathcal{K}_2$ all the appliances are in the operating condition corresponding to their minimal consumption energy (namely, all of them are off);

$\mathcal{K}_3$ only one appliance is on.

Conditions $\mathcal{K}_2$-$\mathcal{K}_3$ are equal to conditions $\mathcal{C}_2$-$\mathcal{C}_3$ already used in Section 5.3.1 to reduce the computational complexity of the dynamic programming based approach. Condition $\mathcal{K}_1$ allows us to discard the configurations which are predicted to be "unlikely".

5.4 Experimental tests

The proposed disaggregation algorithms are tested against the AMPds dataset [74], which consists of the power readings of a house located in Canada, and it comprises the consumption profiles of 19 appliances, recorded over a year (from April 1, 2012 to March 31, 2013) at one-minute time resolution. The goal of the test is to assess: (i) the capabilities of the proposed disaggregation algorithms in reconstructing the end-use power consumption from the aggregate readings; (ii) their computational complexity; (iii) their robustness against modelling errors. In running Algorithm 13 the covariance matrices $P_x(0|0)$, $Q[h]$ and $R[h]$ are chosen as diagonal matrices with non-zero entries equal to 1000, 10 and 800, respectively. The initial parameter $\hat{x}(0|0)$ is a zero vector and the initial probabilities $\alpha[h](0|0)$ are set to and $\frac{1}{|\mathcal{S}|}$, for $h \in \mathcal{S}$. The threshold ε characterizing condition $\mathcal{K}_1$ (Section 5.3.2)is equal to $\frac{1}{|\mathcal{S}|}$.

5.4.1 Learning appliance behaviour

Part of the AMPds dataset is used to construct the training sets $\mathcal{M}_i$, $i = 1, \ldots, N$ needed to estimate the models for the single devices. As

discussed in Section 5.2, in the training phase we assume to have access to the consumption patterns of the single appliances.

The following appliances are modelled: cloth dryer (CDE); dishwasher (DWE); fridge (FGE); heat pump (HPE); basements plugs & lights (BME).

First, static models (5.4) for each of the considered device are estimated. The following parameters θ_i^j are obtained:

$$\begin{aligned}
&\text{CDE} : \begin{bmatrix} 0.4 & 251.8 & 4644.3 \end{bmatrix}, \\
&\text{DWE} : \begin{bmatrix} 0.1 & 146.6 & 781.2 \end{bmatrix}, \\
&\text{FGE} : \begin{bmatrix} 1.1 & 132.8 \end{bmatrix}, \\
&\text{HPE} : \begin{bmatrix} 33.7 & 1822.1 \end{bmatrix}, \\
&\text{BME} : \begin{bmatrix} 5.7 & 330.5 \end{bmatrix}.
\end{aligned}$$

As discussed in Section 5.2.1, these parameters represent the estimate of the power consumption of the single appliances at different operating regimes.

For the fridge and the heat pump, second-order dynamical models are also estimated. Results regarding the quality of the estimated dynamical models have been already presented in Figure 48.

5.4.2 Performance metrics

Disaggregation is performed on a dataset $\mathcal{D}_T$ of length T (disjoint from the training sets $\mathcal{M}_i$, $i = 1, \dots, N$) which consists only of the aggregate power readings. The length T of the dataset $\mathcal{D}_T$ is equal to 17280, which corresponds to 12 consecutive days of observations. The available end-use profiles are employed only as ground-truth data to assess the quality of the disaggregated consumption patterns, which is measured with respect to the following metrics:

1. the F-score (Fs) [8]

$$Fs_i = 2\frac{PC_i \times RC_i}{PC_i + RC_i}, \tag{5.27}$$

where the indexes RC_i and PC_i in (5.27) are the so-called *recall* and *precision*, respectively, and they are defined as

$$RC_i = \frac{TP_i}{TP_i + FN_i}, \quad PC_i = \frac{TP_i}{TP_i + FP_i},$$

where TP_i, FP_i and FN_i are respectively: the number of events correctly classified when the appliance is on (true positive); the number of events classified as on when the appliance is actually off (false positive); the number of events classified as off when the appliance is actually on (false negative). The score Fs_i measures the capability of the disaggregation method in correctly classifying the on/off state of the i-th appliance. Since multiple-state appliances are used, we set 10 W as a threshold to claim whether the appliance is on or off. The threshold is set to 50 W for the heat pump, as its estimated low-level power consumption is 33.7 W (see Section 5.4.1);

2. the *Estimated Energy Fraction Index* (EEFI)

$$\text{EEFI}_i = \frac{\sum_{t=1}^{T} \hat{y}_i(t, \Theta_i, s_i^*(t))}{\sum_{i=1}^{N} \sum_{t=1}^{T} \hat{y}_i(t, \Theta_i, s_i^*(t))}, \tag{5.28}$$

which represents the fraction of energy assigned to the i-th appliance. This index is compared with the *Actual Energy Fraction Index* (AEFI)

$$\text{AEFI}_i = \frac{\sum_{t=1}^{T} y_i(t)}{\sum_{i=1}^{N} \sum_{t=1}^{T} y_i(t)}, \tag{5.29}$$

which indicates the actual fraction of energy consumed by the i-th appliance. A similar value between EEFI_i and AEFI_i indicates that the contribution of i-th on the total power consumption is correctly estimated.

3. the *Relative Square Error* (RSE)

$$\text{RSE}_i = \frac{\sum_{t=1}^{T} (y_i(t) - \hat{y}_i(t, \Theta_i, s_i^*(t)))^2}{\sum_{t=1}^{T} (y_i(t))^2}. \tag{5.30}$$

The RSE_i index provides a normalized measure of the mismatch between the actual and the reconstructed consumption pattern for the i-th appliance.

Table 20: Achieved F-scores Fs.

	Dynamic-programming approach		Kalman-filtering approach	
	complete	reduced	complete	reduced
Clothes dryer	97.8 %	98.8 %	90.4 %	89.5 %
Dishwasher	95.6 %	97.4 %	90.7 %	89.6 %
Fridge	94.4 %	96.1 %	76.0 %	75.5 %
Heat Pump	99.9 %	99.9 %	81.5 %	80.6 %
Basement	96.4 %	97.7 %	98.2%	98.9 %

4. the R^2 coefficient;

$$R_i^2 = 1 - \frac{\sum_{t=1}^{T}(y_i(t) - \hat{y}_i(t, \Theta_i, s_i^*(t)))^2}{\sum_{t=1}^{T}(y_i(t) - \bar{y}_i)^2} \tag{5.31}$$

with $\bar{y}_i = \frac{1}{T}\sum_{t=1}^{T} y_i(t)$. Both R_i^2 and RSE_i measure the match over time between estimated and actual end-use power profiles.

These metrics provide increasing levels of information on the end-use power consumptions. Indeed, the F-score only gives an indication on the capabilities of the disaggregation approach in detecting whether a device is on or off, while the EEFI_i index provides the power consumed by each appliance. This is more informative than the F-score to design customized feedbacks and demand management strategies. Finally, the RSE_i and R_i^2 indexes measure the quality of the reconstructed single-appliance power consumption trajectories over time, which is crucial to retrieve information about consumptions during peak hours.

5.4.3 Numerical results

The aggregate power readings $y(t)$ forming the validation dataset $\mathcal{D}_T$ are constructed by summing up the power consumptions of the 5 appliances specified in Section 5.4.1. To assess the robustness of the developed disaggregation approaches with respect to modelling errors, the obtained signal $y(t)$ is corrupted by a fictitious zero-mean Gaussian noise with standard deviation 4 W. Furthermore, the unmodelled consumption patterns of bedroom, garage and dining room are added on top of $y(t)$.

Table 21: Relative Square Errors RSE and R^2 coefficients. Results of the dynamic programming (DP)-based algorithm and its simplified version with reduced computational complexity.

	DP algorithm		DP with reduced complexity	
	RSE_{i}	R^2_i	RSE_{i}	R^2_i
Clothes dryer	15.7 %	84.1 %	12.8 %	87.0 %
Dishwasher	37.0 %	62.4 %	24.1 %	75.4 %
Fridge	42.5 %	39.6 %	33.6 %	52.2 %
Heat Pump	1.3 %	98.3 %	1.1 %	98.6 %
Basement	34.3 %	56.6 %	19.8 %	74.9 %

Table 22: Relative Square Errors RSE and R^2 coefficients. Results of the multiple-model Kalman-filtering (KF)-based algorithm and its simplified version with reduced computational complexity.

	KF algorithm		KF with reduced complexity	
	RSE_{i}	R^2_i	RSE_{i}	R^2_i
Clothes dryer	3.5 %	96.5 %	14.6 %	85.2 %
Dishwasher	8.0 %	91.8 %	25.9 %	73.6 %
Fridge	31.6 %	55.1 %	33.1 %	52.9%
Heat Pump	0.6 %	99.2 %	3.5 %	95.5 %
Basement	14.4 %	81.7 %	6.7%	91.6 %

Estimated end-use profiles

The obtained values of the performance metrics are provided in Tables 20, 21 and 22, and in Figures 54-55, while the disaggregated signals are plotted in Figures 56-65. For the sake of visualization, only a portion of the disaggregated profiles is reported in the figures. The obtained results show that both the dynamic-programming and the Kalman-filter-based approach accurately estimate the fraction of energy consumed by each appliance (see Figures 54-55, where the EEFI and the AEFI indexes are compared). This good performance is mainly due to an accurate estimate of the disaggregated trajectories over time (as shown in Figures 56-65, and quantified in terms of the RSE and R^2 indexes in Tables 21-22). It is interesting to note in Tables 20 and 21 that the simplified version of the

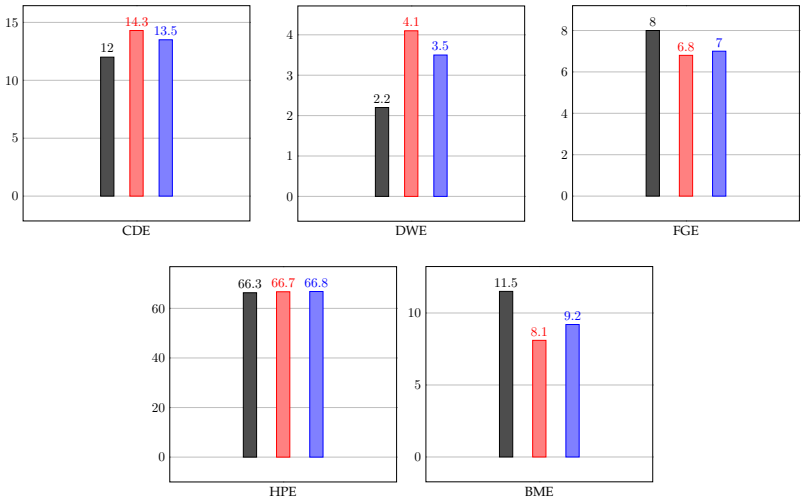


Figure 54: Dynamic-programming-based approach. Actual Energy Fraction Index [%] (black); Estimated Energy Fraction Index (EEFI) [%] by complete version (red) and simplified version (blue).

Table 23: Average CPU time, in milliseconds, required to disaggregate the total power consumption at a given time instant.

	Dynamic-programming approach		Kalman-filtering approach	
	complete	reduced	complete	reduced
CPU time [ms]	0.25	0.25	1.10	0.24

DP-based approach might even outperform the complete version of the algorithm. This is due to the fact that, because of conditions $\mathcal{C}_1$-$\mathcal{C}_3$ (see Section 5.3.1), the simplified approach a-priori discards some configurations which are unlikely to happen in practice.

Computational complexity

The tests were run on a MacBook Pro 2.8 GHz-Intel i7 in MATLAB R2016b. The average CPU times required to compute the disaggregated signals at each time instant are reported in Table 23. Both the complete and the reduced-complexity DP-based approaches take 0.25 ms to perform

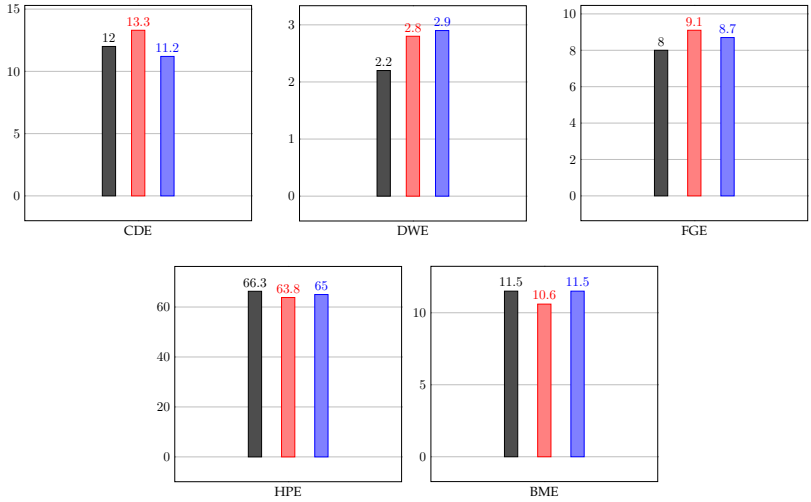
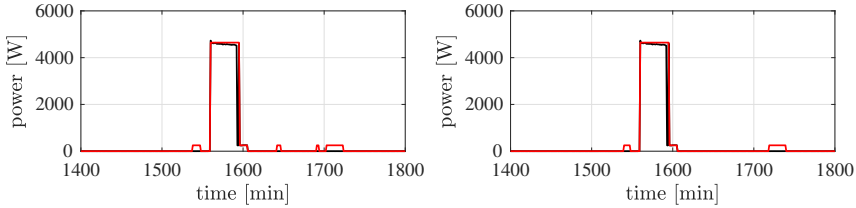


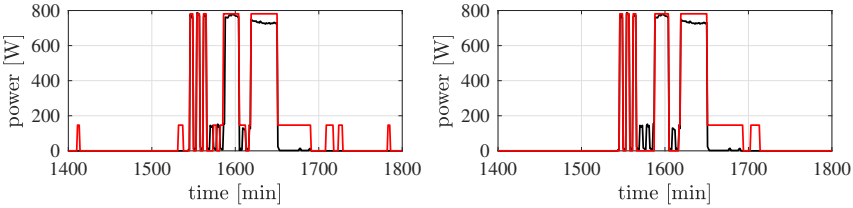
Figure 55: Multiple-model Kalman-filtering-based approach. Actual Energy Fraction Index [%] (black); Estimated Energy Fraction Index (EEFI) [%] by complete version (red) and simplified version (blue).



(a) DP algorithm (b) DP with reduced complexity

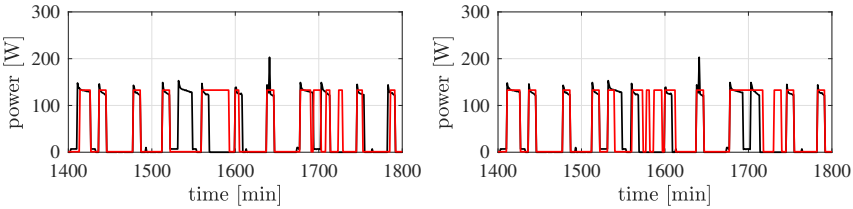
Figure 56: Cloths dryer: True (black) vs estimated (red) power demands.

a disaggregation step. Thus, in the considered application, the simplified method does not lead to any improvement in terms of CPU time, because of the overhead time required to check conditions $\mathcal{C}_1$-$\mathcal{C}_3$. On the other hand, the simplified KF-based approach is about $5\times$ faster than the complete version. These results show the potentiality of the proposed algorithms for big-data processing.



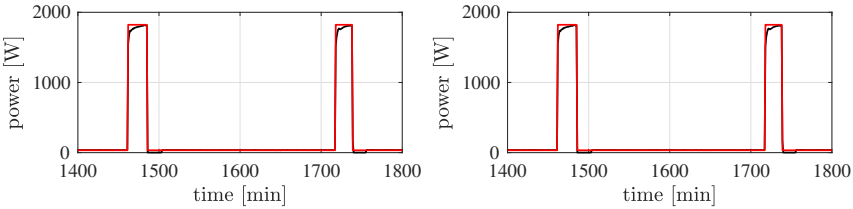
(a) DP algorithm (b) DP with reduced complexity

Figure 57: Dishwasher: True (black) vs estimated (red) power demands.



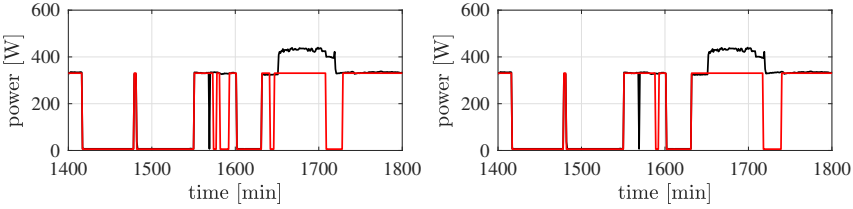
(a) DP algorithm (b) DP with reduced complexity

Figure 58: Fridge: True (black) vs estimated (red) power demands.



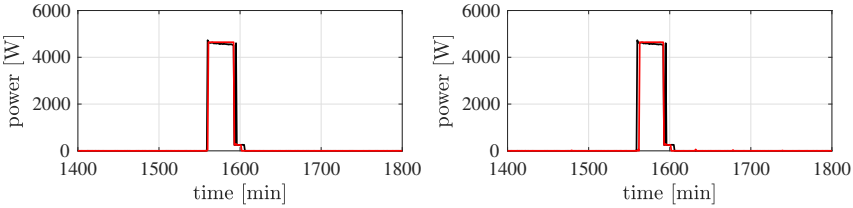
(a) DP algorithm (b) DP with reduced complexity

Figure 59: Heat Pump: True (black) vs estimated (red) power demands.



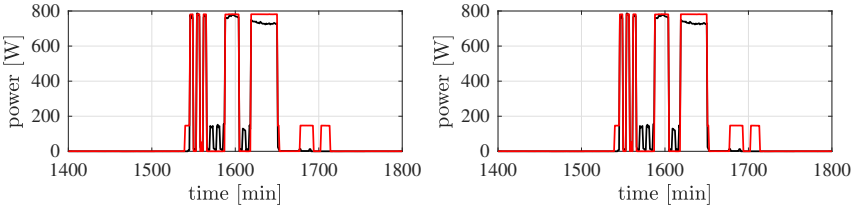
(a) DP algorithm (b) DP with reduced complexity

Figure 60: Basement plugs & lights: True (black) vs estimated (red) power demands.



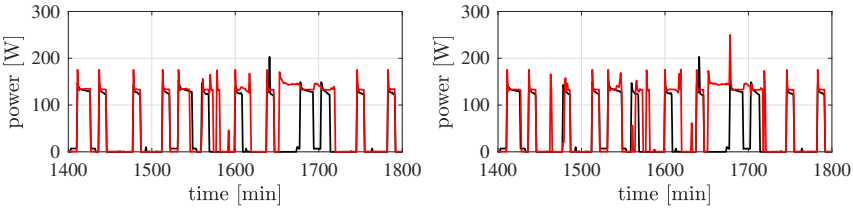
(a) KF algorithm (b) KF with reduced complexity

Figure 61: Cloths dryer: True (black) vs estimated (red) power demands.



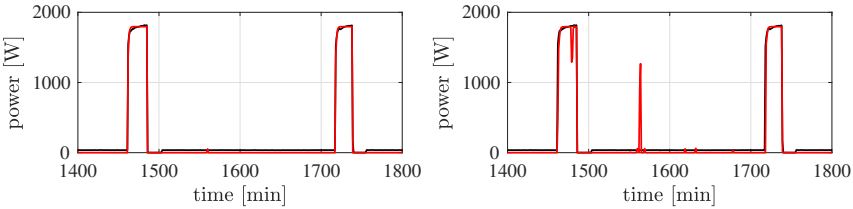
(a) KF algorithm (b) KF with reduced complexity

Figure 62: Dishwasher: True (black) vs estimated (red) power demands.



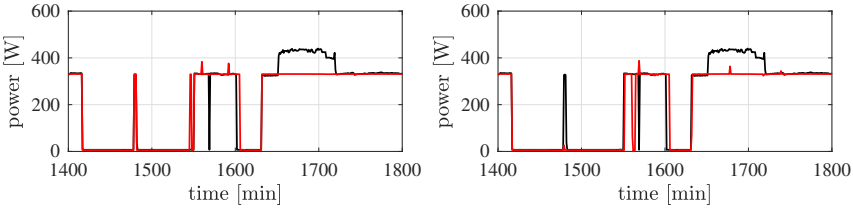
(a) KF algorithm (b) KF with reduced complexity

Figure 63: Fridge: True (black) vs estimated (red) power demands.



(a) KF algorithm (b) KF with reduced complexity

Figure 64: Heat Pump: True (black) vs estimated (red) power demands.



(a) KF algorithm (b) KF with reduced complexity

Figure 65: Basement plugs & lights: True (black) vs estimated (red) power demands.

Chapter 6

Cloud-aided collaborative estimation
The Linear case

This chapter presents methods for cloud-aided estimation, focusing on approaches tailored to handle linear consensus constraints based on the Alternating Direction Method of Multipliers (ADMM). Section 6.1 is thus devoted to briefly summarize the Alternating Direction Method of Multipliers. In Section 6.2 the collaborative estimation problem is formalized, while the solution for the case of full consensus is presented in Section 6.3. Section 6.4 is devoted to the description of the method designed to handle collaborative estimation problems with partial consensus. An extension of this approach to tackle constrained estimation problems is presented in Section 6.5. Simulation examples are considered to test all the algorithms, thus showing their performance in different settings.
While the approaches proposed in Section 6.3-6.4 for unconstrained full and partial consensus problems rely on Node-to-Cloud-to-Node (N2C2N) communications only, both a strategy based on N2C2N transmissions and a method relying on a Node-to-Cloud (N2C) communication scheme are presented in Section 6.5.

Throughout the Chapter $\mathcal{P}_{\mathcal{A}}$ denotes the Euclidean projection onto the

set $\mathcal{A}$.

6.1 The Alternating Direction Method of Multipliers

The Alternating Direction Method of Multipliers (ADMM) [23] is an algorithm tailored to solve problems in the form

$$\begin{aligned} &\text{minimize} && f(\theta) + g(z) \\ &\text{subject to} && A\theta + Bz = c, \end{aligned} \tag{6.1}$$

where $\theta \in \mathbb{R}^{n_\theta}$, $z \in \mathbb{R}^{n_z}$, $f : \mathbb{R}^{n_\theta} \to \mathbb{R} \cup \{+\infty\}$ and $g : \mathbb{R}^n_z \to \mathbb{R} \cup \{+\infty\}$ are closed, proper, convex functions and $A \in \mathbb{R}^{p \times n_\theta}$, $B \in \mathbb{R}^{p \times n_z}$, $c \in \mathbb{R}^p$.

The ADMM iterations to be performed to solve problem (6.1) are

$$\theta^{(k+1)} = \underset{\theta}{\operatorname{argmin}}\ \mathcal{L}(\theta, z^{(k)}, \delta^{(k)}), \tag{6.2}$$

$$z^{(k+1)} = \underset{z}{\operatorname{argmin}}\ \mathcal{L}(\theta^{(k+1)}, z, \delta^{(k)}), \tag{6.3}$$

$$\delta^{(k+1)} = \delta^{(k)} + \rho(A\theta^{(k+1)} + Bz^{(k+1)} - c), \tag{6.4}$$

where $k \in \mathbb{N}$ indicates the ADMM iteration, $\mathcal{L}$ is the augmented Lagrangian associated to (6.1), *i.e.*,

$$\mathcal{L}(\theta, z, \delta) = f(\theta) + g(z) + \delta' (A\theta + Bz - c) + \frac{\rho}{2} \|A\theta + Bz - c\|_2^2, \tag{6.5}$$

$\delta \in \mathbb{R}^p$ is the Lagrange multiplier and $\rho \in \mathbb{R}^+$ is a tunable parameter (see [23] for possible tuning strategies). Iterations (6.2)-(6.4) have to be run until a stopping criteria is satisfied, *e.g.*, the maximum number of iterations is attained.

6.1.1 ADMM for constrained convex optimization

Suppose that the problem to be addressed is

$$\begin{aligned} &\min_{\theta} && f(\theta) \\ &\text{s.t.} && \theta \in \mathcal{C}, \end{aligned} \tag{6.6}$$

with $\theta \in \mathbb{R}^{n_\theta}$, $f : \mathbb{R}^{n_\theta} \to \mathbb{R} \cup \{+\infty\}$ being a closed, proper, convex function and $\mathcal{C}$ being a convex set.
As explained in [23], the problem in (6.6) can be recast as in (6.1) through the introduction of the auxiliary variable $z \in \mathbb{R}^{n_\theta}$ and the indicator function of set $\mathcal{C}$

$$g(z) = \begin{cases} 0 & \text{if } z \in \mathcal{C}, \\ +\infty & \text{otherwise.} \end{cases} \tag{6.7}$$

Problem (6.6) thus becomes

$$\begin{aligned} \min_{\theta,z} \quad & f(\theta) + g(z), \\ \text{s.t.} \quad & \theta - z = 0, \end{aligned} \tag{6.8}$$

and the ADMM scheme to solve (6.8) is

$$\theta^{(k+1)} = \operatorname*{argmin}_{\theta} \mathcal{L}(\theta, z^{(k)}, \delta^{(k)}), \tag{6.9}$$

$$z^{(k+1)} = \mathcal{P}_{\mathcal{C}}(\theta^{(k+1)} + \delta^{(k)}), \tag{6.10}$$

$$\delta^{(k+1)} = \delta^{(k)} + \rho(\theta^{(k+1)} - z^{(k+1)}), \tag{6.11}$$

with the augmented Lagrangian $\mathcal{L}$ equal to

$$\mathcal{L}(\theta, z, \delta) = f(\theta) + g(z) + \delta'(\theta - z) + \frac{\rho}{2}\|\theta - z\|_2^2$$

6.1.2 ADMM for consensus problems

Consider the optimization problem

$$\min_{\theta^g} \sum_{n=1}^{N} f_n(\theta^g), \tag{6.12}$$

where $\theta^g \in \mathbb{R}^{n_\theta}$ and each term f_n of the objective function, $f_n : \mathbb{R}^{n_\theta} \to \mathbb{R} \cup \{+\infty\}$, is a proper, closed, convex function.
If N processors are available to solve (6.12), ADMM [23] can be used to reformulate this problem, so that each term of the cost function in (6.12) is handled by its own.

In particular, problem (6.12) is equivalent to

$$\begin{aligned} &\text{minimize} && \sum_{n=1}^{N} f_n(\theta_n) \\ &\text{subject to} && \theta_n - \theta^g = 0 \qquad n = 1, \dots, N, \end{aligned} \tag{6.13}$$

Thanks to the introduction of the consensus constraints, the cost function in (6.13) is separable.
The augmented Lagrangian corresponding to (6.13) is given by

$$\begin{aligned} &\mathcal{L}(\{\theta_n\}_{n=1}^N, \theta^g, \{\delta_n\}_{n=1}^N) = \sum_{n=1}^{N} \mathcal{L}_n(\theta_n, \theta^g, \delta_n), \\ &\mathcal{L}_n = f_n(\theta_n) + (\delta_n)'(\theta_n - \theta^g) + \frac{\rho}{2}\|\theta_n - \theta^g\|_2^2. \end{aligned} \tag{6.14}$$

and the ADMM iterations to solve (6.13) are

$$\theta_n^{(k+1)} = \underset{\theta_n}{\operatorname{argmin}}\ \mathcal{L}_n(\theta_n, \delta_n^{(k)}, \theta^{g,(k)}),\ n = 1, \dots, N \tag{6.15}$$

$$\theta^{g,(k+1)} = \frac{1}{N}\sum_{n=1}^{N}\left(\theta_n^{(k+1)} + \frac{1}{\rho}\delta_n^{(k)}\right), \tag{6.16}$$

$$\delta_n^{(k+1)} = \delta_n^{(k)} + \rho\left(\theta_n^{(k+1)} - \theta^{g,(k+1)}\right),\ n = 1, \dots, N. \tag{6.17}$$

On the one hand (6.15) and (6.17) can be carried out independently by each agent $n \in \{1, \dots, N\}$. On the other hand, equation (6.16) depends on all the updated local estimates. The global estimate should thus be updated in a *fusion center*, where all the local estimates are collected and merged.

6.2 Collaborative estimation: problem statement

Assume that (i) the measurements acquired by N agents are available and that (ii) the behavior of the N data-generating systems is described by the same model, with parameters $\theta_n \in \mathbb{R}^{n_\theta}$, with $n = 1, \dots, N$. As the agents share the same model, it is legitimate to assume that (iii) there exists a set of parameters $\theta^g \in \mathbb{R}^{n_g}$, with $n_g \leq n_\theta$, common to all the agents.

We aim at (i) retrieving *local* estimates of $\{\theta_n\}_{n=1}^N$, employing information available at the local level only, and (ii) identifying the *global* parameter θ^g at the cloud level, using the data collected from all the available sources. To accomplish these tasks (i) N local processors must be available and (ii) all the agents have to be connected to the cloud, where the data are merged are needed. The estimation problem to be solved can be cast into the separable optimization problem

$$\begin{aligned} \min_{\theta_n} \quad & \sum_{n=1}^{N} f_n(\theta_n) \\ \text{s.t.} \quad & F(\theta_n) = \theta^g, \\ & \theta_n \in \mathcal{C}_n,\ n = 1, \dots, N \end{aligned} \tag{6.18}$$

with $f_n : \mathbb{R}^{n_\theta} \to \mathbb{R} \cup \{+\infty\}$ being a closed, proper and convex function, $F : \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_g}$ is a nonlinear operator and $\mathcal{C}_n \subset \mathbb{R}^{n_\theta}$ is a convex set representing constraints on the local parameter values. Constraints on the value of the global parameter can be enforced if $\mathcal{C}_n = \mathcal{C} \cup \{\mathcal{C}_n \cap \breve{\mathcal{C}}\}$, with $\theta^g \in \mathcal{C}$.

Assume that the available data are the output/regressor pairs collected from each agent $n \in \{1, \dots, N\}$ over an horizon of length $T \in \mathbb{N}$, *i.e.*, $\{y_n(t), X_n(t)\}_{t=1}^T$. Relying on the hypothesis that the regressor/output relationship is well modeled as

$$y_n(t) = X_n(t)'\theta_n + e_n(t), \tag{6.19}$$

with $e_n(t) \in \mathbb{R}^{n_y}$ being a zero-mean additive noise independent of the regressor $X_n(t) \in \mathbb{R}^{n_\theta \times n_y}$, we focus on developing a recursive algorithm to solve (6.18) with the local cost functions given by

$$f_n(\theta_n) = \frac{1}{2} \sum_{t=1}^{T} \lambda_n^{T-t} \left\| y_n(t) - X_n(t)'\theta_n \right\|_2^2, \tag{6.20}$$

where the forgetting factors $\lambda_n \in (0, 1]$, for $n = 1, \dots, N$, are introduced to be able to estimate time-varying parameters. Different forgetting factors can be chosen for different agents.

6.3 Case study 1. Full consensus

Suppose that the problem to be solved is

$$\begin{aligned} &\text{minimize} && \sum_{n=1}^{N} f_n(\theta_n) \\ &\text{subject to} && \theta_n - \theta^g = 0 \qquad n = 1, \dots, N. \end{aligned}$$

i.e., we are aiming at achieving full consensus among N agents. The consensus constraint in (6.18) has thus to be modified as

$$F(\theta_n) = \theta^g \rightarrow \theta_n = \theta^g$$

and $\mathcal{C}_n = \mathbb{R}^{n_\theta}$. As we are focusing on the problem of collaborative least-squares estimation, we are interested in the particular case in which the local cost functions in (6.13) are equal to (6.20) .
The considered problem can be solved in a centralized fashion (C-RLS), transmitting all the outputs and the regressors to the cloud. However, our goal is to obtain estimates of the unknown parameters both (i) at a local level and (ii) on the cloud. With the objective of distributing the computation among the local processors and the cloud, we present an ADMM scheme to address the problem in (6.13). A similar approach has been used to develop a fully distributed scheme for consensus-based estimation over Wireless Sensor Networks (WSNs) in [75]. Our approach differs from the one introduced in [75] as we aim at exploiting the cloud to attain consensus and, at the same time, we want local estimates to be computed by each node.

The ADMM iterations to be performed to solved (6.13) are exactly equal to (6.14)-(6.17), *i.e.*,

$$\hat{\theta}_n(T)^{(k+1)} = \underset{\theta_n}{\operatorname{argmin}} \left\{ f_n(\theta_n) + (\delta_n^{(k)})'(\theta_n - \hat{\theta}^{g,(k)}) + \frac{\rho}{2}\|\theta_n - \hat{\theta}^{g,(k)}\|_2^2 \right\},$$

$$\hat{\theta}^{g,(k+1)} = \frac{1}{N}\sum_{n=1}^{N}\left(\theta_n^{(k+1)} + \frac{1}{\rho}\delta_n^{(k)}\right),$$

$$\delta_n^{(k+1)} = \delta_n^{(k)} + \rho\left(\hat{\theta}_n^{(k+1)}(T) - \hat{\theta}^{g,(k+1)}\right), \; n = 1, \dots, N$$

with the cost functions f_n defined as in (6.14). The dependence on T of the local estimates is stressed to underline that only the updates of $\hat{\theta}_n$ are directly influenced by the current measurements.

As equation (6.16)-(6.17) are independent from the specific choice of $f_n(\theta_n)$, we focus on the update of the local estimates (6.15), with the ultimate goal of finding recursive updates for $\hat{\theta}_n$. Thanks to the characteristics of the chosen local cost functions, the closed-form solution for the problem in (6.15) is given by

$$\hat{\theta}_n^{(k+1)}(T) = \phi_n(T)\left(\mathcal{Y}_n(T) - \delta_n^{(k)} + \rho\hat{\theta}^{g,(k)}\right), \tag{6.21}$$

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau), \quad t = 1, \dots, T, \tag{6.22}$$

$$\mathcal{X}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau)(X_n(\tau))', \quad t = 1, \dots, T, \tag{6.23}$$

$$\phi_n(t) = (\mathcal{X}_n(t) + \rho I_{n_\theta})^{-1}, \quad t = 1, \dots, T. \tag{6.24}$$

With the aim of obtaining recursive formulas to update $\hat{\theta}_n$, we introduce the local estimate obtained at $T-1$, which is given by

$$\hat{\theta}_n(T-1) = \phi_n(T-1)\left(\mathcal{Y}_n(T-1) + \rho\hat{\theta}^g(T-1) - \delta_n(T-1)\right), \tag{6.25}$$

with $\delta_n(T-1)$ and $\hat{\theta}^g(T-1)$ denoting the Lagrange multiplier and the global estimate computed at $T-1$, respectively.
Consider the matrix $\phi_n(T)$ defined in equation (6.24). The inverse of matrix $\phi_n(T)$ is given by

$$\phi_n(T)^{-1} = \mathcal{X}_n(T) + \rho I_{n_\theta}.$$

Based on (6.24), it can be proven that $\phi_n(T)^{-1}$ can be computed recursively, as a function of $\phi_n(T-1)^{-1}$. In particular:

$$\begin{aligned}
&\phi_n(T)^{-1} = \mathcal{X}_n(T) + \rho I_{n_\theta} = \\
&= \lambda_n \mathcal{X}_n(T-1) + X_n(T)(X_n(T))' + \rho I_{n_\theta} = \\
&= \lambda_n \phi_n(T-1)^{-1} + X_n(T)(X_n(T))' + (1-\lambda_n)\rho I_{n_\theta}.
\end{aligned} \tag{6.26}$$

Introducing the extended regressor vector $\tilde{X}_n(T)$

$$\tilde{X}_n(T) = \begin{bmatrix} X_n(T) & \sqrt{(1-\lambda_n)\rho} I_{n_\theta} \end{bmatrix} \in \mathbb{R}^{n_\theta \times (n_y + n_\theta)}, \tag{6.27}$$

the inverse of $\phi_n(T)$ in (6.26) can be further simplified as

$$\phi_n(T)^{-1} = \lambda_n \phi_n(T-1)^{-1} + \tilde{X}_n(T)(\tilde{X}_n(T))'.$$

Applying the matrix inversion lemma [114], the resulting recursive formulas to update ϕ_n are

$$\mathcal{R}_n(T) = \lambda_n I_{(n_y+n_\theta)} + (\tilde{X}_n(T))' \phi_n(T-1) \tilde{X}_n(T), \tag{6.28}$$

$$K_n(T) = \phi_n(T-1)\tilde{X}_n(T)(\mathcal{R}_n(T))^{-1}, \tag{6.29}$$

$$\phi_n(T) = \lambda_n^{-1} \left(I_{n_\theta} - K_n(T)(\tilde{X}_n(T))' \right) \phi_n(T-1), \tag{6.30}$$

with the gain K_n and matrix ϕ_n are updated as in standard RLS [72], substituting the regressor X_n with $\tilde{X}_n$ and increasing the dimension of the identity matrix in (6.28). Only when $\lambda_n = 1$ the regressor X_n and $\tilde{X}_n$ are equal.
Observe that (6.28)-(6.30) are independent from k and, consequently, $\{\mathcal{R}_n, K_n, \phi_n\}_{n=1}^N$ can be updated once per step t.

Adding and subtracting

$$\lambda_n \phi_n(T) \left[\rho \hat{\theta}^g(T-1) - \delta_n(T-1) \right]$$

to (6.21), the solution of (6.15) is equal to

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T), \tag{6.31}$$

with

$$\hat{\theta}_n^{RLS}(T) = \phi_n(T) \left\{ \lambda_n \left(\mathcal{Y}_n(T-1) + \rho \hat{\theta}^g(T-1) - \delta_n(T-1) \right) \right\} + \\ + \phi_n(T) X_n(T) y_n(T), \tag{6.32}$$

$$\hat{\theta}_n^{ADMM,(k+1)}(T) = \phi_n(T) \left[\rho \Delta_{g,\lambda_n}^{(k+1)}(T) - \Delta_{\lambda_n}^{(k+1)}(T) \right], \tag{6.33}$$

and

$$\Delta_{g,\lambda_n}^{k+1}(T) = \hat{\theta}^{g,(k)} - \lambda_n \hat{\theta}^g(T-1), \tag{6.34}$$

$$\Delta_{\lambda_n}^{(k+1)}(T) = \delta_n^{(k)} - \lambda_n \delta_n(T-1). \tag{6.35}$$

Observe that (6.33) is independent from the observations $\{y_n(t), X_n(t)\}$, for $t = 1, \dots, T$, while (6.32) depends on $\mathcal{Y}_n(T-1)$. To obtain recursive formulas to update $\hat{\theta}_n$, the dependence of (6.32) on $\mathcal{Y}_n(T-1)$ should be eliminated.
Exploiting (6.30) and (6.25), $\hat{\theta}_n^{RLS}(T)$ in (6.32) is given by

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n(T-1) + K_n(T)(\tilde{y}_n(T) - \tilde{X}_n(T)'\hat{\theta}_n(T-1)), \tag{6.36}$$

where we have introduced the extended measurement vector

$$\tilde{y}_n(T) = \begin{bmatrix} (y_n(T))' & 0_{1\times n_g} \end{bmatrix}'.$$

and we have used the equality $\phi_n(T)\tilde{X}_n(T) = K_n(T)$. Similarly to what is done for standard RLS, this equality can be proven using the matrix inversion lemma as follows

$$\begin{aligned}
&\phi_n(T)\tilde{X}_n(T) = \lambda_n^{-1}\left(I_{n_\theta} - K_n(T)(\tilde{X}_n(T))'\right)\phi_n(T-1)\tilde{X}_n(T) = \\
&= \lambda_n^{-1}\left(I_{n_\theta} - \phi_n(T-1)\tilde{X}_n(T)(\mathcal{R}_n(T))^{-1}(\tilde{X}_n(T))'\right)\phi_n(T-1)\tilde{X}_n(T) = \\
&= \phi_n(T-1)\tilde{X}_n(T)\left(\lambda_n^{-1}I_{n_\theta} - \lambda_n^{-1}(\mathcal{R}_n(T))^{-1}(\tilde{X}_n(T))'\phi_n(T-1)\tilde{X}_n(T)\right) = \\
&= \phi_n(T-1)\tilde{X}_n(T)\left(\lambda_n^{-1}I_{n_\theta} - \lambda_n^{-1}(\mathcal{R}_n(T))^{-1}(\tilde{X}_n(T))'\phi_n(T-1)\tilde{X}_n(T)\right) = \\
&= \phi_n(T-1)\tilde{X}_n(T)\left(\lambda_n I_{n_\theta} + (\tilde{X}_n(T))'\phi_n(T-1)\tilde{X}_n(T)\right)^{-1} = K_n(T),
\end{aligned}$$

where the matrix inversion lemma [114] and (6.29)-(6.30) are used.

The update for $\hat{\theta}_n^{ADMM}$ (6.33) depends on both the values of the Lagrange multipliers and the global estimates, while $\hat{\theta}_n^{RLS}$ in (6.36) is computed only on the basis of the previous local estimate and the current measurements. As $\hat{\theta}^g$ is updated using information collected from all the agents, $\hat{\theta}^{g,(k)}$ should be computed on the cloud. Due to the dependence of (6.17) on $\hat{\theta}^{g,(k+1)}$, also the Lagrange multipliers and $\hat{\theta}_n^{ADMM}$ should

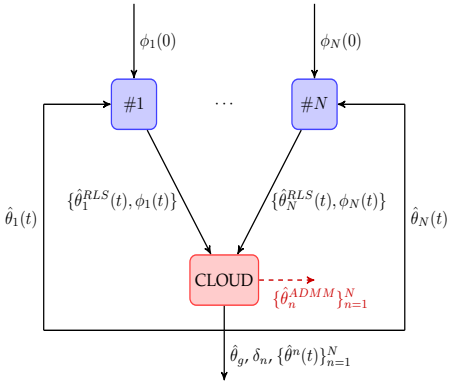


Figure 66: ADMM-RLS. Schematic of the information exchanges between the agents and the cloud using an Node-to-Cloud-to-Node (N2C2N) communication scheme.

be updated in the cloud to reduce the transmission complexity. Instead, the partial estimates $\hat{\theta}_n^{RLS}$, $n = 1, \ldots, N$, can be computed by the local processors.

The presented method is summarized in Algorithm 14 and Figure 66, and it allows us to obtain estimates both at the (i) agent and (ii) cloud level. Thanks to the independence of (6.36) from k, $\hat{\theta}_n^{RLS}$ can be updated once per step t. Furthermore, looking at (6.28)-(6.30) and (6.36), it can be noticed that $\hat{\theta}_n^{RLS}$ is updated through standard RLS, with the exceptions that the update depends on the previous estimate $\hat{\theta}_n(t-1)$ computed on the cloud, instead of depending on $\hat{\theta}_n^{RLS}(t-1)$, and that the output/regressor pair $\{y_n(t), X_n(t)\}$ is replaced with $\{\tilde{y}_n(t), \tilde{X}_n(t)\}$. The proposed method can thus be easily integrated with pre-existing RLS estimators already available locally.

Algorithm 14 requires the initialization of the local and global estimates. If some data are available to be processed in a batch mode, $\hat{\theta}_n(0)$ can be chosen as the best linear model, *i.e.*,

$$\hat{\theta}_n(0) = \underset{\theta_n}{\operatorname{argmin}} \sum_{t=1}^{\tau} \|y_n(t) - X_n(t)'\theta\|_2^2$$

Algorithm 14 ADMM-RLS for full consensus (N2C2N)

Input: Data flow $\{X_n(1), y_n(1)\}$,$\{X_n(2), y_n(2)\}, \ldots$, initial matrices $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n(0)$, initial dual variables $\delta_{n,\mathrm{o}}$, $n = 1, \ldots, N$, initial global estimate $\hat{\theta}_\mathrm{o}^g$, parameter $\rho \in \mathbb{R}^+$.

1. **for** $t = 1, 2, \ldots$ **do**

   Local

   1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $\tilde{X}_n(t)$ as in (6.27);

   1.1.2. **compute** $K_n(t)$ and $\phi_n(t)$ with (6.29) - (6.30);

   1.1.3. **compute** $\hat{\theta}_n^{RLS}(t)$ with (6.36);

   1.2. **end for**;

   Global

   1.1. **do**

   1.1.1. **compute** $\hat{\theta}_n^{ADMM,(k+1)}(t)$ with (6.33), $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}^{g,(k+1)}(t)$ with (6.16);

   1.1.3. **compute** $\delta_n^{(k+1)}$ with (6.17), $n = 1, \ldots, N$;

   1.2. **until** a stopping criteria is satisfied (*e.g.*, maximum number of iterations attained);

2. **end**.

Output: Estimated global parameters $\hat{\theta}^g(t)$, estimated local parameters $\hat{\theta}_n(t)$, $n = 1, \ldots, N$.

and $\hat{\theta}^g(0)$ can be computed as the mean of $\{\hat{\theta}_n(0)\}_{n=1}^N$. Moreover, the matrices ϕ_n, $n = 1, \ldots, N$, can be initialized as $\phi_n(0) = \gamma I_{n_\theta}$, with $\gamma > 0$.

Remark 5 *The chosen implementation requires $\hat{\theta}_n^{RLS}$ and ϕ_n to be transmitted from the local processors to the cloud at each step, while the cloud has to communicate $\hat{\theta}_n$ to all the agents.* ■

6.3.1 Example 1

Suppose that N data-generating systems are described by the following model

$$y_n(t) = 0.9y_n(t-1) + 0.4u_n(t-1) + e_n(t), \tag{6.37}$$

where $y_n(t) \in \mathbb{R}$, $X_n(t) = \begin{bmatrix} y_n(t-1) & u_n(t-1) \end{bmatrix}'$, u_n is a measured sequence of i.i.d. elements uniformly distributed in the interval $[2,\ 3]$ and $e_n \sim \mathcal{N}(0, R_n)$ is a white noise sequence, with $\{R_n \in \mathbb{N}\}_{n=1}^{N}$ randomly chosen in the interval $[1,\ 3]$. Evaluating the effect of the noise on the output y_n through the Signal-to-Noise Ratio SNR_n

$$\text{SNR}_n = 10 \log \frac{\sum_{t=1}^{T} \left(y_n(t) - e_n(t)\right)^2}{\sum_{t=1}^{T} e_n(t)^2} \text{ dB} \tag{6.38}$$

the chosen covariance matrices yield SNR_n in the interval $[7.8\ 20.8]$ dB, $n = 1, \ldots, N$. Note that the model (6.37) can be equivalently written as

$$y_n(t) = (X_n(t))'\theta^g + e_n(t) \text{ with } \theta^g = \begin{bmatrix} 0.9 & 0.4 \end{bmatrix}'.$$

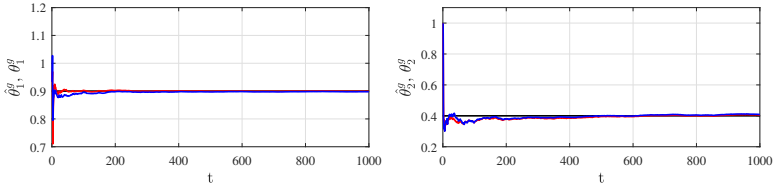
We initialize ϕ_n as $\phi_n(0) = 0.1I_{n_\theta}$, while $\hat{\theta}_n(0)$ and $\hat{\theta}_o^g$ are sampled from the distributions $\mathcal{N}(\hat{\theta}^g, 2I_{n_\theta})$ and $\mathcal{N}(\hat{\theta}^g, I_{n_\theta})$, respectively, $\{\lambda_n = 1\}_{n=1}^{N}$ and $\rho = 0.1$. The performance of ADMM-RLS are assessed for different values of N and T. Moreover, the retrieved estimates are compared to the ones obtained in a fully centralized fashion (C-RLS).
The accuracy of the estimate $\hat{\theta}^g$ is assessed through the Root Mean Square Error (RMSE), computed as

$$\text{RMSE}_i^g = \sqrt{\frac{\sum_{t=1}^{T} \left(\theta_i^g - \hat{\theta}_i^g(t)\right)^2}{T}},\ i = 1, \ldots, n_g. \tag{6.39}$$

As it can be noticed from the RMSEs reported in Table 24, the estimates tend to be more accurate if the number of local processors N and the estimation horizon T increase. This result can be expected as an increase in N and/or T corresponds to an increase in the data available for estimation. In the specific case $N = 100$ and $T = 1000$, both the RMSEs obtained solving the estimation problem in a fully centralized fashion

Table 24: Example 1: ADMM-RLS: $\|\text{RMSE}^g\|_2$

N \ T	**10**	$\mathbf{10^2}$	$\mathbf{10^3}$	$\mathbf{10^4}$
2	1.07	0.33	0.16	0.10
10	0.55	0.22	0.09	0.03
$\mathbf{10^2}$	0.39	0.11	0.03	0.01



(a) θ_1^g vs $\hat{\theta}_1^g$ (b) θ_2^g vs $\hat{\theta}_2^g$

Figure 67: Example 1. Model parameters. True (black), centralized approach (red), ADMM-RLS (blue). The estimates obtained with C-RLS and ADMM-RLS are barely distinguishable.

(C-RLS) and the one resulting from the use of ADMM-RLS are equal to 0.03. The estimates obtained with C-RLS and ADMM-RLS are further compared in Figure 67. As expected, the estimates obtained with the different methods are barely distinguishable for most of the estimation horizon.

Non-informative agents

Assume that some of the available data sources are non-informative, *i.e.*, some systems are not excited enough to be able to retrieve an accurate estimate of all the unknown parameters [72] locally. Null input sequences and white noise sequences characterized by $R_n = 10^{-8}$ are used to simulate the behavior of the $N_{ni} \leq N$ non-informative agents.
Consider $N = 100$ systems and an estimation horizon of length $T = 5000$. The performance of ADMM-RLS are studied under the hypothesis that an increasing number N_{ni} of systems is non-informative. Table 25 reports the RMSEs obtained for different values of N_{ni}. It can be noticed that the quality of the estimate starts to deteriorates only when half of the available systems are non-informative.

Table 25: Example 1. $\|\text{RMSE}^g\|_2$ vs N_{ni}

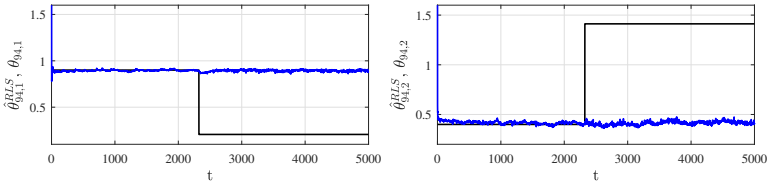
	N_{ni}			
	1	10	20	50
$\|\text{RMSE}^g\|_2$	0.02	0.02	0.02	0.03

Table 26: Example 1. ADMM-RLS: $\|\text{RMSE}^g\|_2$ vs N_f

	N_f			
	1	10	20	50
$\|\text{RMSE}^g\|_2$	0.03	0.03	0.03	0.04

Agents failure

Consider a set of $N = 100$ agents and an estimation horizon $T = 5000$. Suppose that, due to a change in the behavior of N_f local agents, the parameters of their models suddenly assume different values with respect to $\begin{bmatrix}0.9 & 0.4\end{bmatrix}$. We study the performance of ADMM-RLS when the parameters change at an (unknown) instant t_n, randomly chosen in the interval $t_n \in [1875,\ 3750]$. For $t \geq t_n$, the parameters $\theta_{n,1}$ and $\theta_{n,2}$ are sampled from the uniform distributions $\mathcal{U}_{[0.2\ 0.21]}$ and $\mathcal{U}_{[1.4\ 1.43]}$, respectively. The forgetting factors λ_n, $n = 1, \ldots, N$, are set to 0.99.
The performance of ADMM-RLS are initially assessed considering an increasing number of systems subject to failure. From the RMSEs reported in Table 26, it can be noticed that the failure of the agents seems not to influence the accuracy of the obtained global estimates if $N_f \neq 50$. The use of ADMM-RLS thus allows to compute accurate global estimates even when some of the agent experience a failure.
In Figure 68, the local estimate $\hat{\theta}_{94}^{RLS}$ is compared with the actual value of θ_{94}, with the 94th system being subject to failure. Observe that the local estimates does not follow the change in the actual parameters, but they tend to reproduce the behavior of the global estimates. This does not allow to identify changes on the local parameters looking at $\hat{\theta}_n^{RLS}$. However, this is crucial when the algorithm is used for monitoring/diagnostic purposes. Even if the change in the local parameters is not directly identifiable from $\hat{\theta}_n^{RLS}$, it can be detected looking at other indicators, *e.g.*, the condition number of ϕ_{94} (see Figure 69).



(a) $\theta_{94,1}$ vs $\hat{\theta}_{94,1}^{RLS}$ (b) $\theta_{94,2}$ vs $\hat{\theta}_{94,2}^{RLS}$

Figure 68: Example 1. True vs estimated model parameters $\hat{\theta}_{94}$. True (black), $\hat{\theta}_{94}^{RLS}$ obtained with ADMM-RLS (blue).

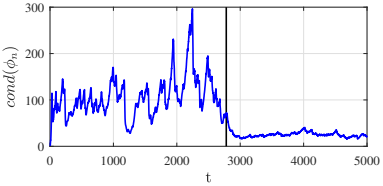


Figure 69: Example 1. Condition number of ϕ_n, with $n = 94$.

6.4 Case study 2. Partial consensus

Consider the more general hypothesis that there exists a parameter vector $\theta^g \in \mathbb{R}^{n_g}$, with $n_g \leq n_\theta$ such that:

$$P\theta_n = \theta^g \;\; \forall n \in \{1, \dots, N\}, \tag{6.40}$$

where $P \in \mathbb{R}^{n_g \times n_\theta}$ is a matrix assumed to be known a priori. The problem we want to solve is then given by

$$\begin{aligned} \min_{\{\theta_n\}_{n=1}^N} \quad & \sum_{n=1}^{N} f_n(\theta_n) \\ \text{s.t.} \quad & P\theta_n = \theta^g, \;\; n = 1, \dots, N, \end{aligned} \tag{6.41}$$

with f_n defined as in (6.20). Note that (6.41) corresponds to (6.18) with the consensus constraint modified as

$$F(\theta_n) = \theta^g \rightarrow P\theta_n = \theta^g.$$

The considered consensus constraint allows us to enforce consensus over a linear combination of the components of θ_n. Through proper choices of

P, different settings can be considered, *e.g.*, if $P = I_{n_\theta}$ then $\theta_n = \theta^g$ and thus (6.41) is equal to (6.12). We can also enforce consensus only over some components of θ_n, so that some of the unknowns are assumed to be *global*, while others assume different values for each agent.

The ADMM iterations to solve problem (6.41) are given by

$$\hat{\theta}_n^{(k+1)}(T) = \underset{\theta_n}{\operatorname{argmin}}\ \mathcal{L}(\theta_n, \hat{\theta}^{g,(k)}, \delta_n^{(k)}), \tag{6.42}$$

$$\hat{\theta}^{g,(k+1)} = \underset{\theta^g}{\operatorname{argmin}}\ \mathcal{L}(\{\hat{\theta}_n^{(k+1)}(T)\}_{n=1}^N, \theta^g, \{\delta_n^{(k)}\}_{n=1}^N), \tag{6.43}$$

$$\delta_n^{(k+1)} = \delta_n^{(k)} + \rho(P\hat{\theta}_n^{(k+1)}(T) - \hat{\theta}^{g,(k+1)}), \tag{6.44}$$

with $k \in \mathbb{N}$ indicating the ADMM iteration, $\rho \in \mathbb{R}^+$ being a tunable parameter, $\delta_n \in \mathbb{R}^{n_g}$ representing the Lagrange multiplier and the augmented Lagrangian $\mathcal{L}$ given by

$$\mathcal{L} = \sum_{n=1}^{N} \left\{ f_n(\theta_n) + \delta_n'(P\theta_n - \theta^g) + \frac{\rho}{2} \left\| P\theta_n - \theta^g \right\|_2^2 \right\}. \tag{6.45}$$

Note that the dependence on T is explicitly indicated only for the local estimates $\hat{\theta}_n$, as they are the only quantities directly affected by the measurement and the regressor at T.
Consider the update of the estimate $\hat{\theta}^g$. The closed form solution for (6.43) is

$$\hat{\theta}^{g,(k+1)} = \frac{1}{N} \sum_{n=1}^{N} \left(P\hat{\theta}_n^{(k+1)}(T) + \frac{1}{\rho}\delta_n^{(k)} \right). \tag{6.46}$$

The estimate of the global parameter is thus updated through the combination of the mean of $\{\delta_n\}_{n=1}^N$ and the mean of $\{P\hat{\theta}_n^{(k+1)}(T)\}_{n=1}^N$.
Consider the update for the estimated of the local parameters. The close

form solution for (6.42) is given by:

$$\hat{\theta}_n^{(k+1)}(T) = \phi_n(T)\left\{\mathcal{Y}_n(T) + P'(\rho\hat{\theta}^{g,(k)} - \delta_n^{(k)})\right\}, \tag{6.47}$$

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau), \quad t = 1, \ldots, T,$$

$$\mathcal{X}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau)(X_n(\tau))', \quad t = 1, \ldots, T,$$

$$\phi_n(t) = \left(\mathcal{X}_n(t) + \rho P'P\right)^{-1}, \quad t = 1, \ldots, T. \tag{6.48}$$

Consider $\hat{\theta}_n(T-1)$, defined as

$$\hat{\theta}_n(T-1) = \phi_n(T-1)\left(\mathcal{Y}_n(T-1) + P'(\rho\hat{\theta}^g(T-1) - \delta_n(T-1))\right), \tag{6.49}$$

where $\phi_n(T-1)$ is defined as in equation (6.48), and $\hat{\theta}^g(T-1)$ and $\delta_n(T-1)$ are the global estimate and the Lagrange multiplier obtained at $T-1$, respectively.
Observe that the following equalities hold

$$\phi_n(T) = \left(\lambda_n \phi_n(T-1)^{-1} + X_n(T)X_n(T)' + \rho(1-\lambda_n)P'P\right)^{-1}.$$

Introducing the extended regressor

$$\tilde{X}_n(T) = \begin{bmatrix} X_n(T) & \sqrt{\rho(1-\lambda_n)}P' \end{bmatrix} \in \mathbb{R}^{n_\theta \times (n_y + n_g)} \tag{6.50}$$

and applying the matrix inversion lemma [114], it can be proven that ϕ_n can be updated as

$$\mathcal{R}_n(T) = \lambda_n I_{(n_y+n_g)} + (\tilde{X}_n(T))'\phi_n(T-1)\tilde{X}_n(T), \tag{6.51}$$

$$K_n(T) = \phi_n(T-1)\tilde{X}_n(T)\left(\mathcal{R}_n(T)\right)^{-1}, \tag{6.52}$$

$$\phi_n(T) = \lambda_n^{-1}(I_{n_\theta} - K_n(T)(\tilde{X}_n(T))')\phi_n(T-1). \tag{6.53}$$

Note that the equations (6.51)-(6.53) are similar to (6.28)-(6.30), with differences due to the new definition of the extended regressor.

Adding and subtracting

$$\lambda_n \phi_n(T) P' \left(\rho\hat{\theta}^g(T-1) - \delta_n(T-1)\right)$$

to (6.47), it can be shown that $\hat{\theta}_n^{(k+1)}$ can be computed as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T), \tag{6.54}$$

with

$$\begin{aligned}\hat{\theta}_n^{RLS}(T) = \phi_n(T)\lambda_n \left\{ \mathcal{Y}_n(T-1) + \rho P' \hat{\theta}(T-1) - P'\delta_n(T-1) \right\} + \\ + \phi_n(T)\tilde{X}_n(T)y_n(T),\end{aligned} \tag{6.55}$$

and

$$\begin{aligned}\hat{\theta}_n^{ADMM,(k+1)}(T) &= \phi_n(T)P' \left(\rho\Delta_{g,\lambda_n}^{(k+1)}(T) - \Delta_{\lambda_n}^{(k+1)} \right), \\ \Delta_{g,\lambda_n}^{k+1}(T) &= \hat{\theta}^{g,(k)} - \lambda_n \hat{\theta}^g(T-1), \\ \Delta_{\lambda_n}^{(k+1)}(T) &= \delta_n^{(k)} - \lambda_n \delta_n(T-1).\end{aligned} \tag{6.56}$$

Observe that, as for equations (6.16) and (6.46), the updates of the $\hat{\theta}_n^{ADMM,(k)}$ in (6.56) differs from the one in (6.33) because of the presence of P.

Accounting for the definition of $\phi_n(T-1)$, exploiting the equality $K_n(T) = \phi_n(T)\tilde{X}_n(T)$ and introducing the extended measurement vector

$$\tilde{y}_n(T) = \begin{bmatrix} y_n(T)' & O_{1\times n_g} \end{bmatrix}',$$

the local estimate $\hat{\theta}_n^{RLS}$ in (6.56) can be updated as

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n(T-1) + K_n(T)(\tilde{y}_n(T) - (\tilde{X}_n(T))'\hat{\theta}_n(T-1)).$$

As for the method tailored to attain full consensus presented in Section 6.3, both $\hat{\theta}^g$ and δ_n should be updated on the cloud. As a consequence, also $\hat{\theta}_n^{ADMM}$ should be updated on the cloud, due to its dependence on both $\hat{\theta}^g$ and δ_n. On the other hand, $\hat{\theta}_n^{RLS}$ can be updated by the local processors. Moreover, as equation (6.57) is independent from k, $\hat{\theta}_n^{RLS}$ can be updated once per time step t.

The approach is outlined in Algorithm 15 and the employed Node-to-Cloud-to-Node transmissions is reported in Figure 66. Due to the employed communication scheme, the observations made in Section 6.3 with respect to the information exchange between the nodes and the cloud hold also in this case.

Algorithm 15 ADMM-RLS algorithm for partial consensus

Input: Data flow $\{X_n(1), y_n(1)\}$,$\{X_n(2), y_n(2)\}, \ldots$, initial matrices $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n(0)$, initial dual variables $\delta_{n,\mathrm{o}}$, forgetting factors λ_n, $n = 1, \ldots, N$, initial global estimate $\hat{\theta}^g_{\mathrm{o}}$, parameter $\rho \in \mathbb{R}^+$.

1. **for** $t = 1, 2, \ldots$ **do**

   Local

   1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $\tilde{X}_n(t)$ with (6.50);

   1.1.2. **compute** $K_n(t)$ and $\phi_n(t)$ with (6.52) - (6.53);

   1.1.3. **compute** $\hat{\theta}^{RLS}_n(t)$ with (6.57);

   1.2. **end for**;

   Global

   1.1. **do**

   1.1.1. **compute** $\hat{\theta}^{ADMM,(k+1)}_n(t)$ with (6.56), $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}^{(k+1)}_n(t)$ with (6.54), $n = 1, \ldots, N$;

   1.1.3. **compute** $\hat{\theta}^{g,(k+1)}$ with (6.46);

   1.1.4. **compute** $\delta^{(k+1)}_n$ with (6.44), $n = 1, \ldots, N$;

   1.2. **until** a stopping criteria is satisfied (*e.g.*, maximum number of iterations attained);

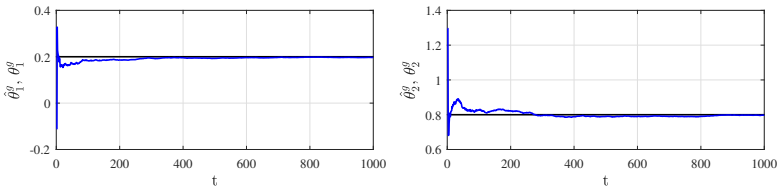
2. **end**.

Output: Estimated global parameters $\hat{\theta}^g(t)$, estimated local parameters $\hat{\theta}_n(t)$, $n = 1, \ldots, N$.

6.4.1 Example 2

Consider $N = 100$ dynamical systems observed over an estimation horizon of length $T = 1000$. Assume that the behavior of the N dynamical systems is modeled as

$$y_n(t) = \theta^g_1 y_n(t-1) + \theta_{n,2} y_n(t-2) + \theta^g_2 u_n(t-1) + e_n(t), \tag{6.57}$$



(a) θ_1^g vs $\hat{\theta}_1^g$ (b) θ_2^g vs $\hat{\theta}_2^g$

Figure 70: Example 2. $\hat{\theta}^g$ vs θ^g. True parameter (black), estimates obtained with ADMM-RLS (blue).

where $\theta^g = \begin{bmatrix} 0.2, & 0.8 \end{bmatrix}'$ and $\theta_{n,2}$ is sampled from a normal distribution $\mathcal{N}(0.4, 0.0025)$, so that it is different for the N systems. The white noise sequence $e_n \sim \mathcal{N}(0, R_n)$, where $R_n \in \mathbb{N}$ and sampled in the interval $[1\ 20]$ yields SNR in $[3.1, 14.6]$ dB (see (6.38)).

We choose the same initial parameters used in Section 6.3[1]. Figure 70 shows that $\hat{\theta}^g$ obtained with ADMM-RLS converges to the actual value of the global parameters. To further assess the performances of ADMM-RLS, θ_n, $\hat{\theta}_n$ and $\hat{\theta}_n^{RLS}$ obtained for the 5th system[2], are compared in Figure 71. It can thus be seen that the difference between $\hat{\theta}_n^{RLS}$ and $\hat{\theta}_n$ is only noticeable at the beginning of the estimation horizon, but then $\hat{\theta}_n^{RLS}$ and $\hat{\theta}_n$ are barely distinguishable.

Non-informative agents

Suppose that, among the $N = 100$ systems described by the model in (6.57), $N_{ni} = 20$ randomly chosen agents are non-informative[3].

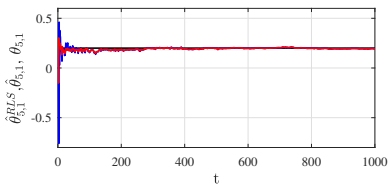
As it can be observed from the estimates reported in Figure 72, $\{\hat{\theta}_i^g\}_{i=1}^2$ converge to the actual values of the global parameters even if 20% of the systems provide non-informative data.

The local estimates $\hat{\theta}_{n,2}$ for the 8th and 65th system ($SNR_{65} \approx 6$ dB) are reported in Figure 73. As the 8th system is among the ones with a non exciting input, $\hat{\theta}_{8,2} = \hat{\theta}_{8,2}(0)$ over the estimation horizon. Instead, $\hat{\theta}_{65,2}$

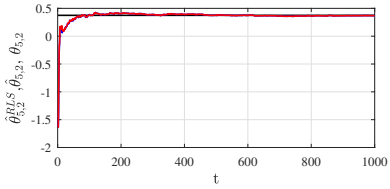
[1]The matrices$\phi_n(0) = 0.1 I_{n_\theta}$, while $\hat{\theta}_n(0)$ and $\hat{\theta}_o^g$ are sampled from the distributions $\mathcal{N}(\hat{\theta}^g, 2I_{n_\theta})$ and $\mathcal{N}(\hat{\theta}^g, I_{n_\theta})$, respectively, $\{\lambda_n = 1\}_{n=1}^N$ and $\rho = 0.1$.

[2]$\text{SNR}_5 = 8.9$ dB.

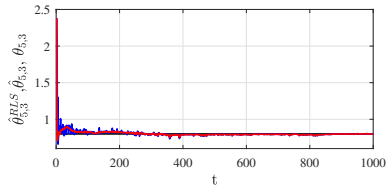
[3]Non-informative agents are characterized by null input sequences u_n and $R_n = 10^{-8}$.



(a) $\theta_{5,1}$ vs $\hat{\theta}_{5,1}$ and $\hat{\theta}^{RLS}_{5,1}$

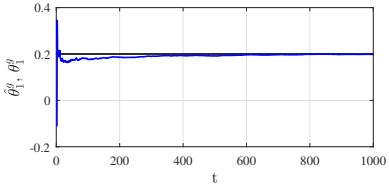


(b) $\theta_{5,2}$ vs $\hat{\theta}_{5,2}$ and $\hat{\theta}^{RLS}_{5,2}$

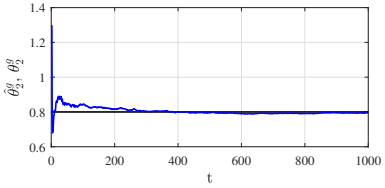


(c) $\theta_{5,3}$ vs $\hat{\theta}_{5,3}$ and $\hat{\theta}^{RLS}_{5,3}$

Figure 71: Example 2. Local parameter $\theta_{5,2}$. True (black), local estimate on the cloud $\hat{\theta}_5$ (blue), $\hat{\theta}_5^{RLS}$ (red). The two estimates are barely distinguishable.



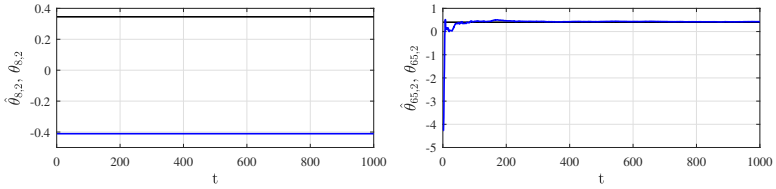
(a) θ_1^g vs $\hat{\theta}_1^g$



(b) θ_2^g vs $\hat{\theta}_2^g$

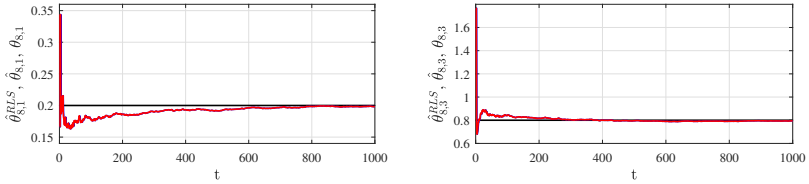
Figure 72: Example 2. $\hat{\theta}^g$ vs θ^g. True (black), estimates obtained with ADMM-RLS (blue).

converges to the actual value of $\theta_{65,2}$. Even if the purely local parameter is not retrieved from the data, note that $\theta_{8,1}$ and $\theta_{8,3}$ are accurately estimated as it can be seen from Figure 74. We can thus conclude that the proposed method "forces" the estimates of the global components of θ_n to follow $\hat{\theta}^g$, which is thus estimated automatically discarding the contributions from the systems that lacked excitation.



(a) $\theta_{8,2}$ vs $\hat{\theta}_{8,2}$ (b) $\theta_{65,2}$ vs $\hat{\theta}_{65,2}$

Figure 73: Example 2. Local parameters $\theta_{n,2}$, $n = 8, 65$. True (black), estimates obtained with ADMM-RLS (blue).



(a) $\theta_{8,1}$ vs $\hat{\theta}_{8,1}$ and $\hat{\theta}_{8,1}^{RLS}$ (b) $\theta_{8,3}$ vs $\hat{\theta}_{8,3}$ and $\hat{\theta}_{8,3}^{RLS}$

Figure 74: Example 2. Local parameters $\theta_{8,i}$, $i = 1, 3$. True (black), $\hat{\theta}_{8,i}^{RLS}$ (blue), local estimates computed on the cloud $\hat{\theta}_{8,i}$ (red).

6.5 Case study 3. Constrained Partial consensus

Suppose that the values of the local parameters θ_n, $n = 1, \ldots, N$, are constrained to the sets $\mathcal{C}_n$. With the objective of reaching partial consensus among the agents, the problem to be solved can be formulated as

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} f_n(\theta_n) \\ \text{s.t.} \quad & P\theta_n = \theta, \;\; n = 1, \ldots, N, \\ & \theta_n \in \mathcal{C}_n, \;\; n = 1, \ldots, N. \end{aligned} \tag{6.58}$$

To use ADMM to solve (6.58), this problem is modified as

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} \{f_n(\theta_n) + g_n(z_n)\} \\ \text{s.t.} \quad & P\theta_n = \theta^g \;\; n = 1, \ldots, N \\ & \theta_n = z_n, \;\; n = 1, \ldots, N, \end{aligned} \tag{6.59}$$

where we introduce the indicator functions $\{g_n\}_{n=1}^N$ of the sets $\{\mathcal{C}_n\}_{n=1}^N$ (defined as in (6.7)) and the auxiliary variables $\{z_n \in \mathbb{R}^{n_\theta}\}_{n=1}^N$. Given the augmented Lagrangian associated with (6.59), *i.e.*,

$$\mathcal{L} = \sum_{n=1}^{N} \{f_n(\theta_n) + g_n(z_n) + \delta'_{n,1}(\theta_n - z_n) + \delta'_{n,2}(P\theta_n - \theta^g) + \\ + \frac{\rho_1}{2}\|\theta_n - z_n\|_2^2 + \frac{\rho_2}{2}\|P\theta_n - \theta^g\|_2^2\}, \tag{6.60}$$

the ADMM iterations that have to be performed to solve the addressed problem are

$$\hat{\theta}_n^{(k+1)}(T) = \underset{\theta_n}{\operatorname{argmin}}\ \mathcal{L}(\theta_n, \hat{\theta}^{g,(k)}, z_n^{(k)}, \delta_n^{(k)}), \tag{6.61}$$

$$z_n^{(k+1)} = \underset{z_n}{\operatorname{argmin}}\ \mathcal{L}(\hat{\theta}_{n,(k+1)}(T), \hat{\theta}^{g,(k)}, z_n, \delta_n^{(k)}), \tag{6.62}$$

$$\hat{\theta}^{g,(k+1)} = \underset{\theta^g}{\operatorname{argmin}}\ \mathcal{L}(\{\hat{\theta}_n^{(k+1)}\}_{n=1}^N, \theta^g, \{z_n^{(k+1)}, \delta_n^{(k)}\}_{n=1}^N), \tag{6.63}$$

$$\delta_{n,1}^{(k+1)} = \delta_{n,1}^{(k)} + \rho_1(\hat{\theta}_n^{(k+1)}(T) - z^{(k+1)}), \tag{6.64}$$

$$\delta_{n,2}^{(k+1)} = \delta_{n,2}^{(k)} + \rho_2(P\hat{\theta}_n^{(k+1)}(T) - \hat{\theta}^{g,(k+1)}). \tag{6.65}$$

Note that two sets of Lagrangian multipliers, $\{\delta_{n,1}\}_{n=1}^N$ and $\{\delta_{n,2}\}_{n=1}^N$, have been introduced, where $\delta_{n,1} \in \mathbb{R}^{n_g}$ is associated with the partial consensus constraint and $\delta_{n,2} \in \mathbb{R}^{n_\theta}$ is related to the constraints on the values of the local parameters.

Solving equations (6.62) and (6.63), the resulting updates for the auxiliary variables and the global estimate are

$$z_n^{(k+1)} = \mathcal{P}_{\mathcal{C}_n}\left(\hat{\theta}_n^{(k+1)}(T) + \frac{1}{\rho_1}\delta_{n,1}^{(k)}\right), \quad n = 1, \ldots, N, \tag{6.66}$$

$$\hat{\theta}^{g,(k+1)} = \frac{1}{N}\sum_{n=1}^{N}\left(P\hat{\theta}_n^{(k+1)}(T) + \frac{1}{\rho_2}\delta_{n,2}^{(k)}\right). \tag{6.67}$$

Observe that z-update is performed projecting a combination of the updated local estimate and $\delta_{n,1}^{(k)}$ onto the set $\mathcal{C}_n$, while $\hat{\theta}^{g,(k+1)}$ is computed as in Section 6.4 (equation 6.44), with δ_n replaced by $\delta_{n,2}$.

Consider the close form solution of (6.61), which is given by

$$\hat{\theta}_n^{(k+1)}(T) = \phi_n(T)\left\{\mathcal{Y}_n(T) - \delta_{n,1}^{(k)} - P'\delta_{n,2}^{(k)} + \rho_1 z_n^{(k)} + \rho_2 P'\hat{\theta}^{g,(k)}\right\}, \tag{6.68}$$

with

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau), \tag{6.69}$$

$$\mathcal{X}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau)(X_n(\tau))', \tag{6.70}$$

$$\phi_n(t) = \left(\left[\sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) X_n(\tau)'\right] + \rho_1 I_{n_\theta} + \rho_2 P'P\right)^{-1}. \tag{6.71}$$

for $t = 1, \ldots, T$.

To obtain recursive formulas to compute $\hat{\theta}_n^{(k+1)}$, we start proving that $\phi_n(T)$ can be computed as a function of $\phi_n(T-1)$. In particular, note that

$$\begin{aligned}\phi_n(T)^{-1} &= \mathcal{X}_n(T) + \rho_1 I_{n_\theta} + \rho_2 P'P = \\ &= \lambda_n \phi_n(T-1)^{-1} + X_n(T)X_n(T)' + (1-\lambda_n)\rho_1 + (1-\lambda_n)\rho_2 P'P.\end{aligned}$$

Defining the extended regressor as

$$\tilde{X}_n(T) = \begin{bmatrix} X_n(T) & \sqrt{(1-\lambda_n)\rho_1} I_{n_\theta} & \sqrt{(1-\lambda_n)\rho_2} P' \end{bmatrix}, \tag{6.72}$$

with $\tilde{X}_n(T) \in \mathbb{R}^{n_\theta \times (n_{\tilde{X}})}$ and $n_{\tilde{X}} = n_y + n_\theta + n_g$, and applying the matrix inversion lemma [114], it can be easily proven that $\phi_n(T)$ can then be computed as:

$$\mathcal{R}_n(T) = \lambda_n I_{(n_y+n_\theta+n_g)} + \tilde{X}_n(T)'\phi_n(T)\tilde{X}_n(T), \tag{6.73}$$

$$K_n(T) = \phi_n(T-1)\tilde{X}_n(T)(\mathcal{R}_n(T))^{-1}, \tag{6.74}$$

$$\phi_n(T) = \lambda_n^{-1}(I_{n_\theta} - K_n(T)\tilde{X}_n(T)')\phi_n(T-1). \tag{6.75}$$

The same observations relative to the update of ϕ_n made in Section 6.3-6.4 holds also in the considered case.

Method relying on N2C2N transmissions

Aiming at finding recursive formulas to update the estimates of the local parameters, we introduce the nth local estimate obtained at $T-1$, *i.e.*,

$$\hat{\theta}_n(T-1) = \phi_n(T-1)\{\mathcal{Y}_n(T-1) - \delta_{n,1}(T-1) - P'\delta_{n,2}(T-1) + \\ +\rho_1 z_n(T-1) + \rho_2 P'\hat{\theta}^g(T-1)\Big\} \tag{6.76}$$

with $\delta_{n,1}(T-1)$, $\delta_{n,2}(T-1)$, $z_n(T-1)$ and $\hat{\theta}^g(T-1)$ being the Lagrange multipliers, the n-th auxiliary variable and the global estimate obtained at $T-1$, respectively.
Adding and subtracting

$$\lambda_n\left[-\delta_{n,1}(T-1) - P'\delta_{n,2}(T-1) + \rho_1 z_n(T-1) + \rho_2 P'\hat{\theta}^g(T-1)\right]$$

to (6.68) and using the definition of $\phi_n(T-1)$ in (6.71), the local estimate $\hat{\theta}_n$ can be updated as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T), \tag{6.77}$$

with

$$\hat{\theta}_n^{RLS} = \phi_n(T)\lambda_n\left(\mathcal{Y}_n(T-1) - \delta_{n,1}(T-1) - P'\delta_{n,2}(T-1) + \\ +\rho_1 z_n(T-1) + \rho_2 P'\hat{\theta}^g(T-1)\right) + \phi_n(T)X_n(T)y_n(T), \tag{6.78}$$

and

$$\hat{\theta}_n^{ADMM,(k+1)}(T) = \phi_n(T)\left[\rho_1\Delta_{z,\lambda_n}^{(k+1)}(T) - \Delta_{1,\lambda_n}^{(k+1)}\right] + \\ + \phi_n(T)P'\left[\rho_2\Delta_{g,\lambda_n}^{(k+1)}(T) - \Delta_{2,\lambda_n}^{(k+1)}\right], \tag{6.79}$$
$$\Delta_{z,\lambda_n}^{(k+1)}(T) = z_n^{(k)} - \lambda_n z_n(T-1),$$
$$\Delta_{g,\lambda_n}^{(k+1)}(T) = \hat{\theta}^{g,(k)} - \lambda_n\hat{\theta}^g(T-1),$$
$$\Delta_{1,\lambda_n}^{(k+1)} = \delta_{n,1}^{(k)} - \lambda_n\delta_{n,1}(T-1),$$
$$\Delta_{2,\lambda_n}^{(k+1)} = \delta_{n,2}^{(k)} - \lambda_n\delta_{n,2}(T-1).$$

It can be noticed that (6.79) differs from (6.56) because of the introduction of the additional terms Δ_{z,λ_n} and Δ_{1,λ_n}.

Similarly to what is presented in Section 6.3, exploiting equation(6.75) and the equality $K_n(T) = \phi_n(T)\tilde{X}_n(T)$, the formula to update $\hat{\theta}_n^{RLS}$ can be further reduced to

$$\hat{\theta}_n^{RLS} = \hat{\theta}_n(T-1) + K_n(T)(\tilde{y}_n(T) - (\tilde{X}_n(T))'\hat{\theta}_n(T-1)), \tag{6.80}$$

with the extended measurement vector $\tilde{y}_n(T)$ is defined as

$$\tilde{y}_n(T) = \begin{bmatrix} y_n(T)' & O_{1\times n_\theta} & O_{1\times n_{n_g}} \end{bmatrix}'.$$

It is worth remarking that $\hat{\theta}_n^{RLS}$ can be updated (i) locally, (ii) recursively and (iii) once per step t.

The proposed method, summarized in Algorithm 16, requires the agents to transmit $\{\hat{\theta}_n^{RLS}, \phi_n\}$ to the cloud, while the cloud has to communicate $\hat{\theta}_n$ to each node once it has been computed.

Method relying on N2C transmissions

Using a N2C2N transmission scheme, the cloud has to communicate the updated estimates $\hat{\theta}_n$ to the local processors, before they can update again their local copies of the estimate $\hat{\theta}_n^{RLS}$. The nodes have thus to wait for the resources on the cloud to complete their computation.
To overcome this limitation, consider the close form solution of problem (6.60), *i.e.*,

$$\hat{\theta}_n^{(k+1)}(T) = \phi_n(T)\left\{\mathcal{Y}_n(T) - \delta_{n,1}^{(k)} - P'\delta_{n,2}^{(k)} + \rho_1 z_n^{(k)} + \rho_2 P'\hat{\theta}^{g,(k)}\right\},$$

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau), \quad t = 1, \dots, T$$

$$\phi_n(t) = \left(\left[\sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) X_n(\tau)'\right] + \rho_1 I_{n_\theta} + \rho_2 P'P\right)^{-1}, \quad t = 1, \dots, T.$$

Instead of introducing $\hat{\theta}_n(T-1)$, consider $\hat{\theta}_n^{RLS}(T-1)$, given by

$$\hat{\theta}_n^{RLS}(T-1) = \phi_n(T-1)\mathcal{Y}_n(T-1). \tag{6.81}$$

Observe that the local estimate can thus be updated as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T), \tag{6.82}$$

with

$$\hat{\theta}_n^{RLS}(T) = \left(I_{n_\theta} - K_n(T)(\tilde{X}_n(T))'\right)\hat{\theta}_n^{RLS}(T-1) + \phi_n(T)X_n(T)y_n(T),$$

$$\hat{\theta}_n^{ADMM,(k+1)}(T) = \phi_n(T)\left\{\rho_1 z_n^{(k)} + \rho_2 P'\hat{\theta}^{g,(k)} - \delta_{n,1}^{(k)} - P'\delta_{n,2}^{(k)}\right\}. \quad (6.83)$$

By using the equality $\phi_n(T)\tilde{X}_n(T) = K_n(T)$ and introducing the extended measurement vector

$$\tilde{y}_n(T) = \begin{bmatrix} y_n(T)' & 0_{1\times n_\theta} & 0_{1\times n_{n_g}} \end{bmatrix}',$$

the local estimate $\hat{\theta}_n^{RLS}$ can thus be updated as

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n^{RLS}(T-1) + K_n(T)\left(\tilde{y}_n(T) - (\tilde{X}_n(T))'\hat{\theta}_n^{RLS}(T-1)\right). \quad (6.84)$$

On the one hand, $\hat{\theta}^{g,(k+1)}$, $z_n^{(k+1)}$ and $\delta_{n,2}^{(k+1)}$ should be computed on the cloud, as $\hat{\theta}^g$ is updated in (6.67) using the estimates collected from all the N agents and equations (6.65)-(6.66) depend on the current global estimate. On the other hand, $\delta_{n,1}$ can be updated by the nth processor. However, under the hypothesis that the local computational power is limited and aiming at exploiting N2C communications, also $\delta_{n,1}^{(k+1)}$ is computed on the cloud.

The estimate $\hat{\theta}_n^{RLS}$ can be updated (i) at the level of the local processors, (ii) recursively and (iii) independently from quantities computed on the cloud. Moreover, equation (6.84) is independent from k. As a consequence, the clock scheduling the operations performed on the local processor is not influenced by the clock of the cloud.

Remark 6 *The method, outlined in Algorithm 16, requires the local processors to communicate $\hat{\theta}_n^{RLS}$ and ϕ_n at the cloud. On the other hand, to perform the local updates (6.74)-(6.75) and (6.84), the cloud does not have to transmit data to the local processors. The transmission scheme is outlined inFigure 75.* ■

6.5.1 Example 3

Suppose that $N = 100$ systems are described by the ARX model already introduced in Section 6.4 in equation (6.57), *i.e.*,

$$y_n(t) = \theta_1^g y_n(t-1) + \theta_{n,2} y_n(t-2) + \theta_2^g u_n(t-1) + e_n(t),$$

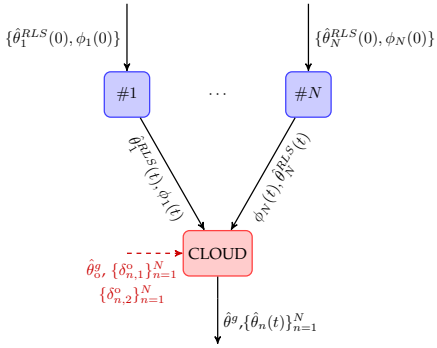


Figure 75: Cloud-aided estimation: scheme for the information exchange with Node-to-Cloud (N2C) transmissions.

where $\theta^g = \begin{bmatrix} 0.2 & 0.8 \end{bmatrix}'$ and $\{\theta_{n,2}\}_{n=1}^N$, for $n = 1, \ldots, N$, are sampled from the distribution $\mathcal{N}(0.4, 0.0025)$. The inputs u_n are generated as sequences of i.i.d. elements uniformly distributed in $[2,\ 3]$. The white noise sequence is $e_n \sim \mathcal{N}(0, R_n)$, with the covariance matrix being a natural number $R_n \in [1,\ 4]$. This choice for R_n yield to SNR_n in the interval $[6.4,\ 16.1]$ dB, $n = 1, \ldots, N$.

Our goal is to estimate both the global and the local parameters using the information collected over an estimation horizon of length $T = 5000$. Observe that the quality of $\hat{\theta}^g$ is evaluated through the RMSE, as defined in (6.39).

Assume that the a priori information constraints parameter estimates to the following ranges:

$$\begin{gathered} \ell_{n,1} \leq \hat{\theta}_{n,1} \leq up_{n,1} \quad \ell_{n,2} \leq \hat{\theta}_{n,2} \leq up_{n,2} \\ \ell_{n,3} \leq \hat{\theta}_{n,3} \leq up_{n,3}. \end{gathered} \tag{6.85}$$

With $\lambda_n = 1$, $\phi_n(0) = 0.1 I_{n_\theta}$, $\hat{\theta}_n(0)$ sampled from $\mathcal{N}(\theta_n, 4I_{n_\theta})$, $n = 1, \ldots, N$, and $\hat{\theta}^g$ sampled from the distribution $\mathcal{N}(\hat{\theta}^g, I_{n_g})$, while $\rho_1, \rho_2 \in \mathbb{R}^+$ are left to be tuned.

Method relying on N2C2N communications

To assess how the choice of ρ_1 and ρ_2 affects the satisfaction of the conditions specified in (6.85), we consider the average number of steps at

which (6.85) are violated, denoted as $\{\bar{N}_i^b\}_{i=1}^3$. These indicators are com-

Algorithm 16 ADMM-RLS algorithm for constrained consensus

Input: Data flow $\{X_n(1), y_n(1)\}, \{X_n(2), y_n(2)\}, \ldots$, initial matrices $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n(0)$, initial dual variables $\delta_{n,1}^{\mathrm{o}}$ and $\delta_{n,2}^{\mathrm{o}}$, initial auxiliary variables $\hat{z}_{n,\mathrm{o}}$, forgetting factors λ_n, $n = 1, \ldots, N$, initial global estimate $\hat{\theta}_{\mathrm{o}}^g$, parameters $\rho_1, \rho_2 \in \mathbb{R}^+$.

1. **for** $t = 1, 2, \ldots$ **do**

   Local

   1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $\tilde{X}_n(t)$ with (6.72);

   1.1.2. **compute** $K_n(t)$ and $\phi_n(t)$ with (6.74) - (6.75);

   1.1.3. **compute** $\hat{\theta}_n^{RLS}(t)$ with (6.80) (N2C2N) or (6.84) (N2C);

   1.2. **end for**;

   Global

   1.1. **do**

   1.1.1. **compute** $\hat{\theta}_n^{ADMM,(k+1)}(t)$ with (6.79) (N2C2N) or (6.83) (N2C), $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}_n^{(k+1)}(t)$ with (6.77), $n = 1, \ldots, N$;

   1.1.3. **compute** $z_n^{(k+1)}(t)$ with (6.66), $n = 1, \ldots, N$;

   1.1.4. **compute** $\hat{\theta}^{g,(k+1)}$ with (6.67);

   1.1.5. **compute** $\delta_{n,1}^{(k+1)}$ with (6.64), $n = 1, \ldots, N$;

   1.1.6. **compute** $\delta_{n,2}^{(k+1)}$ with (6.65), $n = 1, \ldots, N$;

   1.2. **until** a stopping criteria is satisfied (*e.g.*, maximum number of iterations attained);

2. **end**.

Output: Estimated global parameters $\hat{\theta}^g(t)$, estimated local parameters $\hat{\theta}_n(t)$, $n = 1, \ldots, N$.

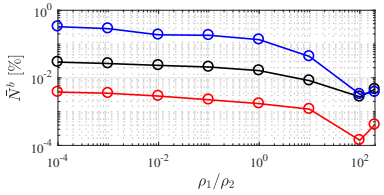


Figure 76: Example 3. $\bar{N}^b$ vs ρ_1/ρ_2. Average violation on the first component of the local parameter vector $\bar{N}_1^b$ (black), second component $\bar{N}_2^b$ (red), third component $\bar{N}_3^b$ (blue).

puted assuming that negligible[4] violations of the constraints are allowed. Consider the set of constraints

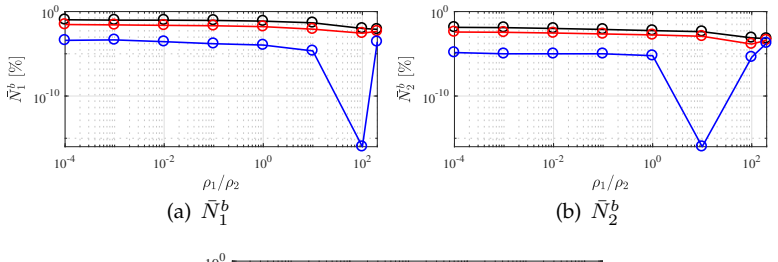
$$\mathcal{S}_2 = \{\ell_n = \left[\,0.19\ \theta_{n,2}{-}0.1\ 0.79\,\right], up_n = \left[\,0.21\ \theta_{n,2}{+}0.1\ 0.81\,\right]\}.$$

Figure 76 shows the average of $\{\bar{N}_i^b\}_{i=1}^3$, obtained fixing $\rho_2 = 0.1$ and choosing $\rho_1 = \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}.10^{-1}, 1, 10, 20\}$. As expected, if ρ_1 dominates over ρ_2 the number of violations tends to decrease, as the constraints on the value of the parameters are weighted more than the consensus constraint. However, if $\rho_1/\rho_2 > 100$, $\{\bar{N}_i^b\}_{i=1}^3$ tend to slightly increase. It is thus important to trade-off between the weights attributed to the conditions in (6.85) and the consensus constraint. We also evaluate how the stiffness of the constraints affects the choice of the parameters. In particular, we consider three different sets of constraints

$$\mathcal{S}_1 = \{\ell_n = \left[\,0.195\ \theta_{n,2}{-}0.05\ 0.795\,\right], up_n = \left[\,0.205\ \theta_{n,2}{+}0.05\ 0.805\,\right]\},$$
$$\mathcal{S}_2 = \{\ell_n = \left[\,0.19\ \theta_{n,2}{-}0.1\ 0.79\,\right], up_n = \left[\,0.21\ \theta_{n,2}{+}0.1\ 0.81\,\right]\},$$
$$\mathcal{S}_3 = \{\ell_n = \left[\,0.15\ \theta_{n,2}{-}0.5\ 0.75\,\right], up_n = \left[\,0.25\ \theta_{n,2}{+}0.5\ 0.85\,\right]\}.$$

The average number of violations $\{\bar{N}_i^b\}_{i=1}^3$ computed imposing the different constraints on the value of the local parameters are reported in Figure 77. It can be noticed that the higher the ratio ρ_1/ρ_2 is, the smaller $\{\bar{N}_i^b\}_{i=1}^3$ are. On the other hand, the constraint tends to be violated more if $\rho_1/\rho_2 > 100$, thus leading to higher values of $\{\bar{N}_i^b\}_{i=1}^3$.

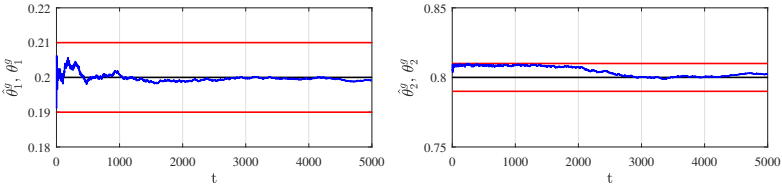
[4]The constraint $\ell_{n,i} \leq \hat{\theta}_{n,i} \leq up_{n,i}$ is assumed to be violated if $\hat{\theta}_{n,i} \notin \mathcal{B}_{n,i} = [\ell_{n,i} - 10^{-4},\ up_{n,i} + 10^{-4}]$, $i = 1, 2, 3$.



(a) $\bar{N}_1^b$ (b) $\bar{N}_2^b$

(c) $\bar{N}_3^b$

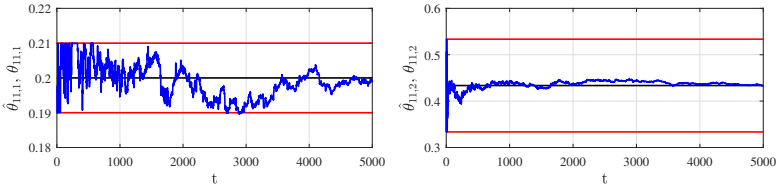
Figure 77: Example 3. $\bar{N}_i^b$ %, $i = 1, 2, 3$, vs ρ_1/ρ_2. First set of constraints $\mathcal{S}_1$ (black), second set of constraints $\mathcal{S}_2$ (red), third set of constraints $\mathcal{S}_3$ (blue).



(a) θ_1^g vs $\hat{\theta}_1^g$ (b) θ_2^g vs $\hat{\theta}_2^g$

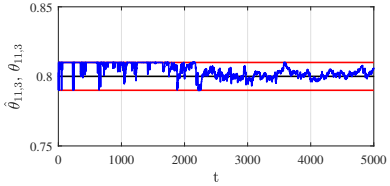
Figure 78: Example 3. $\hat{\theta}^g$ vs θ^g. True (black), estimates obtained with ADMM-RLS (blue), bounds (red).

Consider the set of constraints $\mathcal{S}_2$, Figure 78 shows the global estimates obtained with $\rho_1 = 10$ and $\rho_2 = 0.1$. Note that the estimates satisfy (6.85), showing that the constraints on the global estimate are automatically enforced imposing $\theta_n \in \mathcal{C}_n$.
The RMSEs obtained for the global estimates are equal to $\text{RMSE}_1^g = 0.001$ and $\text{RMSE}_2^g = 0.006$ for the first and the second component of θ^g, respectively. Their relatively small values can be related to the introduction of the constraints on the value of the local parameters, that allow us to limit



(a) $\theta_{11,1}$ vs $\hat{\theta}_{11,1}$

(b) $\theta_{11,2}$ vs $\hat{\theta}_{11,2}$



(c) $\theta_{11,3}$ vs $\hat{\theta}_{11,3}$

Figure 79: Example 3. θ_n, $n = 11$. True (black), estimate obtained with ADMM-RLS (blue), bounds (red).

the resulting estimation error.

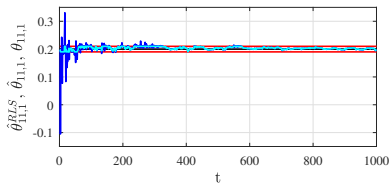
Figure 79 shows the estimate $\hat{\theta}_n$ for $n = 11$ ($SNR_{11} = 10.6$ dB)., while $\hat{\theta}_n$ and $\hat{\theta}_n^{RLS}$, with $n = 11$, are compared Figure 80. As it can be noticed, while $\hat{\theta}_{11}$ satisfies the constraints, the effect produced by the use of $\hat{\theta}_{11}$ in the update $\hat{\theta}_{11}^{RLS}$ (see (6.80)) is not strong enough to enfoce also the local copies of the estimates to satisfy the contraints for the whole estimation horizon.

Method relying on N2C transmissions

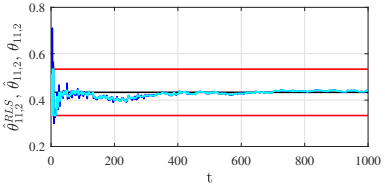
Consider the set of constraints $\mathcal{S}_2$, *i.e.*,

$$\begin{aligned} \ell_n &= \begin{bmatrix} 0.19 & \theta_{n,2} - 0.1 & 0.79 \end{bmatrix}, \\ up_n &= \begin{bmatrix} 0.21 & \theta_{n,2} + 0.1 & 0.81 \end{bmatrix}. \end{aligned} \tag{6.86}$$

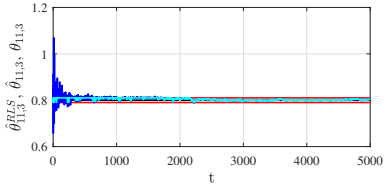
With $\rho_2 = 0.1$, Figures 81 and 82 report $\|\text{RMSE}^g\|_2$ and $\{\bar{N}_i^b\}_{i=1}^3$ for different ratios ρ_1/ρ_2, respectively. Observe that the more ρ_1 dominates over ρ_2, the more $\{\bar{N}_i^b\}_{i=1}^3$ are reduced, but the average number of violations tends to slightly increase when $\rho_1/\rho_2 > 100$. Also $\|\text{RMSE}^g\|_2$ increases significantly when $\rho_1/\rho_2 \geq 100$. Both these results are compliant with



(a) $\theta_{11,1}$ vs $\hat{\theta}_{11,1}$ and $\hat{\theta}^{RLS}_{11,1}$



(b) $\theta_{11,2}$ vs $\hat{\theta}_{11,2}$ and $\hat{\theta}^{RLS}_{11,2}$



(c) $\theta_{11,3}$ vs $\hat{\theta}_{11,3}$ and $\hat{\theta}^{RLS}_{11,3}$

Figure 80: Example 3. θ_n, $n = 11$. True (black), $\hat{\theta}^{RLS}_{11}$ (blue), local estimate computed on the cloud $\hat{\theta}_{11}$ (blue), bounds (red). As $\hat{\theta}^{RLS}_{11,3}$ tends to satisfy the constraints after the first 1000 samples, only the estimate of $\hat{\theta}^{RLS}_{11,3}$ is plotted for the entire horizon.

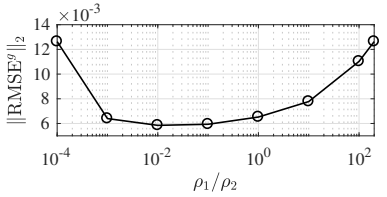


Figure 81: Example 3. $\|\text{RMSE}^g\|_2$ vs ρ_1/ρ_2.

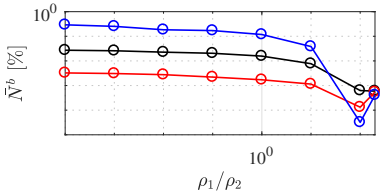
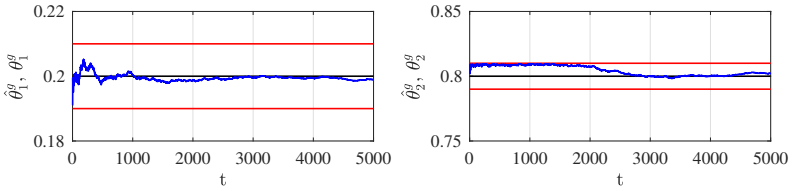


Figure 82: Example 3. $\bar{N}^b$ [%] vs ρ_1/ρ_2. Average number of violation for the first component of the local parameter vector $\bar{N}^b_1$ (black), second component $\bar{N}^b_2$ (red), third component $\bar{N}^b_3$ (blue).

the meaning of the parameters ρ_1 and ρ_2.and they imply the necessity to find for a good trade-off between ρ_1 and ρ_2.
Choosing $\rho_1 = 1$ and $\rho_2 = 0.1$, the obtained estimates of θ^g are reported in Figure 83. Observe that $\hat{\theta}^g$ tends to converge to the actual value of the global parameter and that, enforcing (6.85)-(6.86), $\hat{\theta}^g$ satisfy the condition

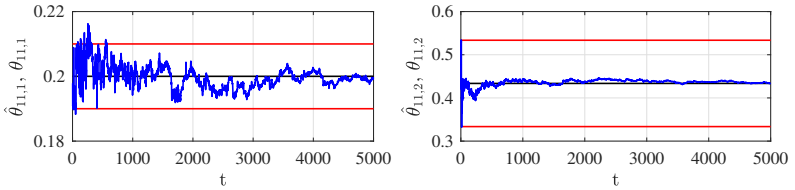
$$0.19 \leq \hat{\theta}_1^g \leq 0.21 \quad 0.79 \leq \hat{\theta}_2^g \leq 0.81,$$

even if the constraints are not directly imposed on the values of the global parameters.
On the one hand, the global estimates reported in Figure 83 are similar to the ones obtained with the method based on N2C2N transmissions plotted in Figure 78). On the other hand, $\hat{\theta}_{11}$ obtained with N2C communications tends to violate the conditions in (6.85) at the beginning of the estimation horizon, while the corresponding estimate computed with the method relying on N2C2N transmission always satisfy the constraints (see Figure 79). Observe that, choosing $\rho_1 = 10$ and $\rho_2 = 0.1$ as in the case of N2C2N communications, $\hat{\theta}_{11}$ tends to verify the constraint over the considered estimation horizon. However, this choice yields to less accurate global estimates as shown in Figure 81. This result can be related to the differences in the updates of $\hat{\theta}_n^{RLS}$, updated as in (6.80) and (6.84) considering N2C2N and N2C transmissions, respectively.
In Figure 85, the local estimates $\hat{\theta}_{11}$ are further compared with $\hat{\theta}_{11}^{RLS}$ and the estimate obtained applying standard RLS [72], $\hat{\theta}_{11}^{RLS\star}$. As expected, the $\hat{\theta}_{11}^{RLS}$ and $\hat{\theta}_{11}^{RLS\star}$ are slightly different and they do not satisfy the conditions in (6.85), while $\hat{\theta}_{11}$ generally satisfies the constraints.

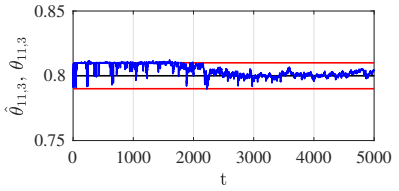


(a) θ_1^g vs $\hat{\theta}_1^g$ (b) θ_2^g vs $\hat{\theta}_2^g$

Figure 83: Example 3. $\hat{\theta}^g$ vs θ^g True (black), estimates obtained with ADMM-RLS (blue), bounds (red).

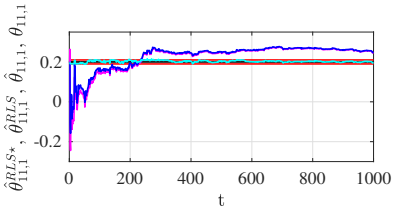


(a) $\theta_{11,1}$ vs $\hat{\theta}_{11,1}$ (b) $\theta_{11,2}$ vs $\hat{\theta}_{11,2}$

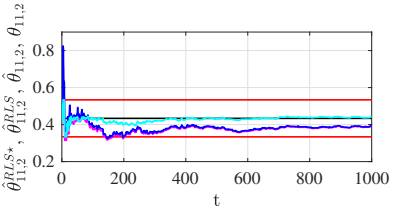


(c) $\theta_{11,3}$ vs $\hat{\theta}_{11,3}$

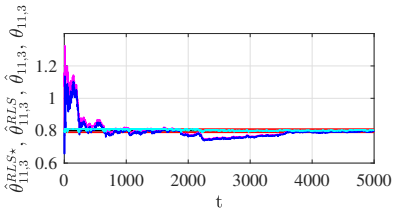
Figure 84: Example 3. θ_{11}. True (black), estimates obtained with ADMM-RLS (blue), bounds (red).



(a) $\theta_{11,1}$ vs $\hat{\theta}^{RLS\star}_{11,1}$,$\hat{\theta}^{RLS}_{11,1}$, $\hat{\theta}_{11,1}$



(b) $\theta_{11,2}$ vs $\hat{\theta}^{RLS\star}_{11,2}$, $\hat{\theta}^{RLS}_{11,2}$, $\hat{\theta}_{11,2}$



(c) $\theta_{11,3}$ vs $\hat{\theta}^{RLS\star}_{11,3}$, $\hat{\theta}^{RLS}_{11,3}$, $\hat{\theta}_{11,3}$

Figure 85: Example 3. θ_{11}. True (black), $\hat{\theta}^{RLS\star}_{11}$ obtained with standard RLS [72] (magenta), $\hat{\theta}^{RLS}_{11}$ (blue), local estimates on the cloud $\hat{\theta}_{11}$ (cyan), bounds (red).

Chapter 7

Cloud-aided collaborative estimation

The Nonlinear and Multi-class cases

In this chapter, schemes based on the Alternating Direction Method of Multipliers (ADMM) similar to the ones introduced in chapter 6 are proposed to tackle consensus-based estimation problem with (i) nonlinear consensus constraints and (ii) multiple classes.

After a brief review of the general collaborative estimation problem, Section 7.1 briefly reviews the ADMM scheme used to tackle the nonlinear collaborative estimation problem. Then, a first strategy to handle nonlinear consensus is presented in Section 7.2. In Section 7.3, a method for constrained cloud-aided estimation with nonlinear consensus is introduced. Finally, a strategy to handle the multi-class estimation problem is described in Section 7.4.

Throughout the Chapter $\mathcal{P}_{\mathcal{A}}$ denotes the Euclidean projection onto the set $\mathcal{A}$.

Assume that (i) the measurements acquired by N agents are available and that (ii) the behavior of the N data-generating systems is described

by the same model with parameters $\theta_n \in \mathbb{R}n_\theta$, $n = 1, \dots, N$. As already pointed out in chapter 6, the agents are associated with the same model and thus it can be assumed that (iii) there exists a set of parameters $\theta^g \in \mathbb{R}^{n_g}$, with $n_g \leq n_\theta$, common to all the agents. We aim at (i) computing *local* estimates of the unknowns $\{\theta_n\}_{n=1}^N$, and (ii) identifying the *global* parameters θ^g on the cloud, using the data collected from all the available sources.
The addressed estimation problem can be cast into the following separable optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} f_n(\theta_n) \\ \text{s.t.} \quad & F(\theta_n) = \theta^g, \\ & \theta_n \in \mathcal{C}_n, \;\; n = 1, \dots, N, \end{aligned} \tag{7.1}$$

where $f_n : \mathbb{R}^{n_\theta} \to \mathbb{R} + \{\infty\}$ is a closed, proper, convex function, $F : \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_g}$ is a nonlinear operator and $\mathcal{C}_n \subset \mathbb{R}^{n_\theta}$ is a convex set. In particular, we focus on the problem of estimating the unknowns in the least-squares sense, *i.e.*, the local cost functions are given by

$$f_n(\theta_n) = \frac{1}{2} \sum_{t=1}^{T} \lambda_n^{T-t} \left\| y_n(t) - (X_n(t))' \theta_n \right\|_2^2, \tag{7.2}$$

for $n = 1, \dots, N$.

7.1 ADMM with nonlinear operator constraints

We briefly review the algorithm proposed in [18], that allows to solve optimization problems with nonlinear constraints in an ADMM-like fashion.
Consider the optimization problem given by

$$\begin{aligned} \text{minimize} \quad & f(\theta) + g(z) \\ \text{s.t.} \quad & H(\theta, z) = c, \end{aligned} \tag{7.3}$$

where f and g are proper, convex and lower semi-continuous functions, c is a given function and H is a nonlinear operator [18]. Through the

introduction of the nonlinear operator H, the constraints are nonlinear with respect to θ and z.
As in standard ADMM [23], consider the augmented Lagrangian associated with the considered problem, given by

$$\mathcal{L}(\theta, z, \rho) = f(\theta) + g(z) + \delta'(H(\theta, z) - c) + \frac{\rho}{2}\|H(\theta, z) - c\|_2^2, \tag{7.4}$$

which depends on the nonlinear operator H. Using the standard ADMM scheme without further approximations, the iterations to be performed require the solution of two nonlinear optimization problems when θ and z are updated.
As presented in [18], problem (7.3) can be alternatively solved through the simultaneous linearization of the nonlinear operator H and the solution of the problem with an inexact ADMM scheme. Not to deal with nonlinear sub-problems, $H(\theta, z)$ is replaced with its truncated Taylor expansion around the current estimates of θ and z, respectively. Defining the Jacobians computed at each ADMM iteration as

$$F^{(k)} = \partial_\theta H(\theta, z^{(k)})\big|_{\theta=\theta^{(k)}}, \tag{7.5}$$
$$G^{(k)} = \partial_z H(\theta^{(k+1)}, z)\big|_{z=z^{(k)}}, \tag{7.6}$$

$H(\theta, z)$ is approximated as

$$H(\theta, z^{(k)}) \approx H(\theta^{(k)}, z^{(k)}) + F^{(k)}(\theta - \theta^{(k)}), \tag{7.7}$$
$$H(\theta^{(k+1)}, z) \approx H(\theta^{(k+1)}, z^{(k)}) + G^{(k)}(z - z^{(k)}), \tag{7.8}$$

when θ and z are updated, respectively.
The ADMM scheme handling nonlinear constraints proposed in [18] is summarized by the following steps:

$$\theta^{(k+1)} = \underset{\theta}{\operatorname{argmin}} \left\{ f(\theta) + (\delta^{(k)})' F^{(k)}\theta + \frac{\rho}{2}\|F^{(k)}\theta - c_1^{(k)}\|_2^2, \right\} \tag{7.9}$$
$$z^{(k+1)} = \underset{z}{\operatorname{argmin}} \left\{ g(z) + (\delta^{(k)})' G^{(k)} z + \frac{\rho}{2}\|G^{(k)}\theta - c_2^{(k)}\|_2^2 \right\}, \tag{7.10}$$
$$\delta^{(k+1)} = \delta^{(k)} + \rho \left(H(\theta^{(k+1)}, z^{(k+1)} - c) \right), \tag{7.11}$$

with

$$c_1^{(k)} = c + F^{(k)}\theta^{(k)} - H(\theta^{(k)}, z^{(k)})$$
$$c_2^{(k)} = c + G^{(k)}z^{(k)} - H(\theta^{(k+1)}, z^{(k)}).$$

The approximation introduced to handle nonlinear constraints is the same as the ones used in nonlinear Recursive Least-Squares (NL-RLS) [1] and in the Extended Kalman Filter (EKF) [19].

7.2 Case study 1. Nonlinear consensus.

Assume that no constraint has to be imposed on the values of the unknowns $\{\theta_n\}_{n=1}^N$, so that the problem to be solved is given by

$$\begin{aligned} &\text{minimze} && \sum_{n=1}^{N} f_n(\theta_n) \\ &\text{s.t.} && F(\theta_n) = \theta^g, \;\; n = 1, \dots, N, \end{aligned} \tag{7.12}$$

with the consensus constraint expressed as a nonlinear function of the local parameters θ_n, $n = 1, \dots, N$, with $F : \mathbb{R}^{n_\theta} \to \mathbb{R}^{n_g}$.
With the ultimate goal of obtaining recursive formulas to estimate both $\theta_n, n = 1, \dots, N$, and θ^g, we solve address problem (7.12) with the method presented in [18] and briefly reviewed in Section 7.1.

Consider the augmented Lagrangian associated with the problem (7.12)

$$\begin{aligned} &\mathcal{L} = \sum_{n=1}^{N} \mathcal{L}_n(\theta_n, \theta^g, \delta), \\ &\mathcal{L}_n(\theta_n, \theta^g, \delta) = f_n(\theta) + \delta_n'(F(\theta_n) - \theta^g) + \frac{\rho}{2}\left\|F(\theta_n) - \theta^g\right\|_2^2, \end{aligned} \tag{7.13}$$

with f_n defined as in (7.2). The ADMM iterations to be performed are

$$\hat{\theta}_n^{(k+1)}(T) = \operatorname{argmin} \theta_n \mathcal{L}_n(\theta, \hat{\theta}^{g,(k)}, \delta_n^{(k)}), \tag{7.14}$$

$$\hat{\theta}^{g,(k+1)} = \frac{1}{N}\sum_{n=1}^{N}\left(F(\hat{\theta}_n^{(k+1)}(T)) + \frac{1}{\rho}\delta_n^{(k)}\right), \tag{7.15}$$

$$\delta_n^{(k+1)} = \delta_n^{(k)} + \rho(F(\hat{\theta}_n^{(k+1)}(T)) - \hat{\theta}^{g,(k+1)}), \tag{7.16}$$

with the local Lagrangian defined as On the one hand, the updates of the global parameters and the Lagrange multipliers in (7.15) and (7.16) require only the computation of $F(\hat{\theta}_n^{(k+1)}(T))$. On the other hand, (7.14) depends on the nonlinear function $F(\theta_n)$. Focusing on the update of the local estimates, not to require each local processor to solve a nonlinear optimization problem, at step t, $F(\theta_n)$ is approximated through its truncated Taylor expansion around $\hat{\theta}_n^{(k)}(t)$, *i.e.*,

$$F(\theta_n) \approx F(\hat{\theta}_n^{(k)}(t)) + A_n^{(k)}(t)\left(\theta_n - \hat{\theta}_n^{(k)(t)}\right) \tag{7.17}$$

with

$$A_n^{(k)}(t) = \left.\frac{\partial F(\theta_n)}{\partial \theta_n}\right|_{\theta_n=\hat{\theta}_n^{(k)}(t)}. \tag{7.18}$$

Once $A_n^{(k)}(t)$ is computed, the local parameters can be updated as:

$$\hat{\theta}_n^{(k+1)}(T) = \phi_n^{(k+1)}(T)\left\{\mathcal{Y}_n(T) - (A_n^{(k)}(t))'\delta_n^{(k)} - \rho(A_n^{(k)}(t))'c_n^{(k)(t)}\right\} \tag{7.19}$$

with

$$\begin{aligned}
\mathcal{Y}_n(t) &= \left[\sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau)\right] \\
c_n^{(k)}(t) &= \left[F(\hat{\theta}_n^{(k)}(t)) - A_n^{(k)(t)}\hat{\theta}_n^{(k)}(t) - \hat{\theta}^{g,(k)}\right] \\
\mathcal{X}_n(t) &= \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau)(X_n(\tau))', \\
\phi_n^{(k+1)}(t) &= \left(\mathcal{X}_n(t) + \rho(A_n^{(k)}(t))'A_n^{(k}(t)\right)^{-1}.
\end{aligned}$$

Note that $\phi_n^{(k+1)}(T)$ depends on $A_n^{(k)}(t)$ and, consequently, on the current ADMM iteration k. The matrices $\{\theta_n\}_{n=1}^N$ cannot thus be updated once per time step, but they must be computed at each ADMM iteration, requiring additional synchronization between the cloud and the local processors.

As we aim at obtaining recursive formulas to update $\hat{\theta}_n$, we introduce

the nth local estimate obtained at $T-1$, given by

$$\hat{\theta}_n(T-1) = \phi_n(T-1)\Big\{\mathcal{Y}_n(T-1) - (A_n(T-1))'\delta_n^{(k)} + \\ -\rho(A_n(T-1))'c_n(T-1)\}\,, \tag{7.20}$$

with $A_n(T-1) = A_n^{(\kappa-1)}(T-1)$, $c_n(T-1) = c_n^{(\kappa-1)}(T-1)$ and κ is the last ADMM iteration performed at step $T-1$.
The matrix $\phi_n^{(\kappa)}(T-1)$ is given by

$$\phi_n^{(\kappa)}(T-1) = \{\mathcal{X}_n(T-1) + \rho(A_n(T-1))'A_n(T-1)\}^{-1}\,. \tag{7.21}$$

Exploiting the definition of $\phi_n(T)$ in (7.21), it can be proven that matrix ϕ_n can be updated as

$$\mathcal{R}_n^{(k+1)}(T) = \lambda_n\mathcal{I} + (\tilde{X}_n^{(k)}(T))'\phi_n(T-1)\tilde{X}_n^{(k)}(T), \tag{7.22}$$

$$K_n^{(k+1)}(T) = \phi_n(T-1)\tilde{X}_n^{(k)}(T)\left(R_n^{(k+1)}(T)\right)^{-1}, \tag{7.23}$$

$$\phi_n^{(k+1)}(T) = \lambda_n^{-1}(I_{n_\theta} - K_n^{(k+1)}(T)(\tilde{X}_n^{(k)}(T))')\phi_n(T-1), \tag{7.24}$$

where we have introduced the extended regressor $\tilde{X}_n^{(k)}(T)$

$$\tilde{X}_n^{(k)}(T) = \begin{bmatrix} X_n(T) & \sqrt{\rho}(A_n^{(k+1)})' & \sqrt{\rho\lambda_n}(A_n(T-1))' \end{bmatrix} \in \mathbb{R}^{n_\theta\times(n_y+2n_g)}, \tag{7.25}$$

and matrix

$$\mathcal{I} = \begin{bmatrix} I_{n_y} & 0_{n_y\times n_g} & 0_{n_y\times n_g} \\ 0_{n_g\times n_y} & I_{n_g} & 0_{n_g} \\ 0_{n_g\times n_y} & 0_{n_g} & -I_{n_g} \end{bmatrix}.$$

Equations (7.22)-(7.24) depend on the ADMM iteration k, due to the dependence of $\phi_n^{(k+1)}(T)$ on $A_n^{(k)}(T)$.

Adding and subtracting

$$-\lambda_n\phi_n^{(k+1)}(T)(A_n(T-1))'\left[\delta_n(T-1) + \rho c_n(T-1)\right]$$

to (7.19), and exploiting the definition of $\hat{\theta}_n(T-1)$ (see (7.21)) and the recursive formula (7.24) derived to compute $\phi_n^{(k+1)}(T)$, $\hat{\theta}_n^{(k+1)}(T)$ can be computed as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS,(k+1)}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T). \tag{7.26}$$

In particular, $\hat{\theta}_n^{ADMM,(k+1)}(T)$ is obtained as

$$\begin{aligned}
\hat{\theta}_n^{ADMM,(k+1)}(T) &= \phi_n^{(k+1)}(T)\left\{\rho\Delta_{g,\lambda_n}^{(k+1)} + \Delta_{\lambda_n}^{(k+1)}\right\}, \qquad (7.27)\\
\Delta_{g,\lambda_n}^{(k+1)} &= \lambda_n(A_n(T-1))'c_n(T-1) - (A_n^{(k+1)})'c_n^{(k+1)}\\
\Delta_{\lambda_n}^{(k+1)} &= \lambda_n(A_n(T-1))'\delta_n(T-1) - (A_n^{(k+1)})'\delta_n^{(k)}.
\end{aligned}$$

The quantity $\Delta_{g,\lambda_n}^{(k+1)}$ depends on the difference between the current global estimate and the one obtained at the previous step as $c_n^{(k+1)} \propto \hat{\theta}^{g,(k)}$ and $c_n(T-1) \propto \hat{\theta}^g(T-1)$.
By using the extended measurement vector $\tilde{y}_n$

$$\tilde{y}_n(T) = \begin{bmatrix} y_n(T)' & 0_{1\times n_g} & 0_{1\times n_g}\end{bmatrix}'$$

and the extended regressor $\tilde{X}_n$ (see (7.25)), $\hat{\theta}_n^{RLS,(k+1)}$ is instead updated as

$$\hat{\theta}_n^{RLS,(k+1)}(T) = \hat{\theta}_n(T-1) + K_n^{(k)}(T)(\tilde{y}_n(T) - \tilde{X}_n^{(k)}(T)\hat{\theta}_n(T-1)), \quad (7.28)$$

that is obtained exploiting the equalities

$$\begin{aligned}
&\mathcal{I}\tilde{y}_n(T) = \tilde{y}_n(T),\\
&\phi_n^{(k+1)}(T)\tilde{X}_n^{(k)}(T) = K_n^{(k+1)}(T)\mathcal{I}.
\end{aligned}$$

This last equality can be proven using the matrix inversion lemma[114] and (7.23). Even though the formulas to update $\hat{\theta}_n^{RLS,(k+1)}$ are still recursive, $\hat{\theta}_n^{RLS,(k+1)}$ depends on the ADMM iteration. Consequently, both $\hat{\theta}_n^{RLS,(k+1)}$ and $\hat{\theta}_n^{ADMM,(k+1)}$ have to be updated at each ADMM run. As the equations (7.15), (7.16), (7.27) and (7.28) depend on the ADMM iteration, the operations performed by the local processors and the ones performed on the cloud have to be synchronized at each iteration k. However, this level of coordination between the nodes and the cloud might be challenging to achieve in practice.

7.3 Case study 2. Constrained nonlinear consensus

Instead of considering the problem addressed in Section 7.2, where no constraint is imposed on the values of the parameters θ_n, in this Section

---

Algorithm 17 ADMM-RLS for nonlinear consensus

Input: Data flow $\{X_n(1), y_n(1)\}, \{X_n(2), y_n(2)\}, \ldots$, initial matrices $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n(0)$, initial dual variables δ_n^{o}, forgetting factors λ_n, $n = 1, \ldots, N$, initial global estimate $\hat{\theta}_{\text{o}}^g$, parameter $\rho \in \mathbb{R}^+$.

---

1. **for** $t = 1, 2, \ldots$ **do**

   1.1. **do**

   Local

   1.1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $A_n^{(k)}$ as (7.18);

   1.1.2. **compute** $\tilde{X}_n^{(k)}(t)$ as in (7.25);

   1.1.3. **compute** $K_n^{(k)}(t)$ and $\phi_n^{(k)}(t)$ with (7.23) - (7.24);

   1.1.4. **compute** $\hat{\theta}_n^{RLS,(k+1)}(t)$ with (7.28);

   1.1.2. **end for**;

   Global

   1.1.1. **compute** $\hat{\theta}_n^{ADMM,(k+1)}(t)$ with (7.27), $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}_n^{(k+1)}(t)$ as in (7.26), for $n = 1, \ldots, N$;

   1.1.3. **compute** $\hat{\theta}^{g,(k+1)}(t)$ with (7.15);

   1.1.4. **compute** $\delta_n^{(k+1)}$ with (7.16), $n = 1, \ldots, N$;

2. **until** a stopping criteria is satisfied (*e.g.*, the maximum number of iterations is attained).

Output: Estimated global parameters $\hat{\theta}^g(t)$, estimated local parameters $\hat{\theta}_n(t)$, $n = 1, \ldots, N$.

---

we design a solution to the following constrained problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} \{f_n(\theta_n)\} \\ \text{s.t.} \quad & \theta_n \in \mathcal{C}_n, \;\; n = 1, \dots, N \\ & F(\theta_n) = \theta^g, \;\; n = 1, \dots, N. \end{aligned} \tag{7.29}$$

Through the introduction of the auxiliary variables $z_n \in \mathbb{R}^{n_\theta}$, for $n = 1, \dots, N$ and the indicator functions

$$g_n(z_n) = \begin{cases} 1 \text{ if } z_n \in \mathcal{C}_n \\ +\infty \text{ otherwise,} \end{cases}$$

the initial constrained problem can be reformulated as

$$\begin{aligned} \text{minimize} \quad & \sum_{n=1}^{N} \{f_n(\theta_n) + g_n(z_n)\} \\ \text{s.t.} \quad & \theta_n = z_n, \;\; n = 1, \dots, N \\ & F(z_n) = \theta^g, \;\; n = 1, \dots, N. \end{aligned} \tag{7.30}$$

The consensus constraint is not defined as a function of the unknown parameter θ_n, but as a function of the auxiliary variable z_n, with $z_n \in \mathbb{R}^{n_\theta}$, *i.e.*,

$$F(\theta_n) = \theta^g \rightarrow F(z_n) = \theta^g. \tag{7.31}$$

This "suboptimal" approximation, which is feasible thanks to the constraints on the value of the local parameters $z_n = \theta_n$, $n = 1, \dots, N$, allows us to update $\hat{\theta}_n^{RLS}$ independently from the ADMM iteration k. This is not achievable maintaining the original formulation in (7.29), as the considered setting would be similar to the one studied in Section 7.2.

Consider the augmented Lagrangian associated with problem (7.30), which is given by

$$\mathcal{L} = \sum_{n=1}^{N} \mathcal{L}_n(\theta_n, z_n, \theta^g, \delta_{n,1}, \delta_{n,2}),$$

$$\mathcal{L}_n = f_n(\theta_n) + g_n(z_n) + (\delta_{n,1})'\varepsilon_{n,1} + (\delta_{n,2})'\varepsilon_{n,2} + \frac{\rho_1}{2}\|\varepsilon_{n,1}\|_2^2 + \frac{\rho_2}{2}\|\varepsilon_{n,2}\|_2^2,$$

with $f_n(\theta_n)$ is defined as in (7.2) and

$$\begin{aligned}\varepsilon_{n,1} &= \theta_n - z_n,\\ \varepsilon_{n,2} &= F(z_n) - \theta^g.\end{aligned}$$

Due to the presence of two sets of constraints, two sets of dual variables, $\{\delta_{n,1} \in \mathbb{R}^{n_\theta}\}_{n=1}^N$ and $\{\delta_{n,2} \in \mathbb{R}^{n_g}\}_{n=1}^N$, are introduced accordingly. Based on the ADMM scheme presented in [18], the steps that have to be performed are:

$$\hat{\theta}_n^{(k+1)}(T) = \underset{\theta_n}{\operatorname{argmin}} \left\{ \mathcal{L}_n(\theta_n, z_n^{(k)}, \hat{\theta}^{g,(k)}, \delta_{n,1}^{(k)}, \delta_{n,2}^{(k)}) \right\}, \tag{7.32}$$

$$z_n^{(k+1)} = \mathcal{P}_{\mathcal{C}_n}\left(\left(\Gamma_n^{(k+1)}\right)^{-1} \gamma_n^{(k+1)} \right), \tag{7.33}$$

$$\hat{\theta}^{g,(k+1)} = \frac{1}{N}\sum_{n=1}^{N}\left[F(z_n^{(k+1)}) + \frac{1}{\rho_2}\delta_{n,2}^{(k)} \right], \tag{7.34}$$

$$\delta_{n,1}^{(k+1)} = \delta_{n,1}^{(k)} + \rho_1(\hat{\theta}_n^{(k+1)} - z_n^{(k+1)}), \tag{7.35}$$

$$\delta_{n,2}^{(k+1)} = \delta_{n,2}^{(k)} + \rho_2(F(z_n^{(k+1)}) - \hat{\theta}^{g,(k+1)}), \tag{7.36}$$

with

$$\begin{aligned}\Gamma_n^{(k+1)} &= \rho_1 I_{n_\theta} + \rho_2 (A_n^{(k)})' A_n^{(k)}\\ \gamma_n^{(k+1)} &= \delta_{n,1} + \rho_1 \hat{\theta}_n^{(k+1)} + \rho_2 (A_n^{(k)})' c_n^{(k)} - (A_n^{(k)})' \delta_{n,2}^{(k)}.\end{aligned}$$

The z-update (see (7.33)) is obtained replacing $F(z_n)$ with its truncated Taylor expansion

$$\begin{aligned}F(z_n) &\approx F(z_n^{(k)}) + A_n^{(k)}(z_n - z_n^{(k)}),\\ A_n^{(k)} &= \partial_{z_n} F(z_n)\big|_{z_n = z_n^{(k)}},\end{aligned}$$

and

$$c_n^{(k)} = A_n^{(k)} z_n^{(k)} + \hat{\theta}^{g,(k)} - F(z_n^{(k)}).$$

The global parameter is updated in (7.34) combining the sample means of $\{F(\theta_n)\}_{n=1}^N$, *i.e.*, the value of the local copy of the global parameter, and the average of the Lagrange multipliers $\{\delta_{n,1}\}_{n=1}^N$. Observe that, as

expected, equation (7.34) corresponds to (7.15), with z_n replacing $\hat{\theta}_n$ and $\delta_{n,2}$ substituting δ_n.
The chosen quadratic cost functions f_n in (7.2) allows one to find a closed for solution for the problem in (7.32). The local estimates can thus be updated as

$$\hat{\theta}_n^{(k+1)} = \phi_n(T)\left\{\mathcal{Y}_n(T) - \delta_{n,1}^{(k)} + \rho_1 \hat{z}_n^{(k)}\right\} \tag{7.37}$$

with

$$\phi_n(t) = \left(\left[\sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) X_n(\tau)'\right] + \rho_1 I_{n_\theta}\right)^{-1} \tag{7.38}$$

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau). \tag{7.39}$$

Thanks to the "suboptimal" choice made in defining the consensus constraints (see (7.31)), (7.37) is now independent from F.

Based on the definition of ϕ_n given in (7.38), it can be easily shown that matrix $\phi_n(T)$ can be recursively updated as

$$\mathcal{R}_n(T) = \lambda_n I_{n_{\tilde{X}}} + \tilde{X}_n(T)' \phi_n(T) \tilde{X}_n(T), \tag{7.40}$$

$$K_n(T) = \phi_n(T-1)\tilde{X}_n(T)(\mathcal{R}_n(T))^{-1}, \tag{7.41}$$

$$\phi_n(T) = \lambda_n^{-1}\left(I_{n_\theta} - K_n(T)(\tilde{X}_n(T))'\right)\phi_n(T-1), \tag{7.42}$$

with the extended regressor defined as

$$\tilde{X}_n(t) = \begin{bmatrix} X_n(t) & \sqrt{(1-\lambda_n)\rho_1} I_{n_\theta} \end{bmatrix} \in \mathbb{R}^{n_\theta \times (n_y + n_\theta)}. \tag{7.43}$$

Solution based on N2C2N transmissions

As our goal is to retrieve recursive formulas to update $\hat{\theta}_n$, we introduce the n-th local estimate $\hat{\theta}_n(T-1)$ at time $T-1$, *i.e.*,

$$\hat{\theta}_n(T-1) = \phi_n(T-1)\left\{\mathcal{Y}_n(T-1) - \delta_{n,1}(T-1) + \rho_1 z_n(T-1)\right\}, \tag{7.44}$$

with $z_n(T-1)$ and $\delta_{n,1}(T-1)$ indicating the auxiliary variable and the Lagrange multiplier obtained at step T-1.

Adding and subtracting $\lambda_n \left(\rho_1 \hat{z}_n(T-1) - \delta_n(T-1)\right)$ to (7.37), the estimate of the local parameters $\hat{\theta}_n^{(k+1)}(T)$ can be updated as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T),$$

with

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n(T-1) + K_n(T)(\tilde{y}_n(T) - \tilde{X}_n(T)'\hat{\theta}_n(T-1)) \tag{7.45}$$

and

$$\begin{aligned}
\hat{\theta}_n^{ADMM,(k+1)}(T) &= \phi_n(T)\left\{\rho_1 \Delta_{z_n,\lambda_n}^{(k+1)} - \Delta_{\lambda_n}^{(k+1)}\right\}, \\
\Delta_{\lambda_n}^{(k+1)} &= \delta_{n,1}^{(k)} - \lambda_n \delta_{n,1}(T-1), \\
\Delta_{z_n,\lambda_n}^{(k+1)} &= z_n^{(k)} - \lambda_n z_n(T-1),
\end{aligned} \tag{7.46}$$

where the extended measurement vector $\tilde{y}$ is introduced for dimension consistency and it is equal to

$$\tilde{y}_n(T) = \begin{bmatrix} y_n(T) & 0_{1\times n_\theta} \end{bmatrix}'. \tag{7.47}$$

On the one hand, $\hat{\theta}^{g,(k+1)}$, $z_n^{(k+1)}$ and $\delta_{n,2}^{(k+1)}$ should be computed on the cloud, as the global parameter is updated using the estimates collected from all the N agents (see (7.32)) and the updates of the auxiliary variables and the Lagrange multipliers (7.33) and (7.36), respectively, depend on the current global estimate. On the other hand, $\delta_{n,1}$ can be updated by the nth processor. However, under the hypothesis that the local computational power is limited, also $\delta_{n,1}^{(k+1)}$ should be computed on the cloud. The local estimate $\hat{\theta}_n^{RLS}$ can be updated (i) by each local processor, (ii) recursively and (iii) independently from quantities computed on the cloud. In addition, observe that (7.45) is independent from k. As a consequence, the clock scheduling the operations performed on the local processor is not influenced by the clock of the cloud.

Remark 7 *The method, summarized in Algorithm 18, requires the local processors to transmit $\hat{\theta}_n^{RLS}$ and ϕ_n to the cloud. On the other hand, the cloud has to communicate $\hat{\theta}_n$ to the nth local processor, $n = 1, \ldots, N$, once the chosen stopping criteria for ADMM is satisfied.* ■

Solution based on N2C transmissions

Instead of introducing $\hat{\theta}_n(T-1)$, the estimates of the local parameter $\hat{\theta}_n$ is updated considering

$$\hat{\theta}_n^{RLS}(T-1) = \phi_n(T-1)\mathcal{Y}_n(T-1),$$

with ϕ_n and $\mathcal{Y}_n$ defined as in equations (7.38) and (7.39), respectively.
It can be easily shown that $\hat{\theta}_n^{RLS}$ can be updated as in (7.26), with

$$\hat{\theta}_n^{ADMM,(k+1)}(T) = \phi_n(T)\left(\phi_1 z_n^{(k)} - \delta_{nn,1}^{(k)}\right), \tag{7.48}$$

and

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n^{RLS}(T-1) + K_n(T)(\tilde{y}_n(T) - \tilde{X}_n(T)'\hat{\theta}_n^{RLS}(T-1)), \tag{7.49}$$

with the extended regressor and the extended measurement vector defined as in (7.43) and (7.47), respectively.
The choice of replacing $\hat{\theta}_n(T-1)$ with $\hat{\theta}_n^{RLS}(T-1)$ allows $\hat{\theta}_n^{RLS}$ to be updated at a local level, without requiring the transmission of any additional information from the cloud. We thus use a Node-to-Cloud (N2C) communications, as summarized in Figure 94.

7.3.1 Cloud-aided estimation over a fleet of vehicles

Suppose that we are interested in estimating the masses and drag coefficients of N vehicles. Furthermore, assume that the considered vehicles are of the same kind, so that we can hypothesize that the drag coefficients are equal for all the N cars.
Consider the equation describing the longitudinal dynamics of a vehicle [94], *i.e.*,

$$ma_x = F_x - F_a - R_x - mg\sin\gamma, \tag{7.50}$$

with

$$F_a = \frac{1}{2}\varrho C_d A_f (v_x + v_w)^2, \tag{7.51}$$

$$R_x = fmg, \tag{7.52}$$

Algorithm 18 ADMM-RLS for nonlinear consensus

Input: Data flow $\{X_n(1), y_n(1)\}, \{X_n(2), y_n(2)\}, \ldots$, initial global estimate $\hat{\theta}_{\text{o}}^g$, $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n^{RLS}(0)$, Lagrange multipliers $\delta_{n,1}^{\text{o}}$ and $\delta_{n,2}^{\text{o}}$ and auxiliary variables $z_{n,\text{o}}$, $n = 1, \ldots, N$, $\{\lambda_n\}_{n=1}^N$, parameters $\rho_1, \rho_2 \in \mathbb{R}^+$.

1. **for** $t = 1, 2, \ldots$ **do**

   Local

   1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $\tilde{X}_n(t)$ as in (7.43);

   1.1.2. **compute** $K_n(t)$ and $\phi_n(t)$ with (7.41) - (7.42);

   1.1.3. **compute** $\hat{\theta}_n^{RLS}(t)$ with (7.45) (N2C2N) or (7.49) (N2C);

   1.2. **end for**;

   Global

   1.1. **do**

   1.1.1. **set** $A_n^{(k)} \leftarrow \left.\frac{\partial F(z_n)}{\partial z_n}\right|_{z_n = z_n^{(k)}}$, $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}_n^{ADMM,(k+1)}(t)$ with (7.46) (N2C2N) or (7.48) (N2C), $n = 1, \ldots, N$;

   1.1.3. **compute** $\hat{\theta}_n^{(k+1)}(t)$ with (7.49), $n = 1, \ldots, N$;

   1.1.4. **compute** $z_n^{(k+1)}$ with (7.33), $n = 1, \ldots, N$;

   1.1.5. **compute** $\hat{\theta}^{g,(k+1)}$ with (7.34);

   1.1.6. **compute** $\delta_{n,1}^{(k+1)}$ with (7.35), $n = 1, \ldots, N$;

   1.1.7. **compute** $\delta_{n,2}^{(k+1)}$ with (7.36), $n = 1, \ldots, N$;

   1.2. **until** a stopping criteria is satisfied (*e.g.* the maximum number of iterations is attained);

2. **end**.

Output: Local estimates $\hat{\theta}_n^{RLS}(t)$, global refinement of the local estimates $\hat{\theta}_n(t)$, $n = 1, \ldots, N$, and global parameter's estimate $\hat{\theta}^g(t)$.

Table 27: Parameters charactering the longitudinal motion of the nth vehicle.

name	physical meaning and UoM	value
m_n	mass (kg)	$1500 + \mu_n$ $\mu_n \sim \mathcal{N}(0, 10^4)$
ϱ	air density (kg/m^3)	1.18
C_d	drag coefficient	0.4
A_f	frontal area (m^2)	3
f	rolling resistant coefficient	0.015
g	gravitational acceleration (m/s^2)	9.81
γ_n	road grade (deg)	0

and F_a N), F_x (N) and R_x (N) represent the aerodynamic drag force, the longitudinal tire force and the force due to rolling resistance, respectively. In equation (7.51), v_x (m/s) and v_w (m/s) are the vehicle and the wind velocity, respectively.
The force $F_{x,n}$ and the velocity $v_{x,n}$ for each vehicle of the fleet, with $n = 1, \ldots, N$, are supposed to be measured, while $v_{w,n}$ is assumed to be zero for all the vehicles during the considered estimation horizon. The values and physical meaning of the other parameters in (7.50)-(7.52) are reported in Table 27.

Instead of considering the continuous-time model for the longitudinal dynamics in (7.50), we introduce the corresponding discrete-time model

$$\check{y}_n(t) = (X_n(t))'\theta_t + e_n(t), \tag{7.53}$$

with

$$\begin{aligned} \check{y}_n(t) &= y_n(t) - T_s R_{x,n}, \\ X_n(t) &= T_s \begin{bmatrix} F_{x,n}(t-1) & -0.5\varrho A_f y_n(t-1)^2 \end{bmatrix}', \\ \theta_n &= \begin{bmatrix} \frac{1}{m_n}, & \frac{C_d}{m_n} \end{bmatrix}'. \end{aligned} \tag{7.54}$$

The measured output $y_n(t)$ is equal to $v_{x,n}(T_s t)$ and the sampling time is $T_s = 0.1$ s. The zero-mean, Gaussian distributed, additive noise sequence $e_n \sim \mathcal{N}(0, 0.1)$ is also introduced.

Based on the definition of the parameter vector in (7.54), observe that

the estimates of the masses can be retrieved as

$$\hat{m}_n = \hat{\theta}_{n,1}^{-1}, \quad n = 1, \dots, N,$$

while the drag coefficients can be identified introducing $\theta^g = C_d$ and imposing the condition

$$\theta^g = F(\theta_n) = (\theta_{n,1})^{-1}\theta_{n,2}. \tag{7.55}$$

To guarantee that F is always well defined, we should enforce $\theta_{n,1} \neq 0$ but, due to the physical meaning of the unknowns, we impose $\theta_n \in \mathcal{C}_n$, with $\mathcal{C}_n \subset \mathbb{R}^+ \times \mathbb{R}^+$.

In estimating the masses and drag coefficient of the N vehicles, the performance of ADMM-RLS (N2C) is evaluated considering two slightly different settings. In particular, we suppose that (i) all the vehicles are moving except for few initial steps and then that (ii) some of the cars are not used for part of the estimation horizon. With the second test, we are able to study the effect of sudden changes in the agents' behavior on the accuracy of the estimates.
The quality of the global estimate is assessed through the RMSE, given by

$$\text{RMSE}_{C_d} = \sqrt{\frac{\sum_{t=1}^{T}(\hat{C}_d(t) - C_d)^2}{T}}. \tag{7.56}$$

Independently from the setting, we have chosen $\lambda_n = 1$, $\phi_n(0) = 10^{-3} I_{n_\theta}$, $z_{n,\text{o}} = \hat{\theta}_n(0)$, with $\hat{\theta}_n(0) = \theta_n + \vartheta_n$, $\vartheta \sim \mathcal{N}(0_{n_\theta \times 1}, \Theta)$ and

$$\Theta = \begin{bmatrix} 10^4 & 0 \\ 0 & 10^{-4} \end{bmatrix}.$$

The initial Lagrange multipliers and local parameters are chosen as: $\delta^{\text{o}}_{n,1} = \delta I_{n_\theta}$, $\delta^{\text{o}}_{n,2} = \delta I_{n_g}$, for $n = 1, \dots, N$, with $\delta = 10^{-8}$, $n_\theta = 2$ and $n_g = 1$ and $\hat{\theta}^g_{\text{o}} = \theta^g + \vartheta^g$, with $\vartheta^g \sim \mathcal{N}(0, 10^{-2})$.

Setting 1

We consider fleets of vehicles of different dimension N, with $N = \{2, 10, 50, 100\}$, where the agents are characterized by velocity profiles similar to the one

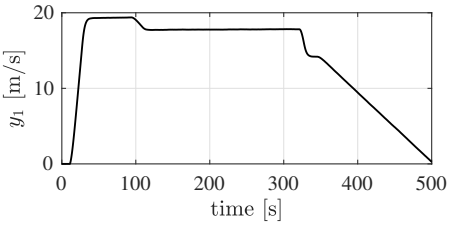
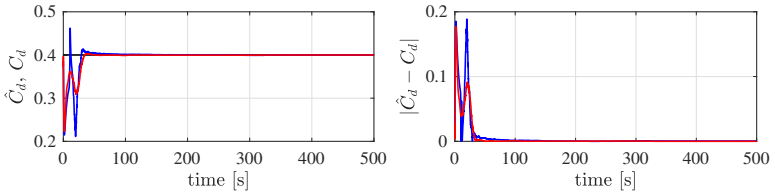


Figure 86: Velocity profile of the nth vehicle, with $n = 1$.



(a) $\hat{C}_d$ vs C_d (b) $|\hat{C}_d - C_d|$

Figure 87: Estimated drag coefficient vs N. True C_d (black), estimated drag coefficient with $N = 1$ (blue), estimated drag coefficient with $N = 100$ (red).

reported in Figure 86. A car belonging to a lower dimensional fleet always belongs also to the fleets composed by more agents.

Selecting $\rho_1 = 1$ and $\rho_1 = 10^{-3}$, Figure 87 shows the the global estimate $\hat{C}_d$ obtained using one vehicle[1], *i.e.*, $N = 1$, and the biggest fleet in size ($N = 100$), among the ones considered in the example. Note that, using a fleet composed of $N = 100$ vehicles, the convergence of the global estimate to the actual value of the C_d tends to be slightly faster. Furthermore, the estimation error over the entire estimation horizon is slightly reduced, as

$$\text{RMSE}_{C_d} = \begin{cases} 0.03 \text{ if } N = 1, \\ 0.02 \text{ if } N > 1. \end{cases}$$

Moreover, the drag coefficient estimated using the data collected from more than one vehicle are less affected by the characteristics of the local datasets. Note that the obtained estimate $\hat{C}_d$ in Figure 87 satisfy the constraint $C_d \in \mathbb{R}^+$.

The estimated mass $\hat{m}_1$ is compared with the actual mass of the 1st ve-

[1]The estimates are computed using standard RLS [72].

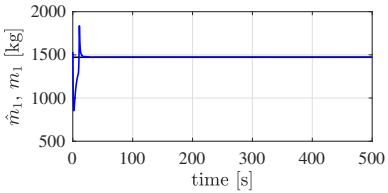


Figure 88: $\hat{m}_1$ vs m_1. True (black), estimated (blue).

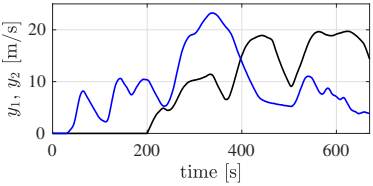
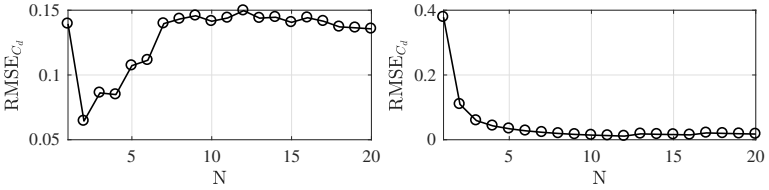


Figure 89: Velocity profiles of the 1st and 2nd available vehicles.



(a) Horizon: $\Upsilon = 670$ s. (b) Horizon: $\upsilon = 490$ s.

Figure 90: RMSE$_{C_d}$ vs N.

hicle in Figure 88. We report the mass obtained for $N = 1$, *i.e.*, applying standard RLS, because the differences between the estimate reported in Figure 88 and $\hat{m}_1$ computed using the consensus-based collaborative estimation scheme are negligible. This result is expected as the mass is a purely local parameter and we are using an N2C transmission scheme.

Setting 2

Suppose that the measurements gathered from $N \in \{2, \ldots, 20\}$ cars are available and that their velocity profiles are similar to the ones presented in Figure 89. The velocity y_1 of the 1st vehicle is zero for 200 (s), *i.e.*, for 30% of the estimation horizon $\Upsilon = 670$(s).

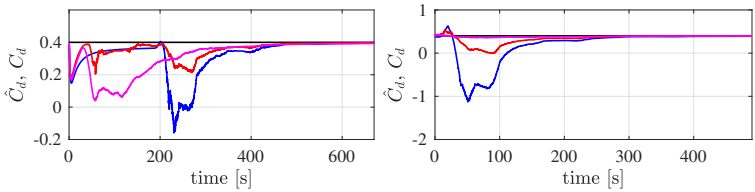
Figure 7.90(a) shows the RMSEs computed increasing the number of

Table 28: N vs instant t at which the estimation error satisfies $|\hat{C}_d - C_d| \leq \varsigma$. It provides an quantitative indication on the convergence of the estimate $\hat{C}_d$

	N		
	1	2	20
$\varsigma = 0.005$ & Υ	635.5 s	573.2 s	-
$\varsigma = 0.005$ & υ	-	457.6 s	353.0 s
$\varsigma = 0.01$ & Υ	555.8 s	466.9 s	523.5 s
$\varsigma = 0.01$ & υ	451.0 s	302.9 s	286.3 s

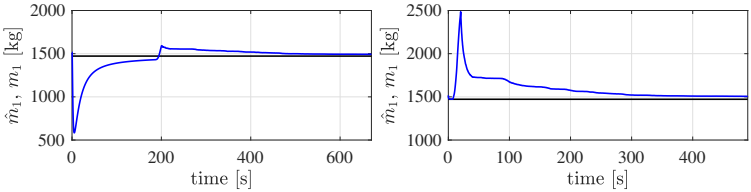
agents composing the fleet and using all the data collected over Υ, with $\rho_1 = 100$ and $\rho_2 = 10^{-3}$. The RMSE does not decrease as N increases, differently from what we would have expected. Nevertheless, when only the data collected from $t_o = 180$ (s) are used, the RMSE actually decreases if N increases as reported in Figure 7.90(b). It thus seems that the accuracy of the global estimate is enhanced if we start estimating the unknowns when most of the vehicles in the fleet are moving. In particular, for $N \geq 3$, the RMSE obtained considering the reduced horizon $\upsilon = \Upsilon - 180$ s is smaller than the one obtained using the information collected over the whole estimation horizon $\Upsilon = 670$ (s). Furthermore, observe that the RMSE obtained with $N > 1$ is generally smaller than RMSE_{C_d} obtained using one vehicle, if the reduced horizon υ is considered.

These results can be further validated looking at the estimates $\hat{C}_d$ obtained for N equal to $\{1, 2, 20\}$, shown in Figure 91 and considering the instants at which the error between the actual and estimated drag coefficients are under a threshold ς, which are reported in Table 28. Using all the data collected over Υ, the estimate obtained for $N = 1$ tends to diverge from C_d when the vehicle start to move, probably because of the sudden change in its velocity profile. The effect of such a change is mitigated by the use of multiple cars. However, by increasing N the convergence of $\hat{C}_d$ to the true value of the drag coefficient C_d is slower, as consensus over multiple vehicles has to be attained. On the other hand, if the data collected over $\upsilon = 490$ (s) are considered, increasing the number of vehicles in the fleet N enables to speed up convergence. The higher N is, the less is the number of steps in which the estimate is saturated.



(a) Horizon: $\Upsilon = 670$ s. (b) Horizon: $\upsilon = 490$ s.

Figure 91: Estimated drag coefficient vs N. True C_d (black), estimate obtained with $N = 1$ (blue), estimate obtained with $N = 2$ (red), estimated obtained with $N = 20$ (magenta).



(a) Horizon: $\Upsilon = 670$ s. (b) Horizon: $\upsilon = 490$ s.

Figure 92: $\hat{m}_1$ vs m_1.True (black), estimated (blue).

Moreover, observe that when $\hat{C}_d$ is negative only when for $N = 1$ and standard RLS [72] is used to compute the estimates, as the constraints on the value of the local parameters are not imposed.

As it concerns the estimate of the purely local parameters, *i.e.*, the masses, $\hat{m}_1$ obtained for $N = 20$ is reported in Figure 92 for both the cases when the complete estimation horizon Υ and the reduced one υ are considered. When the data collected over Υ are used, the convergence of $\hat{m}_1$ to the actual mass is faster then the one of the obtained considering the reduced horizon. However, as expected, this is generally not true. This can be also noticed looking at the final errors in the mass estimates plotted in Figure 93.

7.4 Case study 3. Multiple classes estimation

In many practical cases, θ^g is shared only by subsets of the N agents, $\mathscr{M}_\mu \subset \{1, \dots, N\}$, with $\mu \in \{1, \dots, M\}$.As an example, consider the

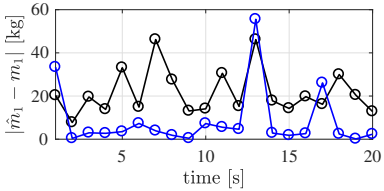


Figure 93: $|\hat{m}_1 - m_1|$ at the end of the estimation horizon. Complete horizon Υ (black), reduced horizon υ (blue).

problem of modeling the aging of a mechanical/electrical component. Given N theoretically equal devices, their degradation might be different due to multiple causes (*e.g.*, environmental conditions and user behavior). It is thus necessary to retrieve different aging models.
Under the hypothesis that similar conditions of use lead to the same degradation, it seems legitimate to retrieve $M < N$ aging models, instead of identifying one model for each of the N available devices.

To tackle this problem, the consensus constraint in (7.1) has thus to be modified as

$$F(\theta_n) = \theta^g \rightarrow \sum_{\mu=1}^{M} \mathbf{1}_{[\mathsf{m}_n(t)=\mu]}(P\theta_n - \theta_\mu^g) = 0, \tag{7.57}$$

where $\{\theta_\mu^g \in \mathbb{R}^{n_g}\}_{\mu=1}^M$ indicates the global parameter vectors and $P \in \mathbb{R}^{n_g \times n_\theta}$ is supposed to be known a priori. The quantity $\mathsf{m}_n(t)$ represents the class sequence associated with the nth agent at time t. Note that the consensus constraint in (7.57) depends on time, as the class associated with each agent might change over the estimation horizon T. The function $\mathbf{1}_{[\mathsf{m}_n(t)=\mu]}$ is equal to

$$\mathbf{1}_{[\mathsf{m}_n(t)=\mu]} = \begin{cases} 1 \text{ if } \mathsf{m}_n(t) = \mu \\ 0 \text{ otherwise,} \end{cases} \tag{7.58}$$

with $\sum_{\mu=1}^{M} \mathbf{1}_{[\mathsf{m}_n(t)=\mu]} = 1$ implying that, at each step t, the nth agent can be associated to one model class only.

If the sets $\mathscr{M}_\mu$, $\mu = 1, \dots, M$, are known a priori, M separate problems can be solved to estimate the unknown parameters. However, both (i) the model parameters and (ii) the sets $\{\mathscr{M}_\mu\}_{\mu=1}^M$ are supposed to be

unknown.

Instead, the number of model classes M is supposed to be known a priori. If M is unknown, it can be chosen through cross-validation or other tuning procedures.

Consider the general problem (7.1), with the consensus constraints defined in (7.57). The unknowns that have to be estimated from data at each step $t \in \{1, \ldots, T\}$ are:

1. the local parameters θ_n, $n = 1, \ldots, N$;
2. the global parameters $\{\theta_\mu^g\}_{\mu=1}^M$;
3. the class sequences associated to the local agents $\{\mathsf{m}_n(t)\}_{t=1}^T$, $n = 1, \ldots, N$.

Aiming at using ADMM to solve the addressed multi-class problem in (7.1) and constraints (7.57), we introduce the associated augmented Lagrangian given by

$$\mathcal{L} = \sum_{n=1}^{N} g_n(z_n) + \delta'_{n,1}\varepsilon_{n,1} + \frac{\rho_1}{2}\|\varepsilon_{n,1}\|_2^2 + \\ + \sum_{t=1}^{T}\left[f_n(\theta_n) + \delta_{n,2}(t)' V_n(\mathsf{m}_n(t)) + \frac{\rho_2}{2}\|V_n(\mathsf{m}_n(t))\|_2^2\right], \tag{7.59}$$

with

$$\varepsilon_{n,1} = \theta_n - z_n, \tag{7.60}$$

$$V_n(\mathsf{m}_n(t)) = \sum_{\mu=1}^{M} \mathbf{1}_{[\mathsf{m}_n(t)=\mu]}(P\theta_n - \theta_\mu^g), \tag{7.61}$$

and $\delta_{n,1}$ and $\delta_{n,2}(t)$ indicating the Lagrange multipliers.

Equation (7.59) can be simplified if either the parameters or the class sequences $\{\mathsf{m}_n(t)\}_{t=1}^T$, for $n = 1, \ldots, N$, are known. Based on this consideration, we propose a two-stage approach to address problem (7.1) with the consensus constraint given by equation (7.57). In particular, the steps to be performed are

PE: compute the estimates for $\{\theta_\mu^g\}_{\mu=1}^M$ and $\{\theta_n\}_{n=1}^N$, with the class sequence $\{\mathsf{m}_n(t)\}_{t=1}^T$, $n = 1, \dots, N$, fixed;

CE: estimate the class of each agent $\{\mathsf{m}_n(t)\}_{t=1}^T$, $n = 1, \dots, N$ with both the global and the local parameters fixed.

The order in which the two steps are performed should not influence the final result of the estimation procedure and it only depends on the chosen initial condition. On the one hand, when no assumption is made on $\hat{m}_n(1)$, while $\hat{\theta}_n$ and $\{\hat{\theta}_\mu^g\}_{\mu=1}^M$ are initialized, CE has to be run first. On the other hand, PE is the first to be performed if an estimate for the class of each agent at $t = 1$ is provided.

Based on how the description of the two steps, PE and CE should be performed on a batch of data. Nonetheless, our objective is to update the estimates recursively and thus PE and CE has to be slightly modified to be performed in a recursive fashion.

7.4.1 Step PE: estimating the unknown parameters

Consider the augmented Lagrangian in (7.59). Assuming that the class sequence is known, *i.e.*, $\mathsf{m}_n(t) = \hat{\mathsf{m}}_n(t)$, for $t = 1, \dots, T$ and $n = 1, \dots, N$, the augmented Lagrangian can be modified as

$$\mathcal{L} = \sum_{n=1}^{N} \left\{ \mathcal{L}_n(\theta_n, z_n, \delta_{n,1}, \{\theta^g_{\hat{\mathsf{m}}_n(t)}, \delta_{n,2}(t)\}_{t=1}^T) \right\}, \tag{7.62}$$

with

$$\begin{aligned} \mathcal{L}_n = {} & g_n(z_n) + \delta'_{n,1}(\varepsilon_n) + \frac{\rho_1}{2}\|\varepsilon_n\|_2^2 + \\ & + \sum_{t=1}^{T} \left[f_n(\theta_n) + \delta_{n,2}(t)' V_n(\hat{\mathsf{m}}_n(t)) + \frac{\rho_2}{2}\|V_n(\hat{\mathsf{m}}_n(t))\|_2^2 \right], \end{aligned} \tag{7.63}$$

$$V_n(\hat{\mathsf{m}}_n(t)) = P\theta_n - \theta^g_{\hat{\mathsf{m}}_n(t)}. \tag{7.64}$$

Then, the ADMM iterations to be performed are

$$\hat{\theta}_n^{(k+1)}(T) = \underset{\theta_n}{\operatorname{argmin}} \left\{ \mathcal{L}_n(\theta_n, z_n^{(k)}, \delta_{n,1}^{(k)}, \{\hat{\theta}_{\hat{\mathfrak{m}}_n(t)}^{g,(k)}, \delta_{n,2}^{(k)}(t)\}_{t=1}^{T}) \right\}, \tag{7.65}$$

$$z_n^{(k+1)} = \mathcal{P}_{\mathcal{C}_n} \left(\hat{\theta}_n^{(k+1)}(T) + \frac{1}{\rho_1} \delta_{n,1}^{(k)} \right), \tag{7.66}$$

$$\hat{\theta}_\mu^{g,(k+1)} = \underset{\theta_\mu^g}{\operatorname{argmin}} \left\{ \mathcal{L}_n(\hat{\theta}_n^{(k+1)}, z_n^{(k+1)}, \delta_{n,1}^{(k)}, \hat{\theta}_\mu^g, \{\delta_{n,2}^{(k)}(t)\}_{t=1}^{T}) \right\}, \tag{7.67}$$

$$\delta_{n,1}^{(k+1)} = \delta_{n,1}^{(k)} + \rho_1(\hat{\theta}_n^{(k+1)}(T) - z_n^{(k+1)}), \tag{7.68}$$

$$\delta_{n,2}^{(k+1)}(T) = \delta_{n,2}^{(k)}(T) + \rho_2(P\hat{\theta}_n^{(k+1)}(T) - \hat{\theta}_{\hat{\mathfrak{m}}_n(t)}^{g,(k+1)}), \tag{7.69}$$

with $\mu = 1, \dots, M$ and

$$\varepsilon_n^{(k)} = \theta_n - z_n^{(k)}, \tag{7.70}$$

$$\hat{V}_n^{(k)}(t) = P\theta_n - \hat{\theta}_{\hat{\mathfrak{m}}_n(t)}^{g,(k)}, \tag{7.71}$$

$$\hat{V}_n^{g,(k+1)}(t) = \left(P\hat{\theta}_n^{(k+1)}(T) \; - \; \theta_{\hat{\mathfrak{m}}_n(t)}^g \right). \tag{7.72}$$

The estimates of the global parameters for each class, $\{\hat{\theta}_\mu^{g,(k+1)}\}_{\mu=1}^{M}$, are computed solving explicitly (7.67). In particular, $\hat{\theta}_\mu^g$ is the solution of equation

$$\sum_{n=1}^{N} \sum_{t=1}^{T} \left[\mathbf{1}_{[\hat{\mathfrak{m}}_n(t)=\mu]} \delta_n^{(k)}(t) + \rho \mathbf{1}_{[\hat{\mathfrak{m}}_n(t)=\mu]} \left(\sum_{\eta=1}^{M} \mathbf{1}_{[\hat{\mathfrak{m}}_n(t)=\eta]} (P\hat{\theta}_n^{(k+1)} - \theta_\eta^g) \right) \right] = 0,$$

with respect to $\hat{\theta}_\mu^g$. This equation can be further simplified as

$$\sum_{t=1}^{T} \sum_{n:\hat{\mathfrak{m}}_n(t)=\mu} \left[-\delta_n^{(k)}(t) - \rho \left(P\hat{\theta}_n^{(k+1)}(T) - \theta_\mu^g \right) \right] = 0, \tag{7.73}$$

exploiting the definition of the indicator function 1 in (7.58). This equality is satisfied at each step t if the following holds:

$$\sum_{n:\hat{\mathfrak{m}}_n(t)=\mu} \left[-\delta_n^{(k)}(t) - \rho \left(P\hat{\theta}_n^{(k+1)}(T) - \theta_\mu^g \right) \right] = 0.$$

holds. Let $\mathscr{M}_\mu(t)$ be the set of agents assigned to mode μ at time t, $\{n \in \{1, \dots, N\} : \hat{\mathfrak{m}}_n(t) = \mu\}$. The global parameter $\hat{\theta}_\mu^{g,(k+1)}$ can be computed

as

$$\hat{\theta}_{\mu}^{g,(k+1)} = \frac{1}{|\mathscr{M}_{\mu}(t)|} \sum_{n \in \mathscr{M}_{\mu}(t)} \left[P\hat{\theta}_{n}^{(k+1)}(T) + \frac{1}{\rho_2} \delta_{n,2}^{(k)}(t) \right]. \tag{7.74}$$

As expected, instead of using the estimates acquired from all the available data sources, only a subset of agents is considered to update the μth global parameter vector. Furthermore, using (7.73) to update the global estimates, we do not account for the past estimates of $\hat{\theta}_{\mu}^{g}$.

As it concerns the local estimates $\hat{\theta}_n$, it can be proven that the close form solution of equation (7.65) is

$$\hat{\theta}_{n}^{(k)}(T) = \phi_n(T) \left\{ \mathcal{Y}_n(T) + \xi_{n}^{(k)}(T) \right\}, \tag{7.75}$$

$$\xi_{n}^{(k)}(t) = \rho_1 z_{n}^{(k)} - \delta_{n,1}^{(k)} + \sum_{\tau=1}^{t} P' \left(\rho_2 \hat{\theta}_{\hat{\mathfrak{m}}_n(\tau)}^{g,k} - \delta_{n,2}^{(k)}(\tau) \right),$$

$$\mathcal{Y}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) y_n(\tau),$$

$$\mathcal{X}_n(t) = \sum_{\tau=1}^{t} \lambda_n^{t-\tau} X_n(\tau) (X_n(\tau))',$$

$$\phi_n(t) = \left(\mathcal{X}_n(t) + t\rho_2 P'P + \rho_1 I_{n_\theta} \right)^{-1}.$$

With the final objective of finding a recursive solution for the consensus-based multi class estimation problem relying on N2C2N transmissions, we introduce $\hat{\theta}_n(T-1)$ given by

$$\hat{\theta}_n(T-1) = \phi_n(T-1) \left\{ \mathcal{Y}_n(T) + \xi_n(T-1) \right\}, \tag{7.76}$$

with

$$\xi_n(T-1) = \rho_1 z_n(T-1) - \delta_{n,1}(T-1) + \sum_{t=1}^{T-1} P'(\rho_2 \hat{\theta}_{\hat{m}_n(t)}^{g}(T-1) - \delta_{n,2}(t))$$

and where $z_n(T-1)$, $\delta_{n,1}(T-1)$, $\delta_{n,2}(t) = \delta_{n,2}(t|T-1)$ and $\hat{\theta}_{\hat{m}_n(t)}^{g}(T-1)$, for $t = 1, \ldots, T-1$, are the auxiliary variables, the Lagrange multipliers and the global estimates obtained at step $T-1$, respectively.

Also in this case, it can be easily shown that matrix ϕ_n can be recursively updated as

$$\mathcal{R}_n(T) = \lambda_n I + (\tilde{X}_n(T))' \phi_n(T) \tilde{X}_n(T), \tag{7.77}$$

$$K_n(T) = \phi_n(T-1)\tilde{X}_n(T)(\mathcal{R}_n(T))^{-1}, \tag{7.78}$$

$$\phi_n(T) = \lambda_n^{-1}(I_{n_\theta} - K_n(T)(\tilde{X}_n(T)'))\phi_n(T-1), \tag{7.79}$$

with the extended regressor $\tilde{X}_n \in \mathbb{R}^{n_{\tilde{x}}}$ defined as

$$\tilde{X}_n(T) = \left[\, X_n(T) \; \sqrt{(T-(T-1)\lambda_n)\rho_2} P' \; \sqrt{(1-\lambda_n)\rho_1} I_{n_\theta} \, \right], \tag{7.80}$$

and $n_{\tilde{X}} = n_y + n_\theta + n_g$.

As in the others Sections (*e.g.*, Section 7.3), it can be proven that the local estimates can be updated as

$$\hat{\theta}_n^{(k+1)}(T) = \hat{\theta}_n^{RLS}(T) + \hat{\theta}_n^{ADMM,(k+1)}(T),$$

where

$$\hat{\theta}_n^{ADMM,(k+1)}(T) = \phi_n(T)\xi_n(T|T), \tag{7.81}$$

$$\xi_n(t|t) = P' \left[\sum_{t=1}^{T-1} \left(\rho_2 \Delta_{g,\lambda_n}^{(k+1)}(t) - \Delta_{2,\lambda_n}^{(k+1)}(t) \right) + \rho_2 \hat{\theta}_{\hat{m}_n(T)}^{g,(k)} - \delta_{n,2}^{(k)}(T) \right] + \\ - \Delta_{1,\lambda_n}^{(k+1)} + \rho_1 \Delta_{z_n,\lambda_n}^{(k+1)}, \tag{7.82}$$

$$\Delta_{z_n,\lambda_n}^{(k+1)} = \hat{z}_n^{(k)} - \lambda_n \hat{z}_n(T-1)$$
$$\Delta_{1,\lambda_n}^{(k+1)} = \delta_{n,1}^{(k)} - \lambda_n \delta_{n,1}(T-1)$$
$$\Delta_{g,\lambda_n}^{(k+1)}(t) = \hat{\theta}_{\hat{m}_n(t)}^{g,(k)} - \lambda_n \hat{\theta}_{\hat{m}_n(t)}^{g}(T-1), \; t = 1, \ldots, T-1$$
$$\Delta_{2,\lambda_n}^{(k+1)}(t) = (1-\lambda_n)\delta_{n,2}(t), \; t = 1, \ldots, T-1,$$

and

$$\hat{\theta}_n^{RLS}(T) = \hat{\theta}_n(T-1) + K_n(T)(\tilde{y}_n(T) - \tilde{X}_n(T)'\hat{\theta}_n(T-1), \tag{7.83}$$

with

$$\tilde{y}_n(T) = \left[\, y_n(T)' \; 0_{1\times(n_g+n_\theta)} \, \right]'.$$

The update for $\hat{\theta}_n^{ADMM}$ is obtained assuming that, at time T, only $\delta_{n,2}(T|T)$ is updated, while $\delta_{n,2}(t|T)$ is equal to $\delta_{n,2}(t|t)$, for $t < T$.

7.4.2 Step CE: estimating the class sequences

Consider again the augmented Lagrangian (7.59). With fixed parameters $\{\theta_\mu^g(t)\}_{\mu=1}^M$, for $t = 1, \dots, T$, the class sequence $\{\mathsf{m}_n(t)\}_{t=1}^T$ for each agent $n \in \{1, \dots, N\}$ can then be identified minimizing (7.59) with respect to $\{\mathsf{m}_n(t)\}_{t=1}^T$, *i.e.*, solving the optimization problem

$$\min_{\{\mathsf{m}_n(t)\}_{t=1}^T} \mathscr{F}(\{\mathsf{m}_n\}_{n=1}^N) \tag{7.84}$$

with

$$\mathscr{F}(\{\mathsf{m}_n\}_{n=1}^N) = \sum_{t=1}^T \left[\delta_{n,2}(t)' V_n(\mathsf{m}_n(t)) + \frac{\rho_2}{2} \|V_n(\mathsf{m}_n(t))\|_2^2\right]$$

and

$$V_n(\mathsf{m}_n(t)) = \sum_{\mu=1}^M \mathbf{1}_{[\mathsf{m}_n(t)=\mu]}(P\hat{\theta}_n(t) - \hat{\theta}_\mu^g(t)).$$

By introducing $h(\mathsf{m}_n(t), t)$, which is equal to

$$h(\mathsf{m}_n(t), t) = \delta_{n,2}(t)'(V_n(\mathsf{m}_n(t))) + \frac{\rho_2}{2}\|V_n(\mathsf{m}_n(t))\|_2^2. \tag{7.85}$$

The original problem (7.84) can then be written as:

$$\min_{\mathsf{m}_n(T)} \{h(\mathsf{m}_n(T), T) + \mathcal{H}(T-1)\}$$

$$\mathcal{H}(T-1) = \min_{\{\mathsf{m}_n(t)\}_{t=1}^{T-1}} \sum_{t=1}^{T-1} h(\mathsf{m}_n(t), t)$$

and the mode estimate can be computed as

$$\hat{\mathsf{m}}_n(T-1) = \operatorname*{argmin}_{\mathsf{m}_n(T-1)} \mathcal{H}(T-1).$$

To find recursive formulas for the estimation of the class sequences $\{\hat{\mathsf{m}}_n(t)\}_{t=1}^T$, $n = 1, \dots, N$, we assume that $\mathsf{m}_n(t-1)$ is known at t, *i.e.*, we perform the approximation $\mathsf{m}_n(t-1) \approx \hat{\mathsf{m}}_n(t-1)$. Then, $\hat{\mathsf{m}}_n(t)$ is

estimated minimizing $h(\mu, t)$ with respect to $\mu \in \{1, \ldots, M\}$.
To compute $\hat{\mathsf{m}}_n(t)$, we evaluate

$$h(\mu, t) = \delta_n(t)'(P\hat{\theta}_n(t) - \hat{\theta}^g_\mu(t)) + \frac{\rho_2}{2}\|P\hat{\theta}_n(t) - \hat{\theta}^g_\mu(t)\|_2^2, \tag{7.86}$$

for all the possible classes $\mu \in \{1, \ldots, M\}$ and then we select the class associated with the least $h(\mu, t)$.

A tentative implementation scheme for the proposed method is presented in Algorithm 19. At Step 1.3 of the proposed implementation, the class of each agent is predicted on the basis of the local estimates and the Lagrange multipliers obtained at the current step.

Remark 8 *Equation (7.84) depends on $\delta'_n V_n(\mathsf{m}_n(t))$. To reduce its influence, ρ_2 has to be big enough for $\|V_n(\mathsf{m}_n(t))\|_2^2$ to dominate over $\delta'_{n,2}V_n(\mathsf{m}_n(t))$in (7.84). A possible strategy to avoid $\delta'_{n,2}V_n(\mathsf{m}_n(t)) > \rho_2\|V_n(\mathsf{m}_n(t))\|_2^2$ is to choose $\rho_2 \gg max(\{\delta^o_{n,2}\}_{n=1}^N)$. It has to be pointed out that ρ_2 in equation (7.84) has a different meaning with respect to the same parameter in equation (7.62). It is consequently advisable to consider different values for ρ_2 in the execution of step PE and CE.* ■

On the one hand, the global estimates have to be updated on the cloud as the update in (7.74) depends on the "partial" estimates collected from all the agents. Due to the dependence on $\{\hat{\theta}^g_\mu\}_{\mu=1}^M$ of equations (7.69) and (7.85), it is also advisable to compute $\delta^{(k)}_{n,2}(t)$ and $\hat{m}_n(t)$ on the cloud. On the other hand, both z_n and $\delta_{n,1}$, $n = 1, \ldots, N$, can be updated locally. However, under the hypothesis that the local computational power and memory are limited, we propose to perform the updates in equations (7.66) and (7.68) on the cloud. See Algorithm 19.

Remark 9 *As outlined in Algorithm 19 and in Figure 94, the local processors have to communicate $\hat{\theta}^{RLS}_n$ and ϕ_n to the cloud at each time step. The cloud has to transmit $\hat{\theta}_n$ to the corresponding local processor.* ■

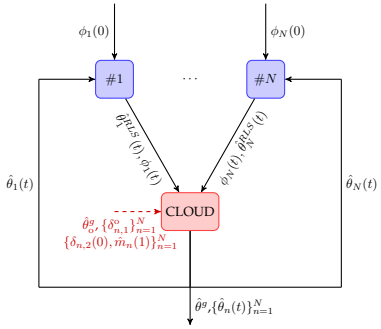


Figure 94: Scheme for the transmission characterizing Algorithm 19 at step t.

7.4.3 Example 2

Suppose that the measurement collected from N dynamical systems are available, with the behavior of the N agents characterized by the input/output relationship

$$y_n(t) = \theta^g_{\mathsf{m}_n(t)} y_n(t-1) + \theta_{n,2} u_n(t-1) + e_n(t), \tag{7.87}$$

$$\theta^g_{\mathsf{m}_n(t)} = \begin{cases} 0.8 \text{ if } \mathsf{m}_n(t) = 1 \\ 0.5 \text{ if } \mathsf{m}_n(t) = 2. \end{cases}$$

The local parameters $\theta_{n,2}$ are realizations of a random variable with distribution $\mathcal{N}(0.4, 4 \cdot 10^{-4})$. The input u_n is chosen as a sequence of i.i.d. elements uniformly distributed in the interval $[2, 3]$ and $e_n \sim \mathcal{N}(0, R_n)$ is a white noise sequence with $R_n \in [1,\ 4]$, which yields to SNR_n in the interval $[4.5,\ 14.3]$ dB.

The parameters $\theta_{n,1}$ changes over time depending on the agent's mode, with the agents are suppose to change class at a randomly chosen step in the interval $[100\ T]$, with $T = 5000$.

We assume that the parameters' estimates are constrained to the following ranges:

$$0.45 \leq \hat{\theta}_{n,1} \leq 0.85 \qquad 0.3 \leq \hat{\theta}_{n,2} \leq 0.5. \tag{7.88}$$

We initialize $\phi_n(0) = 10 I_{n_\theta}$, $\delta^{\text{o}}_{n,1} = 10^{-3} I_{n_\theta}$, $\delta_{n,2}(0) = 10^{-3} I_{n_g}$, for $n = 1, \ldots, N$, and $\theta^{g,\text{o}}_\mu = \theta^g_\mu + \vartheta^g_\mu$, with $\vartheta^g_\mu \sim \mathcal{N}(0, 1)$ for $\mu = 1, 2$. As the class of the system is supposed not to be known a priori, $\{\hat{m}_n(1)\}_{n=1}^N$

Table 29: $\{\text{RMSE}_\mu^g\}_{\mu=1}^2$ vs maximum number of iterations.

	$\mathscr{m}$			
	10	**20**	**50**	**100**
RMSE_1^g	0.023	0.023	0.017	0.017
RMSE_2^g	0.032	0.030	0.024	0.024

Table 30: $\{\text{RMSE}_\mu^g\}_{\mu=1}^2$ vs forgetting factor Λ.

	$\boldsymbol{\Lambda}$		
	0.975	**0.995**	**1**
RMSE_1^g	0.028	0.017	0.023
RMSE_2^g	0.022	0.024	0.088

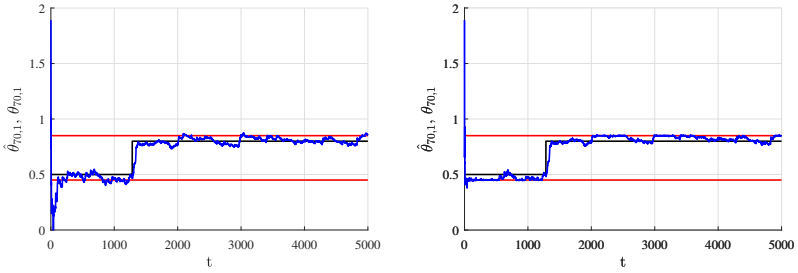
are randomly chosen.

We have decided to terminate the ADMM iterations when the maximum number of runs $\mathscr{m}$ is attained. The parameters $\mathscr{m}$, $\rho_1, \rho_2, \rho_3 \in \mathbb{R}^+$ and the forgetting factors $\{\lambda_n\}_{n=1}^N$ (see Algorithm 19) are left to be tuned. The parameters ρ_1 and ρ_2 influence the computed estimates of the local and global parameters, respectively, while ρ_3 affects the prediction of the model class. Imposing λ_n is equal to Λ, for all $n = 1, \dots, N$, the performance of the consensus-based estimation approach are evaluated changing the value of one of the aforementioned parameters, while the others are fixed.

With $\Lambda = 0.995$, $\rho_1 = 10^{-2}$, $\rho_2 = 10^{-5}$ and $\rho_3 = 1$, we evaluate the effect of performing a different number of iterations $\mathscr{m}$. As expected, RMSE_μ^g slightly decrease when $\mathscr{m}$ increases as shown in Table 29. Looking at the estimates reported in Figure 95, it can be noticed that the higher $\mathscr{m}$ is the more the conditions in (7.88) are enforced on the local estimates, thus causing the reduction in the RMSEs.

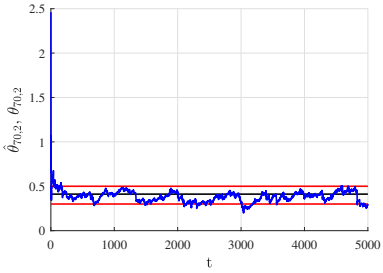
Table 30 reports the RMSEs obtained using different forgetting factors Λ, with $\mathscr{m}$= 50, $\rho_1 = 10^{-2}$, $\rho_2 = 10^{-5}$ and $\rho_3 = 1$. The computed estimates are less accurate if either $\Lambda = 0.975$ or $\Lambda = 1$, *i.e.*, when the estimates are computed relying mainly on the current measurements or if the time-varying nature of $\theta_{n,1} = \theta^g_{m_n(t)}$ is neglected.

The resulting RMSEs can be explained looking at the estimated classes $\hat{\mathfrak{m}}_n$, shown in Figure 96 that reports both $m_{70}(t)$ and $\hat{m}_{70}(t)$, for $t = 1, \dots, T$. If $\Lambda = 0.975$ and $\Lambda = 1$, the class estimate is characterized

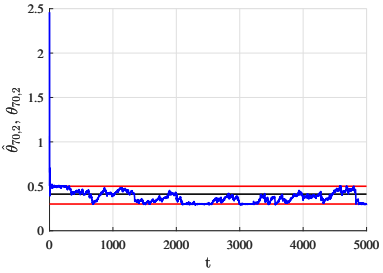


(a) $\hat{\theta}_{70,1}$ vs $\theta_{70,1}$: 20 iterations

(b) $\hat{\theta}_{70,1}$ vs $\theta_{70,1}$: 100 iterations



(c) $\hat{\theta}_{70,2}$ vs $\theta_{70,2}$: 20 iterations



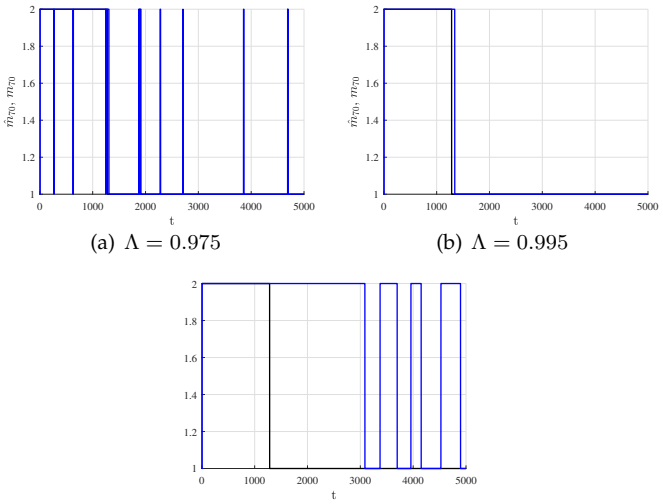
(d) $\hat{\theta}_{70,2}$ vs $\theta_{70,2}$: 100 iterations

Figure 95: θ_{70} vs maximum number of iterations. True (black), estimated (blue).

by multiple jumps, while m_{70} is accurately estimated when $\Lambda = 0.995$. A similar result is obtained fixing $\mathscr{m}$=50, $\Lambda = 0.995$, $\rho_1 = 10^{-2}$, $\rho_2 = 10^{-5}$ and changing the value of ρ_3.

In particular, the choice of $\rho_3 \leq 0.1$ causes the class estimate to be not accurate, as characterized by multiple switches, while $\rho_3 \geq 1$ allow us to accurately estimate the class m_n. See Figure 97. If $\rho_3 \leq 0.1$, the term depending on $\delta_{n,2}$ in equation (7.86) might be weighted more than the error between the local and global estimate, thus causing the multiple switches in the mode estimates. It is worth remarking that the characteristics of the class estimates influence the resulting quality of $\{\hat{\theta}_\mu^g\}_{\mu=1}^2$, as it can be seen in Table 31.

The parameters left to be tuned are ρ_1 and ρ_2, which are associated to the conditions imposed on the values of the local estimates (see (7.88)) and the consensus constraint, respectively. By fixing $\mathscr{m}= 50$, $\Lambda = 0.995$, $\rho_3 = 1$



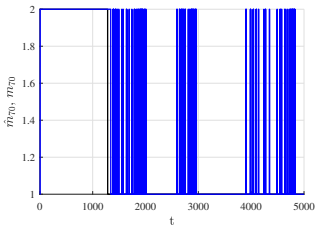
(a) $\Lambda = 0.975$ (b) $\Lambda = 0.995$

(c) $\Lambda = 1$

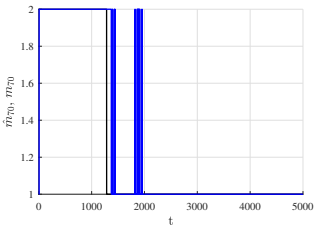
Figure 96: Class estimate vs Λ. True class m_{70} (black), estimated class $\hat{\mathsf{m}}_{70}$ (blue).

and $\rho_2 = 10^{-5}$, Figure 98 shows the global estimates obtained for different values of ρ_1. The estimates seems to be biased for $\rho_1 = 10^{-1}$. This result can be motivated considering that the choice of a relatively high ρ_1 leads to the consensus condition to be neglected. The choice of ρ_1 slightly influences the estimated class, while the value of ρ_2 strongly impact the quality of $\hat{m}_n$[2] as it can be deduced looking at the class reported in Figure 99 for different choices of ρ_2. A low-quality class estimate is obtained if $\rho_1 = 1$, while the number of switches of $\hat{m}_{70}$ tends to decrease as the value of ρ_2 is reduced. It can thus be deduced that the classes are not distinguishable if the consensus condition is enforced too strongly.
The global estimates retrieved with the selected parameters, $\mathscr{m} = 50$, $\Lambda = 0.995$, $\rho_1 = 10^{-2}$, $\rho_2 = 10^{-5}$ and $\rho_3 = 1$, are reported in Figure 100.

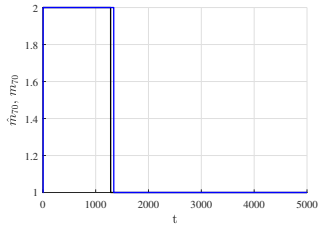
[2]The other parameters are set at $\mathscr{m} = 50$, $\Lambda = 0.995$, $\rho_3 = 1$, $\rho_1 = 10^{-2}$.



(a) $\rho_3 = 0.01$



(b) $\rho_3 = 0.1$

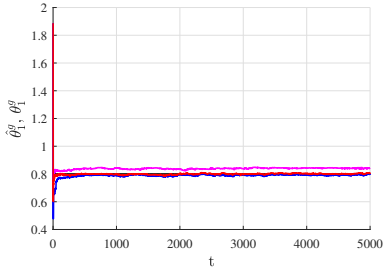


(c) $\rho_3 \geq 1$

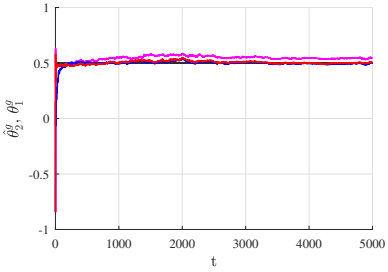
Figure 97: True vs estimated model class for $n = 70$. True m_{70} (black), estimate $\hat{\mathsf{m}}_{70}$ (blue).

Table 31: $\{\text{RMSE}^g\}_{\mu=1}^2$ vs ρ_3.

	ρ_3			
	0.01	**0.1**	**1**	**5**
RMSE_1^g	0.029	0.018	0.017	0.017
RMSE_2^g	0.039	0.027	0.024	0.024

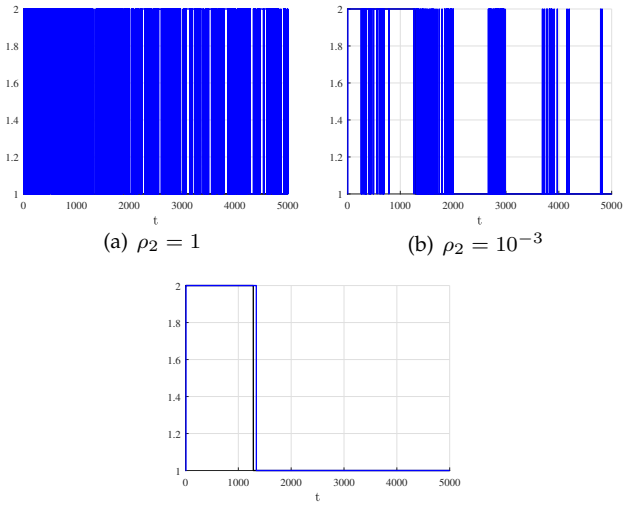


(a) $\hat{\theta}_1^g$ vs θ_1^g



(b) $\hat{\theta}_2^g$ vs θ_2^g

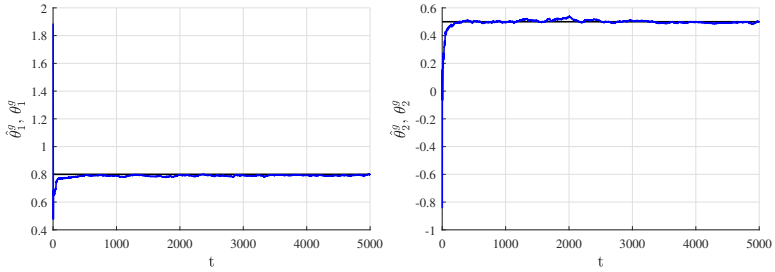
Figure 98: $\{\theta_\mu^g\}_{\mu=1}^2$ vs ρ_1. True parameter (black), estimated parameter with $\rho_1 = 10^{-1}$ (magenta), $\rho_1 = 10^{-5}$ (blue) and $\rho_1 = 10^{-2}$ (red).



(a) $\rho_2 = 1$ (b) $\rho_2 = 10^{-3}$

(c) $\rho_2 = 10^{-5}$

Figure 99: Class estimate vs ρ_2. True class m_{70} (black), estimated class $\hat{\mathsf{m}}_{70}$ (blue).



(a) $\hat{\theta}_1^g$ vs θ_1^g (b) $\hat{\theta}_2^g$ vs θ_2^g

Figure 100: True vs estimated global parameters. True (black), estimated (blue).

Algorithm 19 ADMM-RLS for multiple classes estimation

Input: Data flow $\{X_n(1), y_n(1)\}, \{X_n(2), y_n(2)\}, \ldots$, initial global estimates $\{\hat{\theta}^g_{\mu,\mathrm{o}}\}_{\mu=1}^{M}$, $\phi_n(0) \in \mathbb{R}^{n_\theta \times n_\theta}$, initial local estimates $\hat{\theta}_n(0)$, Lagrange multipliers $\delta^{\mathrm{o}}_{n,1}$ and $\delta_{n,2}(0)$ and auxiliary variables $z_{n,\mathrm{o}}$, $n = 1, \ldots, N$, $\{\lambda_n\}_{n=1}^{N}$, parameters $\rho_1, \rho_2, \rho_3 \in \mathbb{R}^+$.

1. **for** $t = 1, 2, \ldots$ **do**

   Local

   1.1. **for** $n = 1, \ldots, N$ **do**

   1.1.1. **compute** $\tilde{X}_n(t)$ as in (7.80);

   1.1.2. **compute** $K_n(t)$ and $\phi_n(t)$ with (7.78) - (7.79);

   1.1.3. **compute** $\hat{\theta}_n^{RLS}(t)$ with (7.83);

   1.2. **end for**;

   Global

   1.1. **do**

   1.1.1. **compute** $\hat{\theta}_n^{ADMM,(k+1)}(t)$ with (7.81), $n = 1, \ldots, N$;

   1.1.2. **compute** $\hat{\theta}_n^{(k+1)}(t)$ with (7.26), $n = 1, \ldots, N$;

   1.1.3. **compute** $\hat{z}_n^{(k+1)}$ with (7.66), $n = 1, \ldots, N$;

   1.1.4. **for** $\mu = 1, \ldots, M$ **do**

   1.1. **compute** $\hat{\theta}_\mu^{g,(k+1)}$ as in (7.74);

   1.1.5. **end for**;

   1.1.6. **compute** $\delta_{n,1}^{(k+1)}$ with (7.68), $n = 1, \ldots, N$;

   1.1.7. **compute** $\delta_{n,2}^{(k+1)}(t)$ as in (7.69), $n = 1, \ldots, N$;

   1.2. **until** a stopping criteria is satisfied (*e.g.*, the maximum number of iterations is attained);

   1.3. **predict** $\hat{\mathrm{m}}_n(t+1)$ **minimizing** (7.86), $n = 1, \ldots, N$, where $\rho_2 = \rho_3$.

2. **end**.

Output: Local estimates $\{\hat{\theta}_n^{RLS}(t)\}_{t=1}^{T}$, global refinement of the local estimates $\{\hat{\theta}_n(t)\}_{t=1}^{T}$, $n = 1, \ldots, N$, and global parameter's estimate $\{\hat{\theta}^g(t)\}_{t=1}^{T}$.

Chapter 8

Conclusions and Future Work

This thesis has addressed the problem of learning models from data considering two different settings, focusing on the identification of dynamical models. On the one hand, we present a set of algorithms that can be used for the identification of different classes of hybrid models, based on the consideration that some complex phenomena might not be accurately described using a single model. Two algorithms for energy disaggregation are then described to present a possible application of hybrid model to real-world problems. On the other hand, we introduce approaches for learning a single model given the observation gathered from multiple sources of data, that are supposed to share the same model.

In chapter 2 we present a novel two-stage method for PWA regression. The strengths of the proposed approach are: (i) its ability to run both in a batch and a recursive way, (ii) its computational efficiency and (iii) the fit that can be achieved. Indeed, the results of extensive simulations have shown that the methods proposed for the computation of linear multi-category separators outperform existing approaches in terms of CPU time, while generally guaranteeing a low number of misclassified points. However, the proposed discrimination methods only allow

one to find linear separators between the different clusters. Furthermore, the quality of the resulting PWA function mainly depends on the clusters obtained executing the first step of the method. Stage S1 is thus of crucial importance and it is also critical to the quality of the final PWA model. The proposed method relies on the hypothesis that the number of sub-models characterizing the PWA function is known a priori. Consequently, it does not allow to tune this parameter while the model is learned. We rely only on cross validation procedure to select the number of modes, thus requiring a batch of data to be processed off-line.
Through a proper reformulation of the original problems, it has been shown that the method for PWA regression can be successfully used for PWARX and LPV system identification. The computational efficiency of the approach makes it suited for applications with sampling time that can be down to the order of the milliseconds (*e.g.*, adaptive control applications, gain scheduling and LPV Model Predictive Control).

In chapter 3, we present a method for black-box identification of Discrete Hybrid Automata (DHA) extending the PWA regression method described in chapter 2. The presented DHA identification approach thus inherits the advantages of the PWA regression method and, in particular, its low computational complexity. The approach for Discrete Hybrid Automata identification does not require any prior knowledge on the true system, as we estimate models for both the continuous and discrete dynamics from data. However, it allows us only to model threshold events, while it cannot be used when the transitions between different operating conditions are regulated by time events.
The first stage of the approach is further extended to handle multiple discrete states sharing the same continuous model. However, this requires additional parameters to be tuned, whose choice heavily influences the accuracy of the estimated DHA.

In chapter 4 we describe methods for the identification of two specific jump models. At first, we propose a two-step approach to learn Rarely Jump Models (RJMs) from data, which is based on the assumption that

transitions between different modes seldom happen. Then, a two-stage method for the identification of Markov Jump Models (MJMs) is introduced. The method for RJM learning can be used to solve a broader range of problems (*e.g.*, binary classification), as confirmed by the results presented in the chapter. However, it should be run multiple times and it requires the solution of an optimization problem at each iteration, thus making it suited only for off-line learning. Instead, the approach for MJM identification is less general, as it is tailored for least-squares estimation, but it can be used for on-line learning as proven by the simulation results presented in chapter 4. The quality of the model obtained with the algorithm for Markov Jump Model learning strongly depends on the results of the first stage, thus making Stage S1 critical for the success of the whole identification procedure. On the other hand, due to the multiple refinements of both the model parameters and the state sequence, it might be easier to correct errors made on one of the two stages when the RJM learning method is used.

In chapter 5 we describe two methods for energy disaggregation, that allow one to estimate the power consumed by each appliance in an household relying only on the aggregate measures provided by a single smart meter. Differently from most NILM algorithms proposed in the literature, our approaches can be used for on-line disaggregation. The methods allow us to reconstruct the consumption patterns of each appliance an not only to detect ON/OFF events. Furthermore, the proposed energy disaggregation algorithms have proven to be robust to unmodelled appliances and they can be used when multiple appliances consume simultaneously, as it can be noticed from the results reported in chapter 5. However, both the approaches rely on the prior knowledge of a model for the consumption behavior of each appliance, thus requiring an intrusive period of training. Furthermore, no strategy to integrate disaggregation and learning of the appliances' models is proposed. We thus assume that the same consumption model always describe accurately the consumption behavior of the appliances, which might be not true in practice.

Chapter 6 and chapter 7 are devoted to the presentation of cloud-aided collaborative estimation methods that allow one to handle (i) linear consensus constraints, (ii) nonlinear consensus constraints and (iii) multi-class estimation problems. We introduce methods based on the Alternating Direction Method of Multipliers (ADMM) and relying on the local iteration of Recursive Least-Squares (RLS). The approaches can thus be easily integrated with pre-existing local least-squares estimators. However, the use of ADMM impose to tune the hyper-parameters characterizing the augmented Lagrangian. The choice of these parameters strongly influences the results of the estimation procedure and it is thus critical, as a wrong selection of these hyper-parameters negatively affects the quality of the final estimates. Furthermore, as the ADMM steps should be run multiple times, the use of the Alternating Direction Method of Multipliers leads to the introduction of two time scales, associated with the local and the cloud clock, respectively. This might be problematic when applications requiring fast computations are considered. As a preliminary solution to overcome this limitation, we have proposed methods relying on Node-to-Cloud (N2C) transmissions, thus reducing the exchange of information between the cloud and the local processors.

8.1 Future Work

Future research activities will aim at enriching the current approaches for hybrid models identification with techniques to estimate the number of sub-models in real-time, while the model is identified. Future investigations will also include a formal study on the performance of the multi-category discrimination algorithms with respect to the number of sub-models and the dimension of the regressor. In addition, we will study the performance of all the proposed methods for hybrid system identification in presence of correlated covariances and we will test the approaches on challenging real-world applications. Further investigations will be devoted to generalize the approach for PWA regression,

DHA identification and MJM identification to piecewise-nonlinear models (such as piecewise polynomial).
Future research on disaggregation will be focused on the reduction of the intrusiveness of the methods. As a consequence, further efforts will be devoted to the integration of learning techniques to the presented disaggregation approaches, so to update the appliances models while energy disaggregation is performed. To this extent, Variable Structure Multiple Models will be considered as they might help representing possible changes in the consumption behavior of the devices over time. We will also investigate how the use of additional information affects the quality of the disaggregated patterns, with the ultimate goal of proposing a completely automatic technique for energy disaggregation. We will then focus on the application of the approaches to fields other than power disaggregation.
By using the presented ADMM-based approach for cloud-aided collaborative estimation as a starting point, future investigations will be devoted to design alternative methods for collaborative estimation, *e.g.*, approaches designed considering the problem from a probabilistic point of view. Concerning the proposed methods, future research will also be devoted to investigate methods for the automatic tuning of the hyperparameter characterizing the augmented Lagrangian. Furthermore, efforts will be directed to the reduction of the transmission complexity, the design of asynchronous solution based on the proposed approaches and to the extension of the methods to address closed-loop estimation.
Finally, we will investigate the problem of learning multiple models from multiple sources, to combine switching model regression and cloud-based model estimation.

References

[1] A. Alessandri, M. Cuneo, S. Pagnan, and M. Sanguineti. A recursive algorithm for nonlinear least-squares problems. *Computational Optimization and Applications*, 38(2):195–216, Nov 2007. 174

[2] S.T. Alexander and A.L. Ghirnikar. A method for recursive least squares filtering based upon an inverse QR decomposition. *IEEE Trans. Signal Processing*, 41(1):20–30, 1993. 18, 21, 62, 63, 84

[3] P. Apkarian and P. Gahinet. A convex characterization of gain-scheduled $\mathcal{H}_\infty$ controllers. *IEEE Transactions on Automatic Control*, 40(5):853–864, 1995. 42

[4] R. Arablouei, K. Doğançay, S. Werner, and Y. F. Huang. Adaptive distributed estimation based on recursive least-squares and partial diffusion. *IEEE Transactions on Signal Processing*, 62(14):3510–3522, July 2014. 12

[5] L. Bako, K. Boukharouba, E. Duviella, and S. Lecoeuche. A recursive identification algorithm for switched linear/affine models. *Nonlinear Analysis: Hybrid Systems*, 5(2):242–253, 2011. 3, 4, 18, 22

[6] B. A. Bamieh and L. Giarré. Identification of linear parameter-varying models. *International Journal of Robust Nonlinear Control*, 12(9):841–853, 2002. 5, 43, 51, 52

[7] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and X.-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2002. 10

[8] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava. NILMTK: an open source toolkit for non-intrusive load monitoring. In *Proceedings of the 5th international conference on Future energy systems*, pages 265–276. ACM, 2014. 126

[9] A. Bemporad. Hybrid Toolbox - User's Guide, 2004. http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox. 50

[10] A. Bemporad, D. Bernardini, and P. Patrinos. A convex feasibility approach to anytime model predictive control. Technical report, IMT Lucca, February 2015. http://arxiv.org/abs/1502.07974. 25

[11] A. Bemporad, V. Breschi, and D. Piga. Piecewise affine regression via recursive multiple least squares and multicategory discrimination. Technical report, IMT Institute for Advanced Studies, Lucca, October 2015. Available at: http://www.dariopiga.com/TR/TR_PWAReg_BBP_2015.pdf. 5

[12] A. Bemporad, V. Breschi, D. Piga, and S. Boyd. Fitting jump models. 2017. https://arxiv.org/abs/1711.09220. 9, 111, 112

[13] A. Bemporad and S. Di Cairano. Model-predictive control of discrete hybrid stochastic automata. *IEEE Transactions on Automatic Control*, 56(6):1307–1321, June 2011. 6

[14] A. Bemporad, A. Garulli, S. Paoletti, and A. Vicino. A bounded-error approach to piecewise affine system identification. *IEEE Transactions on Automatic Control*, 50(10):1567–1580, 2005. 3, 4

[15] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999. 1, 55

[16] A. Bemporad, M. Morari, and N.L. Ricker. *Model Predictive Control Toolbox for MATLAB 5.0*. The Mathworks, Inc., 2015. http://www.mathworks.com/help/mpc/ug/adaptive-mpc.html. 42

[17] K.P. Bennett and O.L. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, 3:27–39, 1994. 23, 24, 33, 34, 37, 38, 39, 46, 47, 52, 53, 54

[18] M. Benning, F. Knoll, C. Schönlieb, and T. Valkonen. Preconditioned ADMM with nonlinear operator constraint, 2015. https://arxiv.org/abs/1511.00425. 172, 173, 174, 180

[19] D.P. Bertsekas. Incremental least squares methods and the extended kalman filter. *SIAM Journal on Optimization*, 6(3):807–822, 1996. 1, 174

[20] F. Boem, R. M.G. Ferrari, C. Keliris, T. Parisini, and M.M. Polycarpou. A distributed networked approach for fault detection of large-scale systems. *IEEE Transactions on Automatic Control*, 62(1):18–33, Jan 2017. 11

[21] F. Boem, Y. Xu, C. Fischione, and T. Parisini. A distributed estimation method for sensor networks based on pareto optimization. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 775–781, Dec 2012. 12

[22] L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012. 28

[23] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122, January 2011. 12, 136, 137, 173

[24] V. Breschi, A. Bemporad, and D. Piga. Identification of hybrid and linear parameter varying models via recursive piecewise affine regression and discrimination. In *2016 European Control Conference, ECC 2016, Aalborg, Denmark, June 29 - July 1, 2016*, pages 2632–2637, 2016. 5, 7, 8

[25] V. Breschi, D.Piga, and A.Bemporad. Piecewise affine regression via recursive multiple least squares and multicategory discrimination. *Automatica*, 73:155–162, 2016. 5, 7, 8, 83

[26] V. Breschi, I. Kolmanovsky, and A. Bemporad. Cloud-aided collaborative estimation by ADMM-RLS algorithms for connected vehicle prognostics. Technical report, 2017. Available at: https://arxiv.org/abs/1709.07972. 14

[27] V. Breschi, I. Kolmanovsky, and A. Bemporad. Cloud-aided collaborative estimation by ADMM-RLS algorithms for connected vehicle prognostics, 2017. Submitted to *American Control Conference 2018*. 14

[28] V. Breschi, D. Piga, and A. Bemporad. Learning hybrid models with logical and continuous dynamics via multiclass linear separation. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 353–358, Dec 2016. 7, 83

[29] V. Breschi, D. Piga, and A. Bemporad. Jump model learning and filtering for energy-use disaggregation. In *Submitted for publication*, 2018. 11

[30] V. Breschi, D. Piga, S. Boyd, and A. Bemporad. Matlab codes for jump models learning. https://github.com/ValeB8/JumpModels_tlbx, May 2017. 88

[31] M. P. Cabasino, P. Darondeau, M. P. Fanti, and C. Seatzu. *Model Identification and Synthesis of Discrete-Event Systems*. John Wiley and Sons, Inc., 2015. 6

[32] T. R. Camier, S. Giroux, B. Bouchard, and A. Bouzouane. Designing a nialm in smart homes for cognitive assistance. *Procedia Computer Science*, 19(Supplement C):524 – 532, 2013. The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013), the 3rd International Conference on Sustainable Energy Information Technology (SEIT-2013). 10

[33] F. S. Cattivelli, C. G. Lopes, and A. H. Sayed. Diffusion recursive least-squares for distributed estimation over adaptive networks. *IEEE Transactions on Signal Processing*, 56(5):1865–1877, May 2008. 12

[34] F. S. Cattivelli and A. H. Sayed. Diffusion LMS strategies for distributed estimation. *IEEE Transactions on Signal Processing*, 58(3):1035–1048, March 2010. 12

[35] T. H. Chang, M. Hong, and X. Wang. Multi-agent distributed optimization via inexact consensus ADMM. *IEEE Transactions on Signal Processing*, 63(2):482–497, Jan 2015. 12

[36] D. Chen, L. Bako, and S. Lecœuche. A recursive sparse learning method: Application to jump markov linear systems. *IFAC Proceedings Volumes*, 44(1):3198 – 3203, 2011. 18th IFAC World Congres. 7, 83

[37] J. Chen and R. J. Patton. *Robust Model-based Fault Diagnosis for Dynamic Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1999. 1

[38] E. Cinquemani, R. Porreca, G. Ferrari-Trecate, and J. Lygeros. A general framework for the identification of jump Markov linear systems. In *IEEE Conference on Decision and Control*, dec 2007. 7

[39] A. Cominola, M. Giuliani, D. Piga, A. Castelletti, and A.E. Rizzoli. A hybrid signature-based iterative disaggregation algorithm for non-intrusive load monitoring. *Applied Energy*, 185(P1):331–344, 2017. 10

[40] O. L. V. Costa, M. D. Fragoso, and R. P. Marques. *Discrete-time Markov jump linear systems*. Springer Science & Business Media, 2006. 6, 79

[41] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1(2):224–227, February 1979. 66

[42] Donnelly K. A. Ehrhardt-Martinez K. and Laitner J.A. Advanced metering initiatives and residential feedback programs: A meta-review for household electricity-saving opportunities, 2010. Technical report. E105 American Council for an Energy-Efficient Economy (ACEE) Washington, DC. 9

[43] G. Ferrari-Trecate. Hybrid Identification Toolbox (HIT), 2005. 31, 32, 36

[44] G. Ferrari-Trecate, D. Mignone, and M. Morari. Moving horizon estimation for hybrid systems. *IEEE Transactions on Automatic Control*, 47(10):1663–1676, Oct 2002. 1

[45] G. Ferrari-Trecate, M. Muselli, D. Liberati, and M. Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, 2003. xii, 3, 4, 32, 36, 37

[46] M. Figueiredo, B. Ribeiro, and A.M de Almeida. *On the Regularization Parameter Selection for Sparse Code Learning in Electrical Source Separation*, volume 7824. Berlin, Germany: Springer-Verlag, 2013. 10

[47] P. A. Forero, A. Cano, and G. B. Giannakis. Consensus-based distributed support vector machines. *The Journal of Machine Learning Research*, 11:1663–1707, Aug 2010. 11

[48] G.M. Fung and O.L. Mangasarian. Multicategory proximal support vector machine classifiers. *Machine Learning*, 59:77–97, 2005. xii, 36, 37

[49] C. E. García, D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335 – 348, 1989. 1

[50] F. Garin and L. Schenato. *A Survey on Distributed Estimation and Control Applications Using Linear Consensus Algorithms*, pages 75–107. Springer London, London, 2010. 11

[51] A. Garulli, S. Paoletti, and A. Vicino. A survey on switched and piecewise affine system identification. In *System Identification*, volume 16, pages 344–355, 2012. 3

[52] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. 90

[53] M. Grant and Boyd S. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014. 90

[54] G. W. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, Dec 1992. 9

[55] J.A. Hartigan and M.A. Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979. 65, 66

[56] W.P.M.H. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085 – 1091, 2001. 55

[57] T. A. Henzinger. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*, pages 265–292. Springer Berlin Heidelberg, 2000. 6

[58] M.N. Howell, J.P. Whaite, P. Amatyakul, Y.K. Chin, M.A. Salman, C.H. Yen, and M.T. Riefe. Brake pad prognosis system, Apr 2010. US Patent 7,694,555. 13

[59] The MathWorks Inc. Statistics and machine learning toolbox, 2015. url: https://www.mathworks.com/help/stats/index.html. 91

[60] R. Isermann and P. Ballé. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control Engineering Practice*, 5(5):709 – 719, 1997. 1

[61] M. J. Johnson and A. S. Willsky. Bayesian nonparametric hidden semi-markov models. *J. Mach. Learn. Res.*, 14(1):673–701, February 2013. 10

[62] A. L. Juloski, S. Weiland, and W. P. M. H. Heemels. A bayesian approach to identification of hybrid systems. *IEEE Transactions on Automatic Control*, 50(10):1520–1533, 2005. 3, 4

[63] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 1960. 1

[64] E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. *Data mining in time series databases*, 57:1–22, 2004. 6

[65] J. Z. Kolter and T. Jaakkola. Approximate inference in additive factorial hmms with application to energy disaggregation. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22, pages 1472–1482, 2012. 10

[66] H. Kwakernaak. *Linear Optimal Control Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1972. 1

[67] F. Lauer and Y. Guermeur. MSVMpack: a multi-class support vector machine package. *Journal of Machine Learning Research*, 12:2269–2272, 2011. `http://www.loria.fr/~lauer/MSVMpack`. 33

[68] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines, theory, and application to the classification of microarray data and satellite radiance data. *J. American Statistical Association*, 99:67–81, 2004. 33, 34, 37, 39

[69] Z. Li, I. Kolmanovsky, E. Atkins, J. Lu, D. P. Filev, and J. Michelini. Road risk modeling and cloud-aided safety-based route planning. *IEEE Transactions on Cybernetics*, 46(11):2473–2483, Nov 2016. 13

[70] Z. Li, I. Kolmanovsky, E. M. Atkins, J. Lu, D. P. Filev, and Y. Bai. Road disturbance estimation and cloud-aided comfort-based route planning. *IEEE Transactions on Cybernetics*, PP(99):1–13, 2017. 13

[71] M. Lichman. UCI machine learning repository, 2013. 40

[72] L. Ljung. *System identification: theory for the user*. Prentice-Hall Englewood Cliffs, NJ, 1999. xvii, 112, 142, 147, 168, 170, 187, 190

[73] C. G. Lopes and A. H. Sayed. Incremental adaptive strategies over distributed networks. *IEEE Transactions on Signal Processing*, 55(8):4064–4077, Aug 2007. 12

[74] S. Makonin, B. Ellert, I.V. Bajic, and F. Popowich. Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014. *Scientific Data*, 3(160037):1–12, 2016. 94, 109, 125

[75] G. Mateos, I. D. Schizas, and G. B. Giannakis. Distributed recursive least-squares for consensus-based in-network adaptive estimation. *IEEE Transactions on Signal Processing*, 57(11):4583–4588, Nov 2009. 12, 14, 140

[76] P. M. Mell and T. Grance. Sp 800-145. the NIST definition of cloud computing. Technical report, Gaithersburg, MD, United States, 2011. 12

[77] J.F. C. Mota, J. M. F. Xavier, P. M. Q. Aguiar, and M. Puschel. D-ADMM: A communication-efficient distributed algorithm for separable optimization. *Trans. Sig. Proc.*, 61(10):2718–2723, May 2013. 12

[78] H. Nakada, K. Takaba, and T. Katayama. Identification of piecewise affine systems based on statistical clustering technique. *Automatica*, 41(5):905–913, 2005. 3, 4

[79] S. Narasimhan and G. Biswas. Model-based diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(3):348–361, May 2007. 1

[80] B. North, A. Blake, M. Isard, and J. Rittscher. Learning and classification of complex dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):1016–1034, Sep 2000. 6, 9

[81] K. Noto and M. Craven. Learning hidden markov models for regression using path aggregation. *CoRR*, abs/1206.3275, 2012. 6

[82] H. Ohlsson, L. Ljung, and S. Boyd. Segmentation of ARX-models using sum-of-norms regularization. *Automatica*, 46(6):1107–1111, 2010. 8

[83] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *2007 46th IEEE Conference on Decision and Control*, pages 5492–5498, Dec 2007. 11

[84] E. Ozatay, S. Onori, J. Wollaeger, U. Ozguner, G. Rizzoni, D. Filev, J. Michelini, and S. Di Cairano. Cloud-based velocity profile optimization for everyday driving: A dynamic-programming-based solution. *IEEE Transactions on Intelligent Transportation Systems*, 15(6):2491–2505, Dec 2014. 13

[85] A. Packard. Gain scheduling via linear fractional transformations. *Systems & Control Letters*, 22(2):79–92, 1994. 42

[86] S. Paoletti, A. L. Juloski, G. Ferrari-Trecate, and R. Vidal. Identification of hybrid systems a tutorial. *European journal of control*, 13(2):242–260, 2007. 3

[87] O. Parson, S. Ghosh, M. Weal, and A. Rogers. An unsupervised training method for non-intrusive appliance load monitoring. *Artificial Intelligence*, 217(Supplement C):1 – 19, 2014. 10

[88] D. Piga, A. Cominola, M. Giuliani, A. Castelletti, and A. E. Rizzoli. Sparse optimization for automated energy end use disaggregation. *IEEE Transactions on Control Systems Technology*, 24(3):1044–1051, 2016. 10

[89] D. Piga, P. Cox, R. Tóth, and V. Laurain. LPV system identification under noise corrupted scheduling and output signal observations. *Automatica*, 53:329 – 338, 2015. 5

[90] G. Pola, M. L. Bujorianu, J. Lygeros, and M. D. Di Benedetto. Stochastic hybrid models: An overview. In *In Proceedings IFAC Conference on Analysis and Design of Hybrid Systems*, 2003. 6

[91] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007. 113

[92] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989. 6, 9, 88

[93] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSp Magazine*, 1986. 88

[94] R. Rajamani. *Vehicle dynamics and control*. Springer Science, 2 edition, 2012. 183

[95] S. S. Ram, A. Nedić, and V. V. Veeravalli. Stochastic incremental gradient descent for estimation in sensor networks. In *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, pages 582–586, Nov 2007. 12

[96] J.B. Rawlings. *Moving Horizon Estimation*, pages 1–7. Springer London, London, 2013. 1

[97] J. Roll, A. Bemporad, and L. Ljung. Identification of piecewise affine systems via mixed-integer programming. *Automatica*, 40(1):37–50, 2004. 3

[98] D. Ruppert. Efficient estimations from a slowly convergent Robbins-Monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988. 28

[99] A. G. Ruzzelli, C. Nicolas, A. Schoofs, and G. M. P. O'Hare. Real-time recognition and profiling of appliances through a single electricity sensor. In *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 1–9, June 2010. 10

[100] A. H. Sayed and C. G. Lopes. Distributed recursive least-squares strategies over adaptive networks. In *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, pages 233–237, Oct 2006. 12

[101] C. W. Scherer. Mixed $\mathcal{H}_2/\mathcal{H}_\infty$ control for time-varying and linear parametrically-varying systems. *Int. Journal of Robust and Nonlinear Control*, 6(9-10):929–952, 1996. 42

[102] I. D. Schizas, G. Mateos, and G. B. Giannakis. Consensus-based distributed recursive least-squares estimation using ad hoc wireless sensor networks. In *2007 Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers*, pages 386–390, Nov 2007. 12

[103] I. D. Schizas, G. Mateos, and G. B. Giannakis. Distributed LMS for consensus-based in-network adaptive processing. *IEEE Transactions on Signal Processing*, 57(6):2365–2382, June 2009. 12

[104] E. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on Automatic Control*, 26(2):346–358, 1981. 55

[105] D. Srinivasan, W. S. Ng, and A. C. Liew. Neural-network-based signature recognition for harmonic source identification. *IEEE Transactions on Power Delivery*, 21(1):398–405, Jan 2006. 10

[106] K. Suzuki, S. Inagaki, T. Suzuki, H. Nakamura, and K. Ito. Nonintrusive appliance load monitoring based on integer programming. In *2008 SICE Annual Conference*, pages 2742–2747, 2008. 10

[107] A. Svensson, T. B. Schön, and F. Lindsten. Identification of jump markov linear models using particle filters. In *53rd IEEE Conference on Decision and Control*, pages 6504–6509, Dec 2014. 6, 9

[108] E. Taheri, O. Gusikhin, and I. Kolmanovsky. Failure prognostics for in-tank fuel pumps of the returnless fuel systems. In *Dynamic Systems and Control Conference*, Oct 2016. 13

[109] R. Tibshirani, M. Saunders, S. Rosset, J. Zhu, and K. Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005. 6, 8

[110] F.D. Torrisi and A. Bemporad. HYSDEL-a tool for generating computational hybrid models for analysis and synthesis problems. *IEEE Transactions on Control Systems Technology*, 12(2):235–249, 2004. 6, 56, 58

[111] R. Tóth. *Modeling and identification of linear parameter-varying systems*. Lecture Notes in Control and Information Sciences, Vol. 403, Springer, Heidelberg, 2010. 43

[112] V. Vapnik. *Statistical learning theory*. Wiley New York, 1998. 18, 32

[113] E. Wei and A. Ozdaglar. Distributed alternating direction method of multipliers. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 5445–5450, Dec 2012. 12

[114] M.A. Woodbury. *Inverting Modified Matrices*. Statistical Research Group Memorandum Reports, 42. Princeton University, Princeton, NJ, 1950. 142, 143, 151, 158, 177

[115] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490*, 2011. 28

[116] I. C. Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. 40

[117] M. Zeifman and K. Roth. Nonintrusive appliance load monitoring: Review and outlook. *IEEE Transactions on Consumer Electronics*, 57(1):76–84, 2011. 10

[118] R. Zhang and J.T. Kwok. Asynchronous distributed ADMM for consensus optimization. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1701–II–1709. JMLR.org, 2014. 12

[119] A. Zoha, A. Gluhak, M. A. Imran, and S. Rajasegarar. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors*, 12(12):16838–16866, 2012. 10