Research Internship (PRE)

Field of Study: SIM
Scholar Year: 2015-2016

Distributed learning for high-dimensional wind energy production forecasting

Confidentiality Notice
Non-confidential report and publishable on Internet

Author:
David OBST

Promotion:
2017

ENSTA ParisTech Tutor:
Hasnaa Zidani

Host Organism Tutor:
Pierre Pinson

Internship from May 9, 2016 to August 12, 2016

Name of the host organism: Technical University of Denmark
Address: DTU, Building 325
Elektrovej
2800 Kgs. Lyngby, Denmark

Confidentiality Notice

This present document is not confidential. It can be communicated outside in paper format or distributed in electronic format.

Abstract

With their cheap maintenance and operating costs, renewable energy sources are a great opportunity to increase profits in electricity markets. However they are also subject to very high variability, and as such a bad prediction of their expected production can cause high losses too. That is why recently interest for very-short term prediction (between 5 and 30 minutes-ahead) of solar and wind energy production has soared. One of the most successful ideas for the latter has been to take advantage of spatial dependencies between wind power plants. This has been performed in [1] with success, but needed owners of wind farms to share their data which may not always be possible due to competition in electricity markets. The matter of privacy was addressed in both [2] and [3] with the use of the Alternative Direction Method of Multipliers (ADMM), which allowed to distribute a master optimization problem among actors of a smart grid, thus avoiding the sharing of data. Despite satisfying results, the methods from the aforementioned papers required the hypothesis of stationarity of the electricity production (which is rarely true for wind energy) and were expensive in computations. Furthermore they involved numerous and large exchanges during the learning process, which makes their implementation in real life difficult. This paper tackles these issues through the development of an online distributed learning method which still relies on the ADMM as basis. Protection of information, being one of the major constrains in this problem, is achieved through the introduction of encryption matrices. Finally the developed online ADMM algorithm is applied on a data set of power measurements of 349 Danish wind farms and its performance compared with state-of-the-art methods.

Index terms - Very-short term forecasting, Distributed optimization, Online learning, Alternative Direction Method of Multipliers, Encrypted learning, Sparse estimation.

Acknowledgment

I would like to thank my tutor Pierre Pinson who invited me to take part in this exciting project, and who gave me advice and guided me during my whole internship. I would also like to thank Yannig Goude who put me in contact with sir Pinson and without whom I would probably never had taken interest in statistics.
I also thank the Université Paris-Saclay who gracefully accorded me a scholarship, as well as Yannick Pérez who wrote a support letter to help me to apply for it. Many thanks to Hasnaa Zidani too who has accepted to be my referent teacher for this project.

Finally I would like to thank my friends and family who keeps supporting me no matter what, with a special mention to Niklas Vespermann whom I met at DTU and who has become a very good friend.

Contents

List of Figures

Introduction

In order to increase profits in real-time electricity markets or to reduce operational costs for a system operator, predicting wind power production with increased accuracy is of major importance. One of the best ideas which has been found to improve predictions was to use neighbouring wind farms to improve one's own forecasts. However until now, existing methods either need to openly share data, which cannot be done in general due to confidentiality issues, or need to suppose the stationarity of wind energy production, which is a very limiting hypothesis and which may cause inaccurate forecasts in case of changing weather conditions.

That is why the goal of this project in which I was involved with Pierre Pinson at the Elektro laboratory of DTU was to develop an online learning algorithm through the Alternative Direction Method of Multipliers. We aimed at developing a method which would be cheap in calculations, yield good results, and which would be easy to implement in real life. The long term objective would be the development of a platform for wind power plants agents of an electricity market, where they could safely help each other to improve their forecasts without worrying about the privacy of their data. The research project was rather independent from other projects going on at the laboratory. It was furthermore a rather open project since with sir Pinson we did not know if it would be concluded with success, failure, or with open possibilities. The reader will see that the project, though not completely over, has overall been a success, even if not every expectations have been met and that areas of improvement exist.

Therefore the first part of this report will present the concrete motivations behind of our work, the model we chose and the reasons explaining the choice of the Alternative Direction Method of Multipliers to perform distributed learning. The second part will focus on the newly developed algorithm, called online ADMM (OADMM). It will present the calculations, heavily emphasizing on the privacy of data, and will then present results obtained when applied on synthetic data and on real power data. Finally possibilities of development of OADMM will be discussed.

Nomenclature

In this paper, two general conventions for notations are adopted. The first is that vectors are written in bold, matrices in the blackboard bold style and scalars with regular font. Another convention is that temporal or loop indexation is made with indices (for example $\boldsymbol{\beta}_t$ or $\boldsymbol{\beta}_k$) whereas spatial indexation (usually denoting a given agent i) is made with exponents (*e.g.* $\boldsymbol{\beta}^i$).

Symbol	Description
$\lvert\cdot\rvert$	Either the ℓ_1 norm or the absolute value
$\lVert\cdot\rVert$	The euclidian norm
Ω_p	Portefolio of the central agent
Ω_a	Set of contracted wind farms
m_p	Number of wind farms of Ω_p
m_a	Number of wind farms of Ω_a
m	Total number of wind farms of the grid, *i.e.* $m = m_p + m_a$
$\mathbb{X}$	Design matrix
$\mathbb{X}_i$	Submatrix of $\mathbb{X}$ corresponding to agent i's data
$\boldsymbol{x}_t$	Line t of design matrix $\mathbb{X}$. Alternatively vector of data at instant t
$x_{j,t}$	Power production of wind farm j at moment t
$\boldsymbol{Y}$	Response vector corresponding to design matrix $\mathbb{X}$
y_t	Response data at instant t. It can thus also be the t-ieth coefficient of $\boldsymbol{Y}$
ℓ	Maximum lag of an autoregressive model
$\boldsymbol{\beta}$	Vector of coefficients of an autoregressive model
$\hat{\boldsymbol{\beta}}$	Estimation of $\boldsymbol{\beta}$
λ	Shrinkage parameter of the lasso penalization
ρ	Parameter of the augmented lagragian $\mathcal{L}_\rho$
μ, μ_t	Respectively a forgetting factor and dynamic forgetting factor
$\mathbb{M}_i$	Encryption matrix of agent i
$\mathbb{K}_i$	Secondary (orthogonal) encryption matrix of agent i

Part I

Organization of the distributed learning problem and analysis of state-of-the-art methods

Before starting the core of this project, which will be the subject of part II, the first objective of this research project was to understand the context in which our work is done and comprehend in detail the latest methods for very-short term wind energy production forecasting. Being aware of their strengths and weaknesses is mandatory to understand why new methods need to be developed and why the decision to use the use of the Alternative Direction Method of Multipliers was taken.
That is why after a presentation on electricity markets and brief discussion on the latest very-short term forecasting methods in the field of wind power production, we will define the model which has been used for wind energy production and present the main calculations of ADMM used on a stationary data set. Results and limits will be discussed, emphasizing the need of innovating even further the distributed learning approach.

I.1 Context of the work and state-of-the-art

Renewable energy sources occupy a particular role in electricity markets and for system operators. They are the cheapest way to produce electricity, and therefore represent the energy sources with the greatest potential of profits. However as everybody knows they are subject to high variability. Accurate forecasting of wind production for instance is therefore a priority to be able to adapt energy production coming from traditional sources such as coal or nuclear power plants and thus reducing expenditures. Different methods have been developed in the field of very-short term forecasting (between 5 and 30 minutes-ahead), but have different properties or hypotheses on which they are based on.

I.1.1 Improving forecasts to increase profits in electricity markets and for system operators

More and more electricity markets work in real time. In those real time markets, consumers usually declare and pay their expected consumption shortly ahead (typically 15 or 30 minutes before actual delivery of electricity), and producers answer by what they will be able to produce to satisfy the demand. While traditional energy producers such as coal or nuclear power plants barely make mistakes on their expected production, the same cannot be said for stochastic producers such as wind and solar power plants whose production will either be above or below what they declared ahead. This leads to two possible scenarii:

- The total production is above the demand. Since electricity still can't be stored in a reliable way, regular producers must rebuy the excess of electricity. Obviously they do so when the rebuy price is cheaper than their selling price. Stochastic producers on their side will sell their whole production, however the excess will be sold at an inferior price than the market price. Inferior production to what was declared on the opposite has no negative consequences.
- The total production is below the demand. This is a more problematic case, because it forces traditional energy producers to quickly adapt their production, which rises the cost of energy. If a stochastic source produces more than what it had announced, it helps to meet the demand and as such has no penalty. However if what it produces is inferior to what it had announced, it must buy the difference, which has a very high cost.

Therefore an improvement in very-short term wind or solar energy production forecasting would minimize the losses in both cases, this way increasing the earnings of participants of the electricity markets. For the system operator, a better prediction of stochastic electricity production would allow him to be more sure when he can reduce the activity of coal or nuclear power plants. This way he would reduce his maintenance and operational costs, increasing his profit too.

To meet the demand of more accurate very-short term predictions, new mathematical methods have been developed, which is the subject of the next part.

I.1.2 State-of-the-art

The number of wind power plants has been soaring the past years and there is no reason that this trend will change in the next decade. One may therefore be interested in taking advantage of spatial dependencies between wind power plants (WPP), something which had been rarely performed until recently. This is the goal of the methods which are presented hereafter.

Vector Auto-Regression (VAR) and sparse Vector Auto-Regression (sVAR)

First of all, these methods make the hypothesis that contracts have been signed to share data between participants of a grid of wind power plants. As such there is no confidentiality in their produced amount of electricity.

As one could easily imagine, close wind farms are exposed to similar meteorological conditions, such as wind speed and direction for instance. Therefore they may share similarities in their energy production at a given moment. Vector Auto-Regression (VAR) and Sparse Vector Auto-Regression aim at taking advantage of those spatial correlation between wind farms. This can be achieved by reformulating a regular AR model under a vectorial form where the scalar coefficients of autoregression are replaced by matrices $\mathbb{B}_\tau$ which account for the interdependencies between sites. If one puts the electricity production at step t of the different wind power plants which have signed a contract in a vector $\boldsymbol{Y}_t$, this yields the model:

$$\boldsymbol{Y}_t = \sum_{\tau=1}^{\ell} \mathbb{B}_\tau \, \boldsymbol{Y}_{t-\tau} + \boldsymbol{\varepsilon}_t \tag{I.1}$$

where $\boldsymbol{\varepsilon}_t$ is a noise. While VAR already yields improvement of very-short term forecasts of a few percent, often large sets of wind farms are considered and give matrices $\mathbb{B}_\tau$ of at least size 60×60. This leads to high computational costs. Moreover those matrices $\mathbb{B}_\tau$ calculated this way lack sparsity. This is an issue since usually only neighbouring farms hold interesting data for eachother and that too

many coefficients may even decrease the quality of the estimation. That is why sparse VAR has been introduced to address this issue. This has for example been performed in [1].
Sparse Vector Auto-Regression relies on the Bayesian Information Criterion and on Partial Spectral Coherence in order to select a very small number of non-zero coefficients, which correspond to the highest correlation between certain wind farms in terms of electricity production. Furthermore this method can be combined with probabilistic forecasts, which makes it more powerful than regular deterministic forecasts and allows to easily calculate confidence intervals.

Both sVAR and VAR have proved to achieve good performance when compared to regular AR. Indeed after being applied to a set of 22 Australian wind farms, they have both improved prediction accuracy by at least 1 % in terms of Root Mean Squared Error (abridged RMSE) and Mean Absolute Error (abridged MAE, see II.15 for a definition of both criteria).

However notwithstanding those great results, as mentioned at the beginning VAR and sVAR need operators of wind farms to sign a contract for sharing of data. However due to competition in electricity markets and a desire of confidentiality, such contracts are very unlikely to be passed. That is why methods needed to be found in order to guarantee secrecy of the data of different agents, while improving forecasts quality by a similar amount than those methods.

sVAR through the Alternative Direction Method of Multipliers

Researchers were keen on keeping the idea of Sparse Vector Auto-Regression for improving wind electricity production forecasts since it yields good results and allows to reduce calculations through sparsity of matrices. However a method had to be found to avoid sharing of data. In order to do this, the Alternative Direction Method of Multipliers has been chosen (the method will be presented in detail in I.2.2). It may be used to split an optimization problem, which in our case means that it can be distributed among the agents of the smart grid, effectively solving the matter of privacy. This has been performed in the upcoming paper [2]. A VAR formulation identical to I.1 has been chosen for the electricity production of all the wind power plants at step t. If one bases his estimation on data available from time steps 1 to T, one can reformulate the problem under a matricial form as following. All the vectors $\boldsymbol{Y}_t$ (containing the electricity production of the different wind farms at step t) are put together in a matrix $\boldsymbol{Y}$, while all the coefficient matrices $\mathbb{B}_\tau$ and all the past information vectors $\boldsymbol{Y}_{t-p}$ are respectively binded together into a matrix $\mathbb{B}$ and a design matrix $\mathbb{X}$. This yields the matricial sVAR formulation:

$$\mathbb{Y} = \mathbb{B}\mathbb{X} + \varepsilon \tag{I.2}$$

The goal is thus to estimate the coefficient matrix $\mathbb{B}$, which is then applied on newly arriving data to obtain forecasts. The method this time doesn't rely on Bayesian Information Criterion or Partial Spectral Coherence as in [1]. This time a least-squares problem will be solved to calculate $\mathbb{B}$, and sparsity will be ensured by adding a ℓ_1 penalty (called *lasso* penalty) to the least-square problem, *i.e.*

$$\min_{\mathbb{B}} \|\mathbb{Y} - \mathbb{B}\mathbb{X}\|_{\mathsf{F}} + \lambda|\mathbb{B}| \tag{I.3}$$

where $\|\cdot\|_{\mathsf{F}}$ is the squared Frobenius norm and $\lambda > 0$ is a shrinkage parameter (more detail on the latter will be given in I.2.1).

ADMM allows this optimality problem to be distributed among agents, who calculate their share in the matrix $\mathbb{B}$ and then send it back to the one centralizing the optimization problem. This way

the information is never shared directly, and still produces a sparse VAR structure as in the estimation method presented above. Sparse VAR estimations obtained through the method presented in this section generally improve forecasts from the method presented in the previous section by between 1% and 2%. However this time the sparsity of the estimated matrix $\mathbb{B}$ is only around 40 %, whereas in the method presented above it was around 90 %. This can thus increase the cost of calculation and also the financial cost of performing ADMM, since every non-zero coefficient in the estimated matrix $\mathbb{B}$ may be needed to be bought.

However seeing the efficiency of ADMM to distribute a learning problem among agents, it has been decided to investigate further on which use one can do of it. This leads to the main topic of this paper which uses ADMM to a full extend.

I.2 Mathematical modelization of the problem

The mathematical will be almost identical to the one used in [3] and will consist in an autoregressive model with exogenous input, which means that surrounding wind farms also present in the electricity market may be used to improve one's own forecasts.

I.2.1 Organization of the learning problem

Let us consider an electricity market in which m wind farms participate. Every farm j produces a given amount of electricity $x_{j,t}$ at step t, where all the productions have been divided by the nominal capacity P_j of the corresponding wind farm j in order to scale them. In the grid we will take interest in one given wind farm owner. We will call him the *central agent* and he will be the one who tries to improve his forecasts. The central agent owns a set of wind farms, called a *portefolio*, $\Omega_p = \{s_{p,1}, s_{p,2}, \dots, s_{p,m_p}\}$ of m_p wind farms. This set can either consist of one single farm (*i.e.* $m_p = 1$), a couple of farms, or even be the whole set of wind farms of the grid (*i.e.* $m_p = m$). In the latter case the central agent is the *system operator*.
On the grid, another set $\Omega_a = \{s_{a,1}, s_{a,2}, \dots, s_{a,m_a}\}$ of m_a other wind farms is present. The owners of those farms will be referred as *contracted agents*. It is them who will help the central agent to improve his production forecasts in exchange of something, which justifies the term of "contracted".
In order to make notations easier, the productions $x_{j,t}$ of the different wind power plants will be ordered: the indices $j \in \{1, 2, \dots, m_p\}$ will correspond to the productions of the farms from Ω_p belonging to the central agent. $j \in \{m_p + 1, m_p + 2, \dots, m\}$ will correspond to the farms of the contracted agents.

The goal of the central agent is to predict one step in advance his average electricity production y_t of his portefolio:

$$y_t = \frac{1}{\sum_{j=1}^{m_p} P_j} \sum_{j=1}^{m_p} P_j \, x_{j,t}$$

where t denotes one time step which is equivalent to 15 minutes in real life in our case.

The modelization we chose for the average production y_t of the central agent is a stationary autoregressive model of lag ℓ which both takes in account the data of the portefolio Ω_p of the central agent, and the exogenous input coming from the contracted agents' wind farms Ω_a. This yields:

$$y_t = \sum_{j=1}^{m_p} \sum_{\tau=1}^{\ell} \beta_{j,\tau} x_{j,t-\tau} + \sum_{j=m_p+1}^{m} \sum_{\tau=1}^{\ell} \beta_{j,\tau} x_{j,t-\tau} + \varepsilon_t \tag{I.4}$$

where the $\beta_{j,\tau}$ are the coefficients of the autoregression and ε_t is a noise. Hence the first double sum corresponds to the wind farms of his portefolio Ω_p, while the second one corresponds to the farms Ω_a of the contracted agents. This equation may be formulated in the vectorial form

$$y_t = \boldsymbol{x}_t \boldsymbol{\beta} + \varepsilon_t \tag{I.5}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{m\ell}$ is the vector containing all the data $x_{j,t-\tau}$ for $\tau \in \{1, 2, \ldots, \ell\}$ ordened by agent and $\boldsymbol{\beta} \in \mathbb{R}^{m\ell}$ is the vector of all $\beta_{j,\tau}$ also ordened by agent. Both vectors may be thus rewritten $\boldsymbol{x}_t = [\boldsymbol{x}_t^1\, \boldsymbol{x}_t^2\, \ldots\, \boldsymbol{x}_t^m]$ and $\boldsymbol{\beta} = [\boldsymbol{\beta}^1\, \boldsymbol{\beta}^2\, \ldots\, \boldsymbol{\beta}^m]$, with each sub-vector $\boldsymbol{x}_t^i$ or $\boldsymbol{\beta}^i$ corresponding to agent i. Figure I.1 describes the architecture of the learning problem and may make getting used to notations easier.

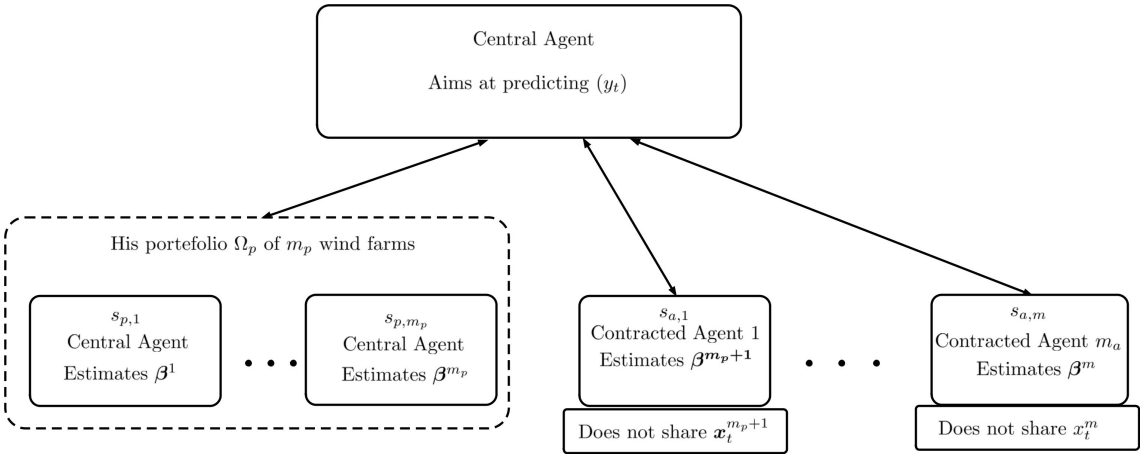

Figure I.1: Organization of the distributed problem

In order to get a prediction of the average production of the central agent y_t, we want to estimate the autoregression vector $\boldsymbol{\beta}$, namely $\hat{\boldsymbol{\beta}}$. The hypothesis is that the production is stationary, *i.e.* that the coefficient $\boldsymbol{\beta}$ is constant over time. This debatable hypothesis will be discussed later on and is one of the reasons why the estimation method must be improved. Since we want to perform predictions one step (15 minutes) ahead, an estimation of the average production at time t obtained at time $t-1$ would thus be $\hat{y}_{t|t-1} = \boldsymbol{x}_t \hat{\boldsymbol{\beta}}$. One must be careful with the chosen notation since $\boldsymbol{x}_t$ only holds information from time $t-\ell$ to time $t-1$, and not t.

Usually, such an estimator $\hat{\boldsymbol{\beta}}$ is calculated by solving a regular least-square problem. However in our case, we try to ensure sparsity of the estimator. That is why we chose to use a lasso type estimator. As mentionned earlier, lasso estimation consists in adding a ℓ_1 penalty to the least-squares problem. If one has access to data corresponding to the time steps t between $\ell+1$ and T[1], putting all $\boldsymbol{x}_t$ and y_t together in respectively a design matrix $\mathbb{X}$ and a response vector $\boldsymbol{Y}$ yields the *batch* lasso estimation problem:

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \frac{1}{2} \|\mathbb{X}\boldsymbol{\beta} - \boldsymbol{Y}\|^2 + \lambda |\boldsymbol{\beta}| \tag{I.6}$$

[1]estimation cannot be started at steps below $\ell+1$ due to the nature of the AR model

where $\lambda \in \mathbb{R}_+$ is a parameter which controls the size of the shrinkage. The greater it becomes, the more the sparsity of the estimation $\hat{\boldsymbol{\beta}}$ will increase. However this comes to the cost of a small bias of the lasso estimator (*i.e.* $\mathbb{E}[\hat{\boldsymbol{\beta}}_{\text{lasso}}] \neq \boldsymbol{\beta}$) due to the shrinkage of the non-zero coefficients. However this is a small price to pay to reduce to improve overall forecast through sparsity and to increase calculations speed. The term "batch" refers to the fact that this estimation is carried out on a stationary data set, the obtained estimation $\hat{\boldsymbol{\beta}}$ then being applied on new incoming data. Advantages and drawbacks of this kind of estimation process is of major importance in this project and are discussed in I.3.1.

I.2.2 Performing distributed learning through the Alternative Direction Method of Multipliers

Similarly to what has been performed in [2], the goal is to distribute the optimization problem I.6 among the different agents. The choice of method has been again the one of the Alternate Direction method of the Multipliers (ADMM) since it yields good results. The main difference consists in that in our case we only focus on improving the forecast of the central agent, thus having only a scalar autoregressive model whereas in [2] the improvement of forecast was done for all the wind farms of the grid.

General formulation and advantages of the ADMM

The ADMM is a variant of dual ascent which is particularly adapted to solve optimization problems in which the objective function may be splitted into two functions. Let us consider for instance the optimality problem where the objective function may be divided in two distinct parts :

$$\min_{\boldsymbol{\beta}} f(\boldsymbol{\beta}) + g(\boldsymbol{\beta})$$

ADMM consists in adding a supplementary variable $\boldsymbol{z}$ through affine constrains and then successively minimizing the augmented lagragian of the problem $\mathcal{L}_\rho$ on the main variable $\boldsymbol{\beta}$ and on the additional one $\boldsymbol{z}$. Typically the optimization problem rewritten under ADMM form is :

$$\begin{cases} \min f(\mathbb{A}\boldsymbol{z}) + g(\boldsymbol{\beta}) \\ \text{s.t.} \quad \boldsymbol{\beta} - \mathbb{A}\boldsymbol{z} = c \end{cases}$$

The introduction of the additional variable $\boldsymbol{z}$ permits to take advantage of the specificities of f and g to perform efficient optimization and to break a master problem into parts which can be distributed. The choice of minimizing the augmented lagragian over the regular one gives better convergence properties, though increasing the difficulty of the optimality problem.

ADMM for our lasso problem

Let us consider our lasso problem (I.6). The objective function perfectly fits in the case of ADMM since it may be split into to parts which are $f(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbb{X}\boldsymbol{\beta} - \boldsymbol{Y}\|^2$ and $g(\boldsymbol{\beta}) = \lambda|\boldsymbol{\beta}|$. In order to put the problem under ADMM form we introduce the variables $\boldsymbol{z}^i \in \mathbb{R}^{(T-\ell)m}$, $i \in \{1, 2, \ldots, m\}$ through the affine constrains $\mathbb{X}_i\boldsymbol{\beta}^i - \boldsymbol{z}^i = 0$, where $\mathbb{X}_i$ is the submatrix of $\mathbb{X}$ corresponding to agent i (one has $\mathbb{X} = [\mathbb{X}_1\, \mathbb{X}_2\, \ldots\, \mathbb{X}_m]$). Considering the separability of the ℓ_1 norm $|\cdot|$, the ADMM problem which is to be solved is :

$$\begin{cases} \min \frac{1}{2} \left\| \sum_{i=1}^{m} \boldsymbol{z}^i - \boldsymbol{Y} \right\|^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}^i = 0, \, \forall i = 1..m \end{cases} \tag{I.7}$$

and the corresponding augmented lagragian of parameter ρ, $\mathcal{L}_\rho$, is:

$$\mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{u}) = \left\| \sum_{i=1}^{m} \boldsymbol{z}^i - \boldsymbol{Y} \right\|^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| + \rho \sum_{i=1}^{m} u^{i\top} (\mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}^i) + \frac{\rho}{2} \left\| \mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}^i \right\|^2 \tag{I.8}$$

where the $\boldsymbol{u}^i$ are the scaled dual variables ($\boldsymbol{u}^i = \frac{\boldsymbol{\mu}^i}{\rho}$ if $\boldsymbol{\mu}^i$ denotes the regular dual variable). The linear and quadratic term in $\boldsymbol{\beta}^i$ are usually put together, which yields the expression:

$$\mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{u}) = \left\| \sum_{i=1}^{m} \boldsymbol{z}^i - \boldsymbol{Y} \right\|^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| + \frac{\rho}{2} \sum_{i=1}^{m} \left\| \mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}^i + \boldsymbol{u}^i \right\|^2 \tag{I.9}$$

where the term $-\frac{\rho}{2} \sum_{i=1}^{m} \|u^i\|^2$ has been dropped since minimization will occur on $\boldsymbol{\beta}$ and $\boldsymbol{z}$ and not on $\boldsymbol{u}^i$. As mentioned earler ADMM procedure succesively minimizes $\mathcal{L}_\rho$ on $\boldsymbol{\beta}$ and then on $\boldsymbol{z}$. The $\boldsymbol{u}$ update is performed last and consists in a regular dual ascent update. If k denotes the k-ieth step in the iterations, then the roughest forms of the ADMM updates are:

$$\begin{cases} \boldsymbol{\beta}_k &= \text{argmin}_\beta \left\{ \mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{z}_{k-1}, \boldsymbol{u}_{k-1}) \right\} \\ \boldsymbol{z}_k &= \text{argmin}_z \left\{ \mathcal{L}_\rho(\boldsymbol{\beta}_k, \boldsymbol{z}, \boldsymbol{u}_{k-1}) \right\} \\ \boldsymbol{u}_k^i &= \boldsymbol{u}_{k-1}^i + \mathbb{X}_i \boldsymbol{\beta}_k^i - \boldsymbol{z}_k^i \end{cases} \tag{I.10}$$

However as one can see, for now distribution of the learning problem has not been achieved and the updates may seem complicated.

I.2.3 Algorithmic scheme of ADMM

The updates I.10 can in fact be drastically simplified. The augmented lagragian is separable in the $\boldsymbol{\beta}^i$, which allows to distribute the updates of these variables among agents. Furthermore, other properties and tricks will reduce the number of calculations needed to perform the updates and therefore increase the efficiency of this method.

Updates of $\boldsymbol{z}$ and $\boldsymbol{u}$

While the $\boldsymbol{z}$ and $\boldsymbol{u}$ update respectively occur in second and third during a loop, simplifications which happen for them also allow simplifications for $\boldsymbol{\beta}^i$ and explain why we are dealing with them first. For a start we will focus on the $\boldsymbol{z}$ update.

Since the update is performed on $\boldsymbol{z}$ only, the second term of I.9 is dropped. Conversly to $\boldsymbol{\beta}^i$, the augmented lagragian isn't separable in $\boldsymbol{z}$, which forces to perform a joint update on all the $\boldsymbol{z}^i$. While this is achievable, it would mean to solve an optimization problem of $m(T - \ell)$ variables. Usually m is around 40 or 50, T around 5000, and ℓ is 2 or 3. Jointly minimizing on $\boldsymbol{z} = [\boldsymbol{z}^1 \, \boldsymbol{z}^2 \, \ldots \, \boldsymbol{z}^m]$ would therefore represent a huge computational cost. However this can be avoided with the introduction of *fusion centers*. Let $\overline{\boldsymbol{z}}_k = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{z}_k^i$ and $\overline{\boldsymbol{X}\boldsymbol{\beta}}_k = \frac{1}{m} \sum_{i=1}^{m} \mathbb{X}_i \boldsymbol{\beta}_k^i$ be, the indice k refering to the k-ieth step of the loop of the algorithm. Those variables are called fusion centers since they gather all the different

variables from the distributed problem. The problem of the $\boldsymbol{z}$ update is thus reformulated by introducing $\overline{\boldsymbol{z}}$:

$$\begin{cases} \boldsymbol{z}_k = \underset{\boldsymbol{z}}{\text{argmin}} \frac{1}{2} \|m\overline{\boldsymbol{z}} - \boldsymbol{Y}\|^2 + \frac{\rho}{2} \sum_{i=1}^{m} \left\| \mathbb{X}_i \boldsymbol{\beta}_k^i - \boldsymbol{z}^i + \boldsymbol{u}_{k-1}^i \right\|^2 \\ \text{s.t.} \quad \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{z}^i - \overline{\boldsymbol{z}} = 0 \end{cases} \tag{I.11}$$

If $\boldsymbol{\nu}$ denotes the dual variable for this problem, since the objective function is convex, a necessary and sufficient optimality condition is the zero-gradient condition of the lagragian. The latter is written:

$$\forall i = 1..m, \; -\rho(\mathbb{X}_i \boldsymbol{\beta}_k^i - \boldsymbol{z}^i + \boldsymbol{u}_{k-1}^i) + \frac{\boldsymbol{\nu}}{m} = 0$$

After summing for i from 1 to m to get the term $\frac{\boldsymbol{\nu}}{\rho m}$, we get the following expression for $\boldsymbol{z}_k^i$:

$$\boldsymbol{z}_k^i = \overline{\boldsymbol{z}} + \mathbb{X}_i \boldsymbol{\beta}_k^i - \overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \boldsymbol{u}_{k-1}^i - \overline{\boldsymbol{u}}_{k-1}, \quad \text{where } \overline{\boldsymbol{u}}_{k-1} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{u}_{k-1}^i. \tag{I.12}$$

Let us now consider the $\boldsymbol{u}^i$ update. It is $\boldsymbol{u}_k^i = \boldsymbol{u}_{k-1}^i + \mathbb{X}_i \boldsymbol{\beta}_k^i - \boldsymbol{z}_k^i$. If one reinjects I.12 in this update, one obtains :

$$\boldsymbol{u}_k^i = \overline{\boldsymbol{u}}_{k-1} + \overline{\boldsymbol{X}\boldsymbol{\beta}}_k - \overline{\boldsymbol{z}}_k \tag{I.13}$$

The dual variables are thus the same for all the agents, so that $\overline{\boldsymbol{u}}_k \equiv \boldsymbol{u}_k$ and doesn't depend of the agent i considered. Furthermore the additional variable $\boldsymbol{z}_k^i$ of agent i is replaced by $\overline{\boldsymbol{z}}_k$: the former can also be completely replaced by the fusion center $\overline{\boldsymbol{z}}$. It can easily be obtained by solving I.11 after reinjection of I.12 [2], which yields:

$$\overline{\boldsymbol{z}}_k = \frac{1}{m+\rho} (\boldsymbol{Y} + \rho \overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \rho \boldsymbol{u}_{k-1}) \tag{I.14}$$

Update of the β^i

As one can see with I.9, $\mathcal{L}_\rho$ is separable in the $\boldsymbol{\beta}^i$. This means that the updates can be performed separately by the different agents. Furthermore, since the first minimization occurs on $\boldsymbol{\beta}$, the first term of the lagragian can be left out. Therefore one has:

$$\boldsymbol{\beta}_k^i = \text{argmin}_{\beta^i} \left\{ \lambda |\boldsymbol{\beta}^i| + \frac{\rho}{2} \left\| \mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}_{k-1}^i + \boldsymbol{u}_{k-1}^i \right\|^2 \right\}$$

Using the results I.12 and I.13 one has $\|\mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{z}_k^i + \boldsymbol{u}_k^i\| = \left\| \mathbb{X}_i \boldsymbol{\beta}^i - \overline{\boldsymbol{z}}_k - \mathbb{X}_i \boldsymbol{\beta}_k^i + \overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \boldsymbol{u}_k \right\|$ (since $\boldsymbol{u}_k^i - \overline{\boldsymbol{u}}_k = 0$). The best form of the $\boldsymbol{\beta}^i$ update is therefore :

$$\begin{aligned} \boldsymbol{\beta}_k^i &= \underset{\boldsymbol{\beta}^i}{\text{argmin}} \left\{ \lambda |\boldsymbol{\beta}^i| + \frac{\rho}{2} \left\| \mathbb{X}_i \boldsymbol{\beta}^i - \boldsymbol{Y}_{k-1}^i \right\|^2 \right\} \\ \text{where} \quad \boldsymbol{Y}_{k-1}^i &:= \mathbb{X}_i \boldsymbol{\beta}_{k-1}^i - \overline{\boldsymbol{X}\boldsymbol{\beta}}_{k-1} + \overline{\boldsymbol{z}}_{k-1} - \boldsymbol{u}_{k-1} \end{aligned} \tag{I.15}$$

The update scheme of the ADMM algorithm for a lasso-type estimation is thus performed as following:

[2] see appendix for this quick calculation

$$\begin{cases} \boldsymbol{\beta}_k^i & = & \underset{\boldsymbol{\beta}^i}{\text{argmin}}\Big\{\lambda|\boldsymbol{\beta}^i| + \frac{\rho}{2}\left\|\mathbb{X}_i\boldsymbol{\beta}^i - \boldsymbol{Y}_{k-1}^i\right\|^2\Big\} \\ \overline{\boldsymbol{X}\boldsymbol{\beta}}_k & = & \frac{1}{m}\sum_{i=1}^{m}\mathbb{X}_i\boldsymbol{\beta}_k^i \\ \overline{\boldsymbol{z}}_k & = & \frac{1}{m+\rho}(\boldsymbol{Y} + \rho\overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \rho\boldsymbol{u}_{k-1}) \\ \boldsymbol{u}_k & = & \boldsymbol{u}_{k-1} + \overline{\boldsymbol{X}\boldsymbol{\beta}}_k - \overline{\boldsymbol{z}}_k \\ \boldsymbol{Y}_k^i & = & \mathbb{X}_i\boldsymbol{\beta}_k^i - \overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \overline{\boldsymbol{z}}_k - \boldsymbol{u}_k \end{cases} \tag{I.16}$$

While every other variable has an explicit form for their update, $\boldsymbol{\beta}_k^i$ is still formulated as the argmin of an optimality problem. Instead of calculating explicitely $\boldsymbol{\beta}_k^i$ (which would represent additional calculation), a method called *shooting* may be used. Shooting consists is an iterative method which will calculate the coefficients of $\boldsymbol{\beta}_k^i$ independently. We will not elaborate here on shooting, and the reader may find details about it at A.2 in appendix A.

If η indicates the minimal tolerated norm increment $\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}$ between interations and M_{loop} is the maximum number of loops performed (both for the ADMM iterations and shooting) then the complete batch algorithm goes as in Algorithm 1. [3]

Algorithm 1 Batch distributed learning

1: Initialize $\boldsymbol{\beta}_0, \boldsymbol{z}_0, \boldsymbol{u}_0$ at 0 ; Initialize k at 1
2:
3: **while** $\|\boldsymbol{\beta}_k - \boldsymbol{\beta}_{k-1}\| > \eta$ and $k \leq M_{\text{loop}}$ **do**
4: **for** $i = 1, \ldots, m$ **do** ▷ Distributed among the agents
5: Update $\mathbf{Y}_{k-1}^i$ as in I.16
6: $\boldsymbol{\beta}_k^i \leftarrow$ Shooting$(\mathbb{X}_i, \mathbf{Y}_{k-1}^i, \lambda, \rho)$
7: Agent i sends $\mathbb{X}_i\boldsymbol{\beta}_k^i$ to the central one
8: **end for**
9:
10: Central agent updates $\overline{\boldsymbol{X}\boldsymbol{\beta}}_k$, $\overline{\boldsymbol{z}}_k$ and $\boldsymbol{u}_k$ as in I.16
11: Central agent sends $\overline{\boldsymbol{X}\boldsymbol{\beta}}_k$, $\overline{\boldsymbol{z}}_k$ and $\boldsymbol{u}_k$ to the contracted agents.
12: $k \leftarrow k + 1$
13:
14: **end while**

return $\boldsymbol{\beta}_k$

I.3 Shortfall and drawbacks of batch ADMM

As one can see in [3], batch ADMM yields good results. Indeed, it improves forecast quality in terms of RMSE and MAE when compared to regular AR by 1 % to 4 % percents. Furthermore, it yields sparse estimators, though still not achieving the sparsity of the sVAR method. Notwithstanding those results,

[3] Note concerning line 3: Normally in the algorithm, one adds a minimal amount of loops to be performed to avoid it of doing 0 loops. This has been omitted to make the algorithm easier to read.

it has a share of drawbacks. These drawbacks may both pose a threat to the quality of estimation, but also to the feasibility of distributed learning through ADMM in real life. In this part we will thus focus on the limits of ADMM, which prove the need for another way of proceeding to distributed learning.

I.3.1 The limit of the stationarity hypothesis

When postulating the model I.4, we made the hypothesis that the wind production is stationary, *i.e.* that $\boldsymbol{\beta}$ is constant over time. The issue is that wind production is subject to high variability, and as such the coefficients of $\boldsymbol{\beta}$ may change over time, both in values and in the farms which have non-zero coefficients. However the issue that since batch ADMM learns over a fixed data set and will average its estimation over the whole data set.

In order to illustrate that point, we proceed to a test on non-stationary synthetic data. We first generate a time-varying weight vector $\boldsymbol{\beta}_t$ where only 8 coefficients are non-zero, and 80 coefficients are zero. The 8 non-zero coefficients will be step functions with a structural break at instant t_1 (which means that at $t = t_1$ their value will suddenly change). In this example, the structural break of $\boldsymbol{\beta}_t$ will be taken at $t_1 = 3500$.
The synthetic design matrix $\mathbb{X}$ and response vector $\boldsymbol{Y}$ are then created as following: every column j of $\mathbb{X}$ is obtained by generating a sample of length $T = 7000$ of gaussian data $\mathcal{N}(0, \sigma_j^2)$, where $\sigma_j^2 \in [0.2, 1]$. The corresponding response vector $\boldsymbol{Y}$ is obtained by applying the following formula for each of its coefficients y_t:

$$y_t = \boldsymbol{x}_t \boldsymbol{\beta}_t + \varepsilon_t$$

where $\boldsymbol{x}_t$ is as usual the t-ieth line of $\mathbb{X}$, $\boldsymbol{\beta}_t$ is the value of the weight vector at step t and ε_t is a noise. However due to randomness in the generation of such a pair $(\mathbb{X}, \boldsymbol{Y})$, we will repeat this procedure $N = 20$ times with the same vector $\boldsymbol{\beta}_t$. This will tone down errors which could be linked to the specificity of a given data set.

Batch ADMM is then performed on those $N = 20$ data sets $(\mathbb{X}, \boldsymbol{Y})$ for three different learning periods T_{batch}: one where $T_{\text{batch}} < t_1$ (learning only on data before the structural break), one where $t_1 < T_{\text{batch}} < T$ (where the majority of the data comes from before the structural break) and finally one where $T_{\text{batch}} = T$ (as much data from before and after the structural break). Finally for each of those three cases the average over the 20 simulations is taken, yielding three estimated weight vectors corresponding to three different sizes of the learning set.

As one can see with fig. I.2, if $T_{\text{batch}} < t_1$ *i.e.* the learning is done before the structural break happens, the value before the break is estimated. If $t_1 < T_{\text{batch}} < T$, which means that algorithm has more data coming from before the structural break, the estimate value is closer to the first value before the break. Finally if $T_{\text{batch}} = T$, which means the learning algorithm has as much data from before and after the jump, the estimated value is the average of both.

Since in the case of real data, the autoregression coefficients β will not be constant over time. This fundamental limit of batch learning, which averages estimation over the learning time, may be very detrimental to the quality of forecasts. That is why the ability to track time-varying parameters is needed to improve forecast accuracy even further than with batch ADMM.

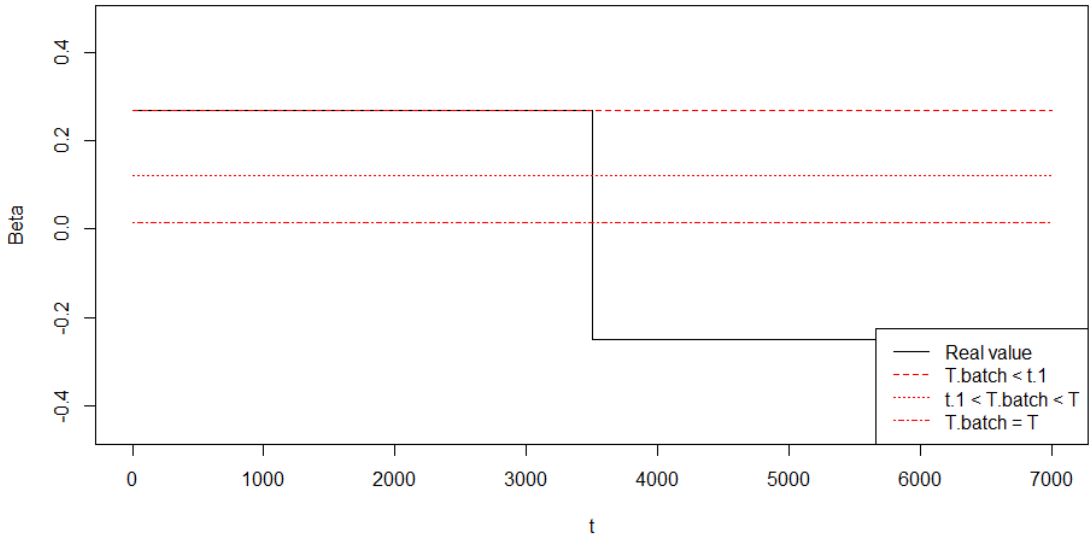

Figure I.2: Plot of a non-zero coefficient of β with a structural break at $t_1 = 3500$ and its batch ADMM estimations for different sizes T_{batch} of the learning set

I.3.2 Convenience of the batch ADMM algorithm in real life

In the previous part, we showed that tracking was impossible with batch learning. It simply averages regression coefficients over time, which would make an estimated vector $\hat{\beta}$ obsolete after a major variation of meteorological condition in a real life case. One could think that performing multiple small batch estimations over time may tackle this issue, but this would reveal another flaw of batch learning.
Indeed, it needs large learning data sets in order to beat simple methods. This is illustrated with figure I.3 which plots the relative improvement of batch ADMM over a regular AR estimation for different grid sizes m and for different sizes of the learning set T_{batch}. The test have been performed on a real data set, namely the one which we will use again in II.4. $N = 20$ simulations have been performed for each couple (m, T_{batch}) and boxplots used in order to tone-down errors which could be linked to a particular set of wind farms.

As one can see, the bigger the number of total agents on the grid m gets, the greater the learning set must be for ADMM to be able to compete with simple methods. This is an issue since grids generally gather more than $m = 40$ wind farms, for which batch ADMM would need more than 5000 samples to learn (which represents 52 days of data). This great amount of data needed makes batch learning unconvenient to use and the effective use of multiple batch procedures very difficult.

Furthermore, batch learning is expensive in calculations. If one considers that a vectorial operation costs 1 flop in the programming language **R**, the distributed learning algorithm presented above yields a global complexity which is generally between $\mathcal{O}(mM_{\text{loop}}\ell^2)$ and $\mathcal{O}(mM_{\text{loop}}^2\ell^2)$, where M_{loop} is the maximal amount of loops tolerated in the global algorithm and the shooting one. With $m \approx 40$ and $M_{\text{loop}} = 200$, the computational cost can increase very quickly.

Finally, implementing the batch distributed learning algorithm in a concrete case is not an easy task. It needs repeated exchanges between the central agent and the contracted ones, which make the development of a platform where distribution would be done tedious. Furthermore because of the high

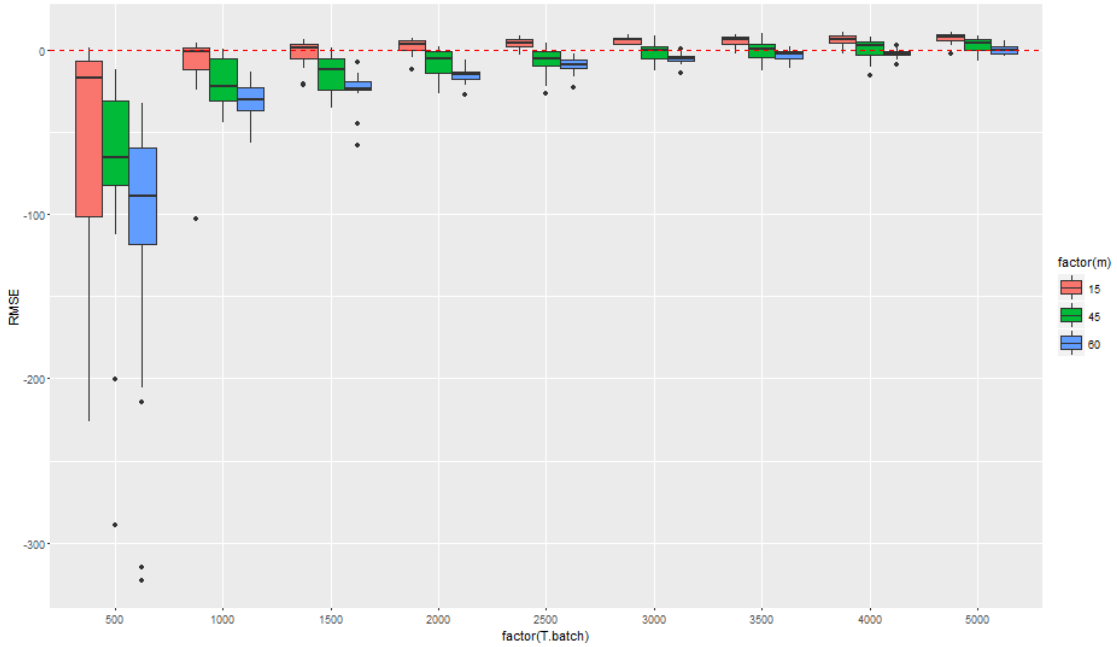

Figure I.3: Boxplots of RMSE improvement (in %) of batch ADMM relatively to regular AR for different sizes T_{batch} of the learning set and different grid sizes m

number (M_{loop}) of exchanges, contracted agents may not agree to participate in the procedure. Besides that, each exchange involves trading multiple vectors of size T_{batch}, which leads to effective exchanges of the size of a few megabytes. This may seem small, but one must keep in mind that in general in a smart grid every actor both plays the role of central and contracted agent. With the increasing number of wind power plants in smart grids this may lead to problems in data management.

Therefore online learning could also be the best solution to develop a platform where exchanges are fewer and of smaller size, which would make it effectively feasible.

Part II

Development of Online ADMM for distributed learning

Wind production is as everybody knows very subject to variability. As such, wind electricity production can't be considered as stationary on long time scales. However in order to yield accurate estimations, batch learning must be performed on large time scales on which the hypothesis of stationarity can't hold. That is why the main goal of this project was to elaborate an "online" version of distributed learning. Online learning would not need the hypothesis of stationarity and achieve quicker calculations than batch estimation through recursion. One must keep in mind that our main constrain is the confidentiality of the data of the different agents and is what makes writing an online ADMM algorithm so difficult.

II.1 Transitioning from batch to online estimation

Online learning differs significantly from batch estimation. In this part we detail the major differences between batch and online learning, as well as presenting why ADMM cannot be simply adapted to an online scheme.

II.1.1 Differences between batch and online learning

Until now, we considered a fixed data set on which we performed one single estimation. The obtained $\hat{\boldsymbol{\beta}}$ estimator is then kept once and for all, and is applied on new incoming data to obtain a forecast. Online learning on the contrary is an "on-the-fly" method, which means that the estimation is adapted whenever new data is available.
More precisely, in the case of online learning the loss function ℓ is not considered constant anymore. It depends now on the time step t and changes at every iteration. Therefore an online learning problem at step t may be written:

$$\min_{\boldsymbol{\beta}_t} \frac{1}{2} \sum_{\tau=1}^{t-1} \ell_\tau\big(\boldsymbol{\beta}_t\big)$$

And the goal is to find at each step $\boldsymbol{\beta}_t$ which minimizes the cumulative loss over all the past iterations.

Usually, the minimization problem does not change a lot from step t to step $t+1$: the true goal is thus to obtain a recursive update of the estimatation $\boldsymbol{\beta}_t$ through a small correction which takes into account the new data obtained at iteration $t+1$. This is for instance the case in the most simple online

learning method, namely online least-squares.[1]. Based on the Newton-Raphson method, the update of the estimator $\boldsymbol{\beta}$ takes the form of a scaled gradient depending on the most recent data available. Recursion is what gives online methods the ability to track time varying signals, as long as their variations are smooth enough. Furthermore, recursion dramatically drops the computational cost of estimation, thus effectively adressing one of the issues of batch learning.

In our case, we add a ℓ_1 penalty to cause sparsity of the estimator and use a quadratic loss function. Therefore the online lasso problem at iteration t may be written:

$$\min_{\boldsymbol{\beta}_t} \sum_{\ell+1}^{t-1} \frac{1}{2}(\boldsymbol{x}_\tau \boldsymbol{\beta}_t - y_\tau) + \lambda|\boldsymbol{\beta}_t| \tag{II.1}$$

II.1.2 The difficulty to develop an online scheme in our case

Sadly a simple Newton-Raphson method does not yield a recursive ADMM scheme in our case due to the presence of the lasso penalty. In fact, developing a purely only scheme of ADMM seems to be a complicated task.

Recently, online methods for it have been developed in [4] and [5]. However the method presented in [4] is not purely online, since the estimator is updated by a matricial calculations and not by recursion. The second method presented in [5] consists in performing a linear approximation of the loss function , which gives the possibility of a purely online update of the estimator under the right conditions. However after running tests in our situation, it yielded even poorer results than OLS.
Furthermore, one of the major issues in our case is that the contracted agents don't want to share their data. This confidentiality clause needs to take particular measures to avoid direct data sharing, something which is not at all addressed in both papers mentioned above. In the case of batch learning, this confidentiality was naturally ensured by the product $\mathbb{X}\boldsymbol{\beta}$, making that the information $\mathbb{X}$ was never shared alone. Together the exchanges between central agent and contracted ones yielded a system with more unknowns than equations, thus effectively protecting the data. In the method which has been developed and which will be presented hereafter, sharing data under one form or another is not avoidable. However the idea of a product protecting the data as in [3] has been the inspiration for the data protection system.

As one will see, we managed to develop a new online method to achieve distributed learning. However, considering the remarks made above, calling it purely "online" wouldn't be accurate. Indeed, it consists in the sharing and updating of five quantities, on which only three are recursive. Moreover, one of the non-recursive updates involves the resolution of a non-sigular $m\ell \times m\ell$ linear system which, though always yielding an unique solution, costs $\mathcal{O}\Big((m\ell)^{c_1}\Big)$, $c_1 \in]2, 3]$ flops. To comparison, the whole ADMM scheme presented in the first part had a complexity between $\mathcal{O}\Big(M_{\text{loop}} m\ell^2\Big)$ and $\mathcal{O}\Big(M_{\text{loop}}^2 m\ell^2\Big)$: the complexity of our "semi-online" method has thus at least a quadratic complexity in the number of agents m, while for the batch method developed above it was linear in the number of agents. We therefore expect the online method to be slower than the one we have already presented.
Ergo one must keep in mind that despite the name of online ADMM we will use, the method is not purely online but rather semi-online. We still don't know if ADMM may be adapted to a purely online fashion without approximations: no such algorithms do exist yet. It even might be that it isn't possible to completely adapt ADMM to online learning, and that if one wants to perform purely online distributed learning, he or she should take interest in other methods.

[1] details on online least-squares can be found in appendix B

II.2 Encrypted online ADMM

In this part we present the core of the work of this research project. The problems when using ADMM to create an online estimation methods were mainly of two different natures. First of all, adapting the ADMM scheme to the online formulation of our problem II.1 was a bit tricky. Secondly, we had to keep information private since the contracted agents refused to share their data.
The adjective "encrypted" will be justified through the method presented below.

II.2.1 Encrypted OADMM formulation

Let us consider the online learning problem II.1. At first, we simply wanted to adapt what was done for instance in [3] for the online case. However the major difference with it is that this time, it is not a constant matrix which multiplies the estimator but a time varying vector. The problem is that ADMM only deals with affine constrains which are constant over time, and thus could not be applied litterally in our online learning problem. Therefore our first objective was to find a way to formulate II.1 appropriately to use the ADMM.

The first possibility we saw to do so was to introduce the $\boldsymbol{z}^i$ in the most simple way possible, namely through the constrains $\boldsymbol{\beta}^i - \boldsymbol{z}^i = 0, \forall i$. It would yield the following ADMM optimization problem:

$$\begin{cases} \min \dfrac{1}{2} \displaystyle\sum_{\tau=\ell+1}^{t-1} (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \boldsymbol{z}^i = 0, \ \forall i \in \{1, 2, \dots, m\} \end{cases} \tag{II.2}$$

However after developing to the end the calculations, it would inevitably lead to the sharing of the vectors $\boldsymbol{x}_t^i$ corresponding to the data of the different agents i, which is forbidden in our case.

However, pondering over the work [3], we had another idea for this kind of approach. What protected the information in the batch case was the fact that the information $\mathbb{X}$ was always multiplied by the vector $\boldsymbol{\beta}$. Our idea was thus to introduce a matrix $\mathbb{M}$ which would serve a similar purpose, *i.e.* multiplying $\boldsymbol{x}_t^i$ in calculations every time it appears. We therefore introduce the additional variables $\boldsymbol{z}^i$ through the affine constrains $\boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0, \ \forall i$, where $\mathbb{M}_i$ is a non-singular matrix of size $\ell \times \ell$ chosen privately by agent i. We refer to these matrices as *encryption matrices* since they allow to protect the data $\boldsymbol{x}_t^i$ as it will be proved later. While in general the ADMM does not need to chose matrices which are invertible for the constrains, not doing so might cause troubles of singularity in a few situations, hence the additional hypothesis of $\mathbb{M}_i$ which are invertible. This does not cause a great loss of generality and does not endanger the privacy of the data. The online ADMM optimality problem is thus:

$$\begin{cases} \min \dfrac{1}{2} \displaystyle\sum_{\tau=\ell+1}^{t-1} (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \mathbb{M}_i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0, \ \forall i \in \{1, 2, \dots, m\} \end{cases} \tag{II.3}$$

to which the augmented lagragian is:

$$\mathcal{L}_\rho(\boldsymbol{\beta}, \boldsymbol{z}, \boldsymbol{u}) = \frac{1}{2} \sum_{\tau=\ell+1}^{t-1} (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \mathbb{M}_i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| + \frac{\rho}{2} \sum_{i=1}^{m} \left\| \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i + \boldsymbol{u}^i \right\|^2 \tag{II.4}$$

In order to make calculations clearer, we introduce the vectors $\boldsymbol{\beta}$ and $\boldsymbol{z} \in \mathbb{R}^{m\ell}$ which result respectively of the binding of vectors $\boldsymbol{\beta}^i$ and $\boldsymbol{z}^i$, and the blockwise diagonal matrix $\mathbb{M} = \text{diag}(\mathbb{M}_1, \mathbb{M}_2, \dots, \mathbb{M}_m)$.

As a reminder, the ADMM scheme consists in minimizing the augmented lagragian successively over the main variable $\boldsymbol{\beta}$ and the additional variable $\boldsymbol{z}$.

Update of β^i

The augmented lagragian is separable in the $\boldsymbol{\beta}^i$, which means that their updates can be performed separably. Furthermore since the minimization occurs on the $\boldsymbol{\beta}^i$, the first term in $\boldsymbol{z}$ of the augmented lagragian can be left out, which gives:

$$\boldsymbol{\beta}_t^i = \text{argmin}_\beta \Big\{ \underbrace{\lambda|\boldsymbol{\beta}^i| + \frac{\rho}{2}\left\|\boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}_{t-1}^i + \boldsymbol{u}_{t-1}^i\right\|^2}_{:=\mathcal{L}_{\beta^i}} \Big\}$$

Although the ℓ_1 norm is not differentiable, since it is convex it is sub-differentiable. As such we have the optimality condition:

$$\boldsymbol{\beta}_t^i \in \text{argmin}_\beta \mathcal{L}_{\beta^i} \Leftrightarrow 0 \in \partial\mathcal{L}_{\beta^i}(\boldsymbol{\beta}_t^i)$$

where $\partial\mathcal{L}_{\beta^i}(\boldsymbol{\beta}_t^i)$ denotes the sub-differential of $\mathcal{L}_{\beta^i}$ at $\boldsymbol{\beta}_t^i$. Finally a few calculations which are detailled in the appendix yield the update at step t:

$$\boldsymbol{\beta}_t^i = \mathcal{S}_{\lambda/\rho}(\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i) \tag{II.5}$$

where $\mathcal{S}_{\lambda/\rho}$ is the vectorial *soft-thresholding* operator whose definition may be found in the glossary.

Update of z

This time, the second term of the augmented lagragian may be dropped. However it is not separable in the $\boldsymbol{z}^i$ this time, which means the minimization must be performed on $\boldsymbol{z}$ and not the different $\boldsymbol{z}^i$. This is a major drawback and is precisely what impinges on our online ADMM method. In the case of the batch ADMM, the problem of non-separability was solved through the introduction of a fusion center $\overline{\boldsymbol{z}}$. While we would have enjoyed to be able to introduce either a fusion center $\overline{\boldsymbol{z}}$ or $\overline{\mathbb{M}\boldsymbol{z}}$, this is not possible due to the multiplication of $\mathbb{M}_i\boldsymbol{z}^i$ by $\boldsymbol{x}_\tau^i$. It leads to a dead end and cannot be performed this time.

The zero-gradient condition with respect to $\boldsymbol{z}$ thus yields:

$$\sum_{\tau=\ell+1}^{t-1} (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M}\boldsymbol{z} - y_\tau) - \rho \mathbb{M}^\top (\boldsymbol{\beta}_t - \mathbb{M}\boldsymbol{z} + \boldsymbol{u}_{t-1}) = 0$$

Let $\mathbb{H}_t = \sum_{\tau=\ell+1}^{t} (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M})$ and $\boldsymbol{P}_t = \sum_{\tau=\ell+1}^{t} (\boldsymbol{x}_t \mathbb{M})^\top y_\tau$ be. Finally the $\boldsymbol{z}_t$ update is given by the solution to the linear system:

$$\Big(\mathbb{H}_{t-1} + \rho\, \mathbb{M}^\top \mathbb{M}\Big) \boldsymbol{z}_t = \boldsymbol{P}_{t-1} + \rho\, \mathbb{M}^\top (\boldsymbol{\beta}_t + \boldsymbol{u}_{t-1}) \tag{II.6}$$

Therefore in order to perform this update, the different contracted agents simply give the central agent their share $\boldsymbol{x}_t^i \mathbb{M}_i$, the data being efficiently protected by the multiplication by the encryption matrix. However as one can see, this update also needs to know $\mathbb{M}^\top\mathbb{M}$, which means that the different agents would have to share their bloc of the matrix $\mathbb{M}_i^\top \mathbb{M}_i$. The reader will see in the next section that this poses a threat to privacy. Ergo the $\boldsymbol{z}$ update can't be left as it is now to perform distributed learning.

Update of $\boldsymbol{u}^i$

This update isn't problematic, is recursive and can even be distributed among agents :

$$\boldsymbol{u}_t^i = \boldsymbol{u}_{t-1}^i + \boldsymbol{\beta}_t^i - \mathbb{M}_i \boldsymbol{z}_t^i \tag{II.7}$$

Updates of $\mathbb{H}_t$ and $\boldsymbol{P}_t$

These two quantities are the major difference to the already existing OADMM methods. They indirectly contain all the past information $(\boldsymbol{x}_\tau, y_\tau)_{\tau=\ell+1,\dots,t}$ and may be updated recursively:

$$\mathbb{H}_t = \mathbb{H}_{t-1} + (\boldsymbol{x}_t\mathbb{M})^\top(\boldsymbol{x}_t\mathbb{M}) \tag{II.8}$$

$$\boldsymbol{P}_t = \boldsymbol{P}_{t-1} + (\boldsymbol{x}_t\mathbb{M})^\top y_t \tag{II.9}$$

However the main issue if one lets these updates this way is that the quantities $\mathbb{H}_t$ and $\boldsymbol{P}_t$ progressively store information over all the learning steps. After many iterations the matrices will have accumulated a lot of past information, which might not be necessarly relevant anymore (especially in the case of wind production which is variable). $\mathbb{H}_t$ and $\boldsymbol{P}_t$ will thus have a certain inertia to change. This will cause a slowing down of convergence after a major change (this point is illustrated in part II.3.2). This kind of problem is common in online learning procedures and may be addressed through the introduction of a *forgetting factor*.

Introducing a forgetting factor $\mu \in]0,1]$ consists in reformulating the optimality problem as following:

$$\begin{cases} \min \dfrac{1}{2} \displaystyle\sum_{\tau=\ell+1}^{t-1} \mu^{t-1-\tau} (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \mathbb{M}_i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0,\ \forall i \in \{1,2,\dots,m\} \end{cases}$$

The coefficient μ will thus help the algorithm to forget past data which might not be relevant for current estimation. More precisely, it gives a memory of $\nu = 1/(1-\mu)$ steps to the algorithm and thus should be tuned according to the memory one desires to give it. While one could think that chosing a low μ in order to give it a short memory is a good idea and improves tracking abilities, it is not as easy as that. Indeed, there is always a tracking/stability quandary. Decreasing μ to improve tracking capacities causes instability, especially in periods of stationarity. This finally drops the quality of estimation dramatically and is counter-productive. That is why to adress this issue we introduce a *dynamic forgetting factor* which would only enter in action during phases of major change. It takes the form of a sigmoid, similarly to the one presented in [1]:

$$\mu_t = 1 - \frac{b}{1 + \exp\Big(c(\hat{e}_t - a)\Big)} \tag{II.10}$$

where:

- $\hat{e}_t$ is a mesure of prediction error over a given number of past steps, as for instance MAE or RMSE. In our case we chose the MAE of the estimation over the past 3 iterations (we average over the 3 past iterations to tone down the effects of an eventual outlier) $\hat{e}_t = \frac{1}{3}\sum_{\tau=0}^{2} |y_{t-\tau} - \hat{y}_{t-\tau}|$. This way the forgetting factor will be effective when the prediction suddenly is far away from the real data (which often means a change of parameters due to particular meteorological conditions) and allow

a "reset" of the algorithm. However the averaging will cause a small delay in the activation of the dynamic forgetting factor, which is not a big matter though.

- b quantifies the maximum amount of forgetting. It should be tuned according of the minimum memory one wants the algorithm to have $\nu = 1/b$
- c tunes the abruptness of the slope at the inflexion point of the sigmoid.
- a is an activation threshold of the dynamic forgetting factor. One should tune a a bit above the usual RMSE or MAE so that the dynamic forgetting factor activates when the estimation gets too bad.

One may refer to figure II.1 to see how the dynamic forgetting factor enters into action a few lags after the threshold has been crossed.

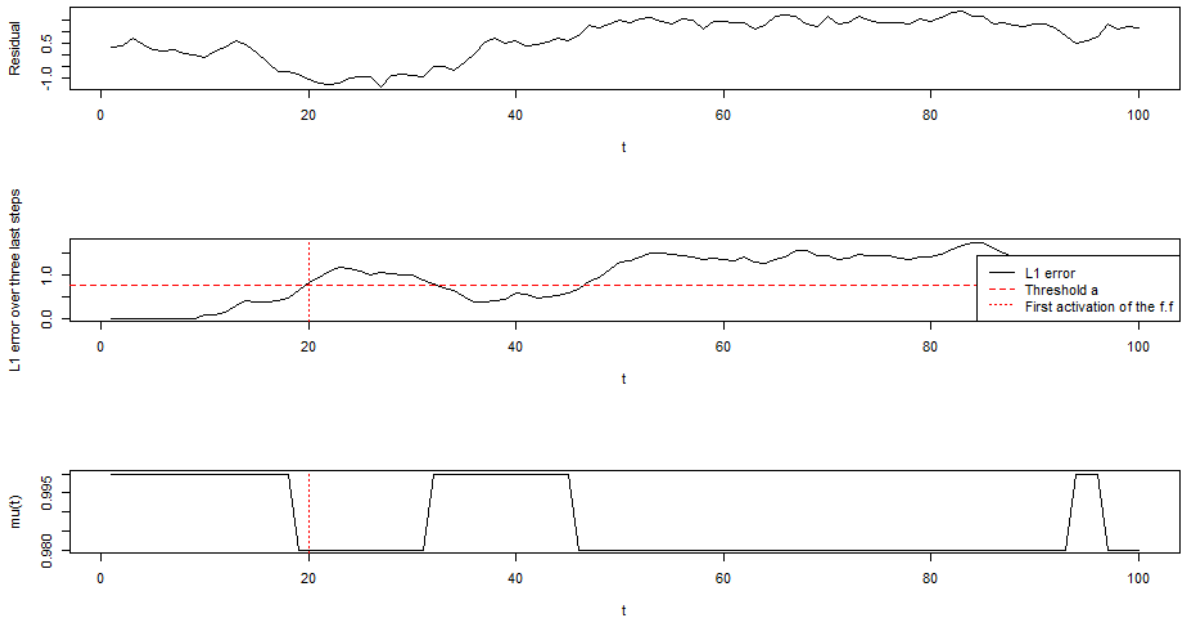

Figure II.1: Working principle of the dynamic forgetting factor μ_t

The formulation in the case of a dynamic forgetting factor is similar to the one of a regular forgetting factor. Old data is multiplied by the dynamic forgetting factors of all the later steps, whereas the latest incoming data is not multiplied by any forgetting factor to leave enough learning data to the algorithm. The optimality problem to solve is therefore written:

$$\begin{cases} \min \frac{1}{2} \sum_{\tau=\ell+1}^{t-1} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \mathbb{M}_i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0, \, \forall i \in \{1, 2, \dots, m\} \end{cases} \tag{II.11}$$

Analogous calculations[2] to the previous ones prove that the $\boldsymbol{\beta}^i$ and $\boldsymbol{z}$ updates remain the same, while the ones of $\mathbb{H}_t$ and $\boldsymbol{P}_t$ take the form:

$$\mathbb{H}_t = \mu_t \mathbb{H}_{t-1} + (\boldsymbol{x}_t \mathbb{M})^\top (\boldsymbol{x}_t \mathbb{M}) \tag{II.12}$$

[2]which can be found in Appendix C

$$\boldsymbol{P}_t = \mu_t \boldsymbol{P}_{t-1} + (\boldsymbol{x}_t \mathbb{M})^\top y_t \tag{II.13}$$

II.2.2 Organization of the learning problem with OADMM

The organization of the online learning problem is quite different from the batch one. While the batch ADMM algorithm only needed a contracted agent to send $\mathbb{X}_i \boldsymbol{\beta}_k^i$ to the central agent, the online version needs the sharing of $\boldsymbol{x}_t^i \mathbb{M}_i$, $\mathbb{M}_i^\top \mathbb{M}_i$ and finally the term $\mathbb{M}_i^\top (\boldsymbol{\beta}_t^i + \boldsymbol{u}_{t-1}^i)$ to achieve distributed learning. The problem is that by sending all this, the central agent has the opportunity to find his way back to the information of an agent $\boldsymbol{x}_t^i$. Indeed, let us consider the three shared information in the algorithmic scheme:

- $\hat{y}_{t|t-1}^i = \boldsymbol{x}_t^i \boldsymbol{\beta}_t^i$ and $\boldsymbol{\beta}_t^i$, respectively the estimation of agent i's part of y_t and tof the total vector of the autoregressive model. $\hat{y}_{t|t-1}^i$ is shared so that the central agent can get his forecast, while $\boldsymbol{\beta}_t^i$ is shared to see if it is worth contracting the agent in question or not (if $\boldsymbol{\beta}^i \approx 0$ it means that the farm of agent i doesn't add a lot of accuracy to the estimation and might be a waste of money). It gives one equation with ℓ unknows which are the coefficients of $\boldsymbol{x}_t^i \in \mathbb{R}^\ell$.

- $\boldsymbol{x}_t^i \mathbb{M}_i$: since $\mathbb{M}_i \in \mathcal{M}_\ell(\mathbb{R})$, it gives the central agent ℓ equations and ℓ^2 other unknows.

- $\mathbb{M}_i^\top \mathbb{M}_i$ gives him ℓ^2 additional equations with no further unknowns.

- $\mathbb{M}_i^\top (\boldsymbol{\beta}_t^i + \boldsymbol{u}_{t-1}^i)$. This finally yields ℓ other equations with ℓ more variables through $\boldsymbol{u}_{t-1}^i$ (the reader might have noted that it is not necessary for the different agents to directly share $\boldsymbol{u}_{t-1}^i$ though) .

The central agent has therefore $\ell^2 + 2\ell + 1$ equations for $\ell^2 + 2\ell$ variables. Although the system is not linear, it presents a risk for being solved and thus to be unfitted to answer the confidentiality constrain set for this problem.

First we tried to change the place of matrix $\mathbb{M}$ in II.3 in hope to find a wind or a trick to avoid having to share $\mathbb{M}^\top \mathbb{M}$. After that all those tries had failed, we realized that if there was no other way than having to share $\mathbb{M}^\top \mathbb{M}$, we should add a secondary safety to protect the encryption matrices $\mathbb{M}_i$. That is how we had the idea to introduce secondary encryption matrices $\mathbb{K}_i$. Every agent thus chooses a second encryption matrix $\mathbb{K}_i$ of size $\ell \times \ell$, but this one needs to be orthogonal. Instead of sharing $\mathbb{M}_i^\top \mathbb{M}_i$, an agent would share $\mathbb{K}_i \mathbb{M}_i$. The secondary encryption matrix would then disappear in the calculation of $\mathbb{H}_t + \rho \mathbb{M}^\top \mathbb{M}$ since $\mathbb{K}_i^\top \mathbb{K}_i = \mathrm{Id}$.
This method indeed solves the issue of confidentiality as we desired. Sharing $\mathbb{K}_i \mathbb{M}_i$ gives the central agent ℓ^2 additional equations but also adds ℓ^2unknowns. Finally the system obtained by all the sharing yields $2(\ell^2 + \ell)$ unknowns for only $\ell^2 + 2\ell + 1$ equations. Therefore $\boldsymbol{x}_t^i$, $\mathbb{M}_i$ nor $\mathbb{K}_i$ can't be calculated with certainty, effectively protecting the whole system. Furthermore contrarly to $\mathbb{M}_i$, $\mathbb{K}_i$ can be changed at any moment (as long as it stays orthogonal), which makes up for another security form of the algorithm. Finally the form and order of the updates of the different variables is al following :

$$\begin{cases} \boldsymbol{\beta}_t^i = \mathcal{S}_{\lambda/\rho}(\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i) \\ \Big(\mathbb{H}_{t-1} + \rho(\mathbb{K}\mathbb{M})^\top(\mathbb{K}\mathbb{M})\Big)\boldsymbol{z}_t = \boldsymbol{P}_{t-1} + \rho\mathbb{M}^\top(\boldsymbol{\beta}_t + \boldsymbol{u}_{t-1}) \\ \boldsymbol{u}_t^i = \boldsymbol{u}_{t-1}^i + \boldsymbol{\beta}_t^i - \mathbb{M}_i \boldsymbol{z}_t^i \\ \mathbb{H}_t = \mu_t \mathbb{H}_{t-1} + (\boldsymbol{x}_t\mathbb{M})^\top(\boldsymbol{x}_t\mathbb{M}) \\ \boldsymbol{P}_t = \mu_t \boldsymbol{P}_{t-1} + (\boldsymbol{x}_t\mathbb{M})^\top y_t \end{cases} \tag{II.14}$$

Where $\mathbb{K} = \text{diag}(\mathbb{K}_1, \mathbb{K}_2, \ldots, \mathbb{K}_m)$. This may be performed as long as the different actors desire. The complete OADMM algorithm hence works as described in algorithm 2. Fig. II.2 represents the organization of the online distributed learning scheme and may make it easier to understand.

Algorithm 2 Online distributed learning

Initialize $\boldsymbol{\beta}_0, \boldsymbol{z}_0, \boldsymbol{u}_0, \mathbb{H}_0$ and $\boldsymbol{P}_0$ at 0.

while the agents want to perform distributed learning **do**

  for $i = 1, \ldots, m$ **do** ▷ Distributed among agents
    $\boldsymbol{\beta}_t^i$ is calculated as in II.14
    $\hat{y}_{t|t-1}^i = \boldsymbol{x}_t \boldsymbol{\beta}_t^i$
    Agent i sends $\hat{y}_{t|t-1}^i$, $\boldsymbol{\beta}_t^i$, $\boldsymbol{x}_t^i\mathbb{M}_i$, $\mathbb{K}_i\mathbb{M}_i$ and $\mathbb{M}_i^\top(\boldsymbol{\beta}_t^i + \boldsymbol{u}_{t-1}^i)$ to the central one
  end for

  Central agent builds $\hat{y}_{t|t-1}$ and performs his forecast.
  y_t is revealed to him
  Central agent updates $\boldsymbol{z}_t, \mathbb{H}_t$ and $\boldsymbol{P}_t$ as in II.14 and sends the $\boldsymbol{z}_t^i$

  for $i = 1, \ldots, m$ **do** ▷ Distributed among agents
    Update $\boldsymbol{u}_t^i$ as in II.14
  end for

end while

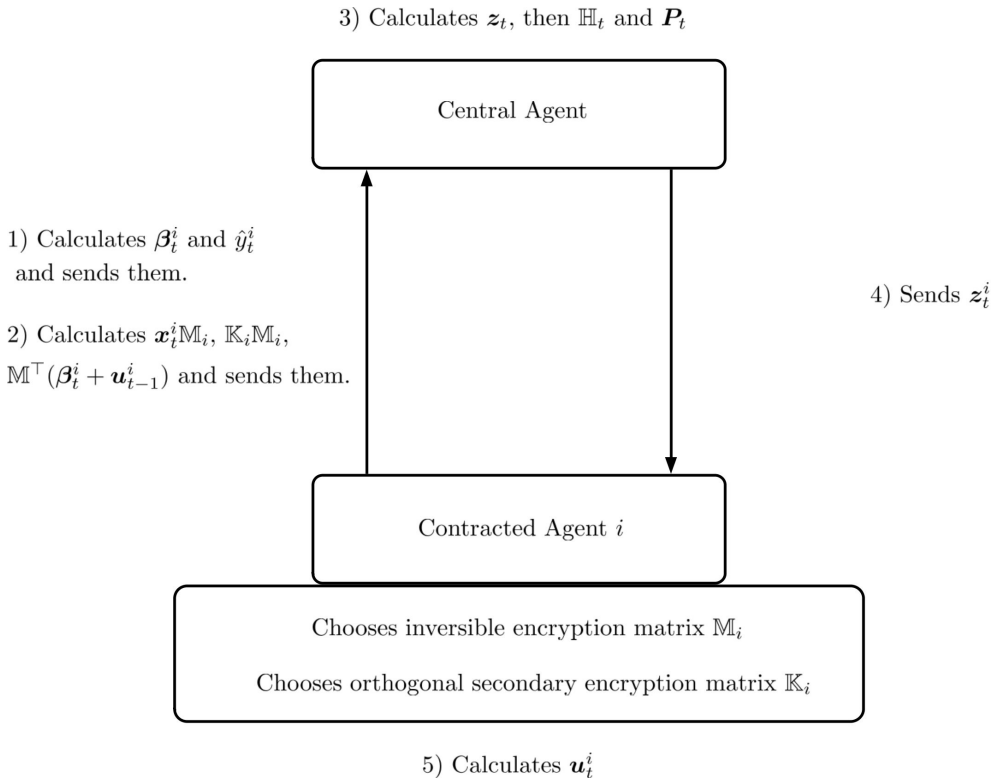

Figure II.2: Organization of the online distributed problem

II.2.3 Reduction of the size of the exchanges when compared to batch ADMM

A major issue when doing distributed learning in general is the quantity of data shared between the different actors. Indeed, receivers sometimes struggle to cope with the quantity of data shared with them. With the advent of Big Data this problem will only get worse, thus needing to find solutions to perform distributed learning with exchanges as small in data as possible. From this point of view, OADMM is a major leap forward when compared to batch ADMM.

Let us consider the regular ADMM algorithm, more specifically the exchange from the central agent to one agent i. Though the former only needs to send three vectors to the latter, namely $\overline{\boldsymbol{X}\boldsymbol{\beta}}_k$, $\overline{\boldsymbol{z}}_k$ and $\boldsymbol{u}_k$, all three are of size T_{batch} which is generally around 5000. Even if one does not need to store those variables in memory and as such the number of loops performed doesn't matter, for one agent this exchange still accounts for $3T_{\text{batch}}$ coefficients. Considering that on a grid, typically $m_a = 50$ other agents are present, at every iteration a contracted agent receives $3m_aT_{\text{batch}}$ coefficients. If one codes these coefficients on a simple float, this represents approximatively 25 Mbit of data received per iteration.

Though it might not seem to much, datasets and number of agents in grids will only increase over time, making such an expensive in exchanges method unconvenient to use. While the organization of the distributed learning problem for OADMM seems more complicated, it reduces the effective size of exchanges between agents.

This time the exchange in the way central agent to agent i only needs the sending of vector $\boldsymbol{z}_t^i$, which accounts only for $\ell = 2$ or 3 coefficients, effectively costing less than 96 bits. Even if one considers the other way around (namely agent i sending data to central agent), the cost of calculation is still decreased.

While ADMM needs the sending of $\mathbb{X}_i\beta_k^i$ for every agent i, effectly representing $m_a T_{\text{batch}}$ coefficients, the three quantities send towards the central agent (see II.2.2) together account for $\ell^2 + 2\ell$ coefficients, representing in total $m_a(\ell^2 + 2\ell)$ received coefficients which is again a lot smaller.

II.3 Analysis of OADMM's performance on synthetic data

In this section, we will ensure that the OADMM algorithm developed works properly. In order to do that, we will apply it on synthetic data. Accuracy and competitiveness compared to state-of-the-art methods will not be addressed here but in the next part. Notations will be the same as in the rest of the paper, though their meaning will obviously be more abstract.

We will generate two types of synthetic data sets. The first will be stationary autoregressive data, whereas the second will be gaussian data with structural breaks, like in I.3.1. As usual in order to avoid randomness of results, we will generate in each test case $N = 20$ independant data pairs $(\mathbb{X}, \boldsymbol{Y})$ of synthetic data. OADMM will be applied on each of them, and the average estimation over the simulations then taken.

The sets generated in this section will always be of size $T = 7000$. The lag ℓ is set to 2. λ and ρ are respectively fixed to 5 and 0.5, unless the contrary is stated. The weight vector $\boldsymbol{\beta}$, which will be common to all the N simulations, will only have $m_p = 4$ blocks of ℓ consecutive non-zero coefficients and $m_a\ell = 80$ zero coefficients. The dynamic forgetting factor will be shut off for the first test case which involves stationary data (*i.e.* $b = 0$) and appropriately tuned in the second case. Matrices $\mathbb{M}$ and $\mathbb{K}$ will be randomly generated once and fixed for all the N simulations. Though normally the central agent does not need to choose encryption matrices himself, we will still do it, which may slow down the algorithm a little but also allows us to assess the robustness of our algorithm.

II.3.1 Stationary autoregressive data

Let us first generate a weight vector $\boldsymbol{\beta} = [\boldsymbol{\beta}^1\, \boldsymbol{\beta}^2 \ldots \boldsymbol{\beta}^m]$ where only m_p subvectors corresponding to a set of indices Ω_p are non-zero. We remind that the weight vector is the same for all the $N = 20$ simulations.
The data generation procedure hereafter is identical for all the N simulations. We generate m_p autoregressive time-series $\{y_t^i\}$ of lag ℓ on the model $y_t^i = \boldsymbol{x}_t^i\boldsymbol{\beta}^i + \varepsilon_t^i$ where $\boldsymbol{x}_t^i = (y_{t-1}^i\, y_{t-2}^i)$ and ε_t^i is a noise. The sum of those m_p time-series is taken, which yields the response vector $\boldsymbol{Y}$. The according design matrix $\mathbb{X}$ is created as following: the submatrices $\mathbb{X}_i$ for $i \in \Omega_p$ are obtained by binding by rows the $\boldsymbol{x}_t^i$ for $t \in \{\ell + 1, \ldots, T\}$. The rest of the design matrix $\mathbb{X}$ is filled with other autoregressive data generated in a same fashion as previously. This effectively yields a linear model $\boldsymbol{Y} = \mathbb{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where the only non-zero coefficients of $\boldsymbol{\beta}$ correspond to the subvectors $\boldsymbol{\beta}^i$, $i \in \Omega_p$.
Our online ADMM algorithm is thus applied on each of the N couples of data $(\mathbb{X}, \boldsymbol{Y})$ giving N estimations, which are finally averaged, yielding an estimator $\hat{\boldsymbol{\beta}}$.

As one can see fig. II.3, the non-zero coefficients of the weight vector $\boldsymbol{\beta}$ are perfectly estimated by the algorithm. However the achieved average sparsity for $\lambda = 5$ is only around 7%, which is rather poor. This is why OADMM is again applied on the same data sets $(\mathbb{X}, \boldsymbol{Y})$ for $\lambda \in \{20, 35, 50\}$ and the same $\rho = 0.5$. This results in a sparsity that dramatically increases as one can see fig. II.4, going up to 89% for $\lambda = 50$.
However this increase of sparsity comes at a price. First, a higher λ/ρ ratio overly shrinks the coefficients of the estimator $\boldsymbol{\beta}$, effectively growing the bias of the lasso estimator: this can impinge on

the estimation and lead to an increase of the prediction error. Furthermore, a higher λ/ρ ratio means that the soft-thresholding operator will either add or substract λ/ρ in an exaggerate manner, slowing down the convergence of the algorithm. These two drawbacks can be seen fig. II.5: for higher λ the asymptotic ℓ_1 error is higher than for lower ones because of the excessive shrinkage. Furthermore, while for $\lambda = 5$ the OADMM algorithm only takes around 700 iterations to meet the convergence criterion $\sum_{i \in \Omega_p} |\hat{\boldsymbol{\beta}}^i - \boldsymbol{\beta}^i| < 5 \times 10^{-3}$ (corresponding to roughly to 1% of the average absolute value of a non-zero coefficient of $\boldsymbol{\beta}$), it takes 5000 iterations for $\lambda = 50$.

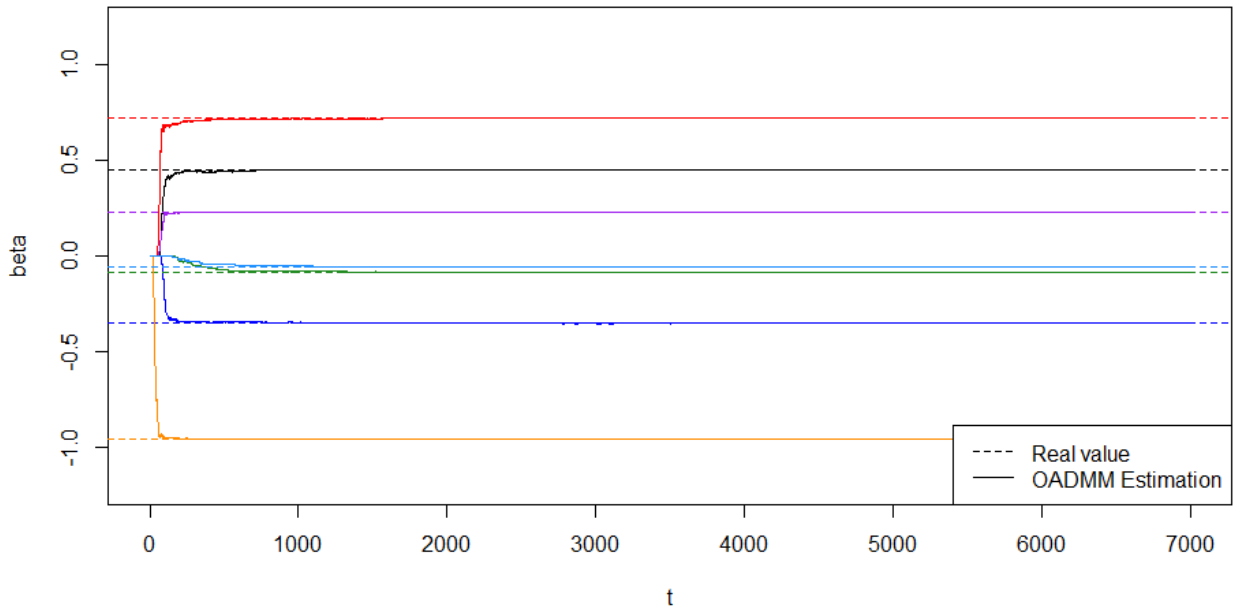

Figure II.3: Average estimation over the $N = 20$ simulations of the 8 non-zero coefficients of $\boldsymbol{\beta}$ for $\lambda = 5$ and $\rho = 0.5$

Therefore one cannot just increase the λ/ρ to obtain increased sparsity. It comes at a price that one must always keep in mind when tuning the parameters. There is a quandary between prediction error and sparsity, and one should chose their parameters according to what is desired: reducing prediction error or promoting sparsity. A good compromise between the two should therefore be chosen.

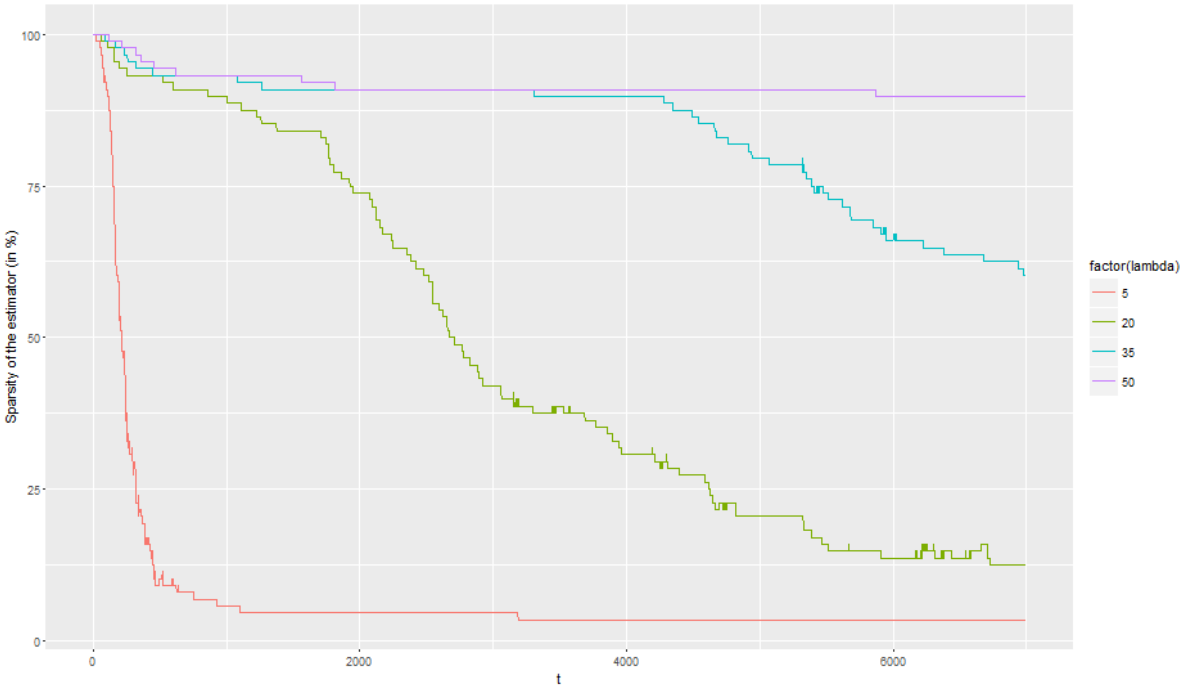

Figure II.4: Sparsity (in %) of the average estimator $\hat{\boldsymbol{\beta}}$ over the $N = 20$ simulations, over time and for different λ

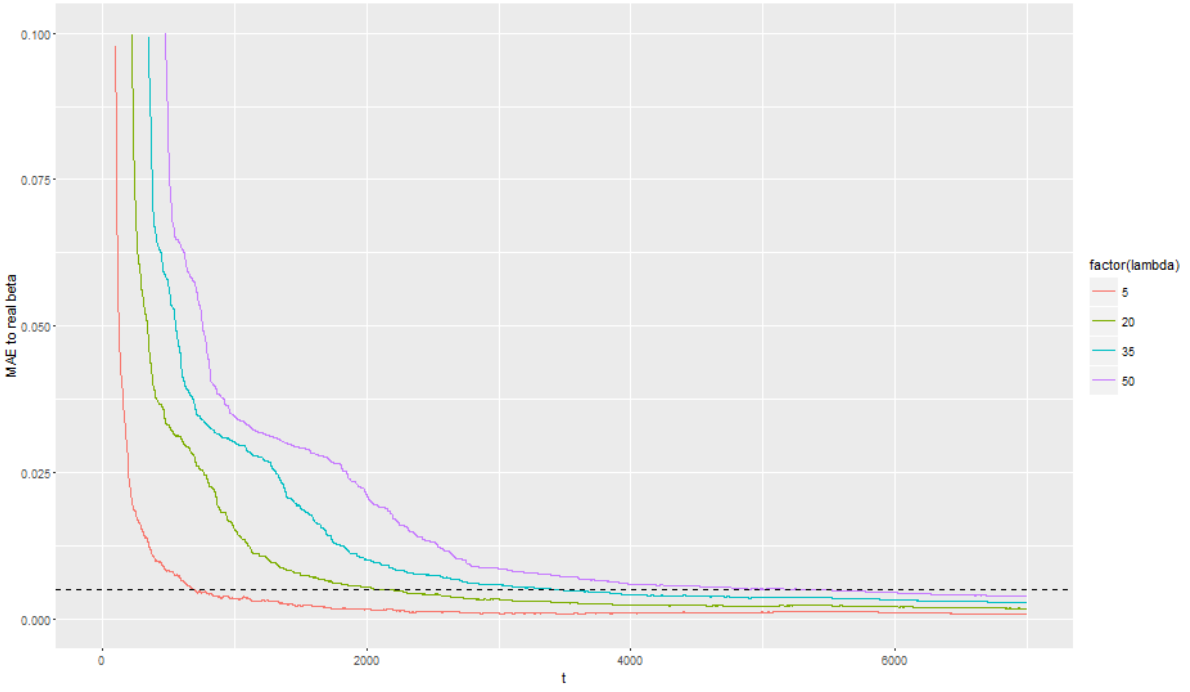

Figure II.5: ℓ_1 error $\sum_{i \in \Omega_p} |\hat{\boldsymbol{\beta}}^i - \boldsymbol{\beta}^i|$ of the average estimation over the $N = 20$ simulations

II.3.2 Non-stationary data with structural breaks

Now that OADMM has proven to be efficient on stationary data, we will apply it on time-varying data to evaluate its tracking abilities, which was one of the motivations when conceiving it.

The test will be the same as one performed in I.3.1. We generate a time dependent weight vector $\boldsymbol{\beta}_t$ with only $m_p = 4$ blocks of $\ell = 2$ consecutive non-zero coefficients, and $m_a\ell = 80$ zero ones. The non-zero coefficients will be stepwise constant function with a set of structural breaks $\mathcal{R} = (t_1, t_2, \ldots, t_r)$. Again, $N = 20$ synthetic data sets $(\mathbb{X}, \boldsymbol{Y})$ will be generated exactly like in I.3.1. In this section we consider again the case of a single structural break at $t_1 = T/2$, *i.e.* $\mathcal{R} = \{3500\}$. The reader may refer to C.3 the appendix for a case with multiple structural breaks.

OADMM is then applied on each of the couples $(\mathbb{X}, \boldsymbol{Y})$, and the average of all the $N = 20$ estimations is then calculated.

In order to illustrate the need for a forgetting factor, this was first performed with $b = 0$, which means without forgetting at all. As one can see in II.7, the algorithm "realizes" that a structural break is happening at $t = 3500$ and the estimated values $\hat{\boldsymbol{\beta}}_t$ are starting to change. However the new convergence is extremely slow due to the inertia accumulated in both $\mathbb{H}_t$ and $\boldsymbol{P}_t$. Therefore without forgetting factor the online algorithm has tracking abilities, but they are not sufficient.

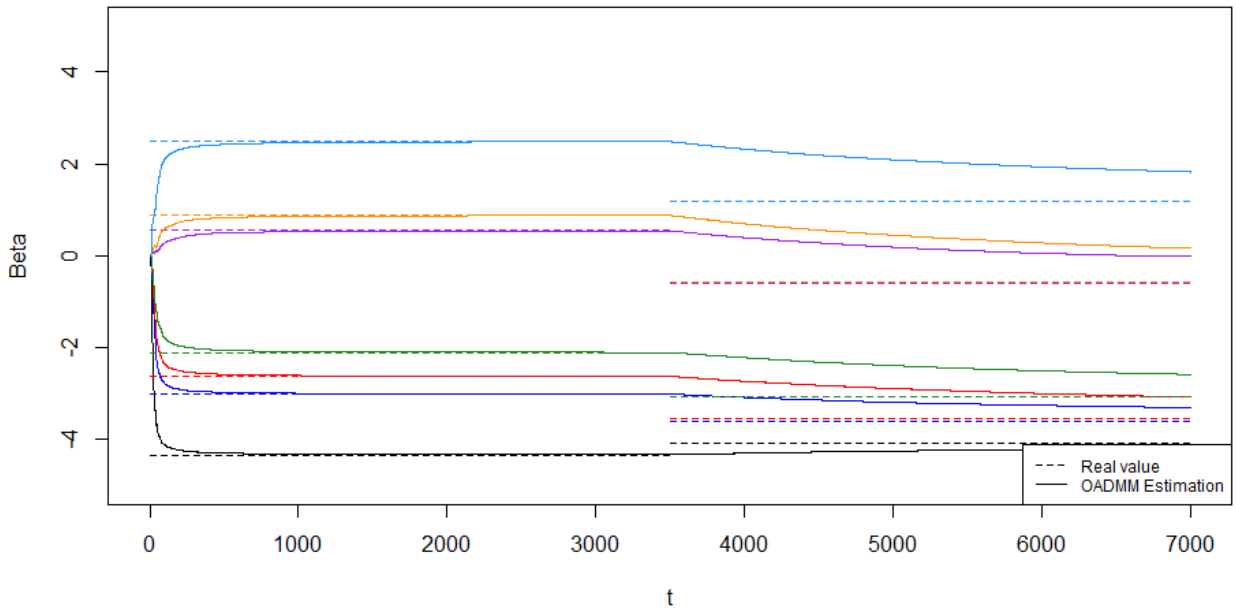

Figure II.6: Average estimation of the non-zero coefficients of $\boldsymbol{\beta}_t$ with a single structural break, without dynamic forgetting factor

We finally put the parameters of the dynamic forgetting factor at $b = 0.02$ (corresponding to a learning memory of 50 steps), $a = 0.7$ (since the average MAE is approximately 0.5) and $c = -800$ (to have a quickly reacting forgetting factor). As one can see fig. II.8, the dynamic forgetting factor steps-in at the very moment the structural break happens, resulting in a dramatically increased tracking ability: the new convergence reached in average only 800 steps after the structural break.

Despite the great improvement in tracking the dynamic forgetting factor brings to the OADMM algorithm, it also has drawbacks. Adding three supplementary parameters to the algorithm, it makes

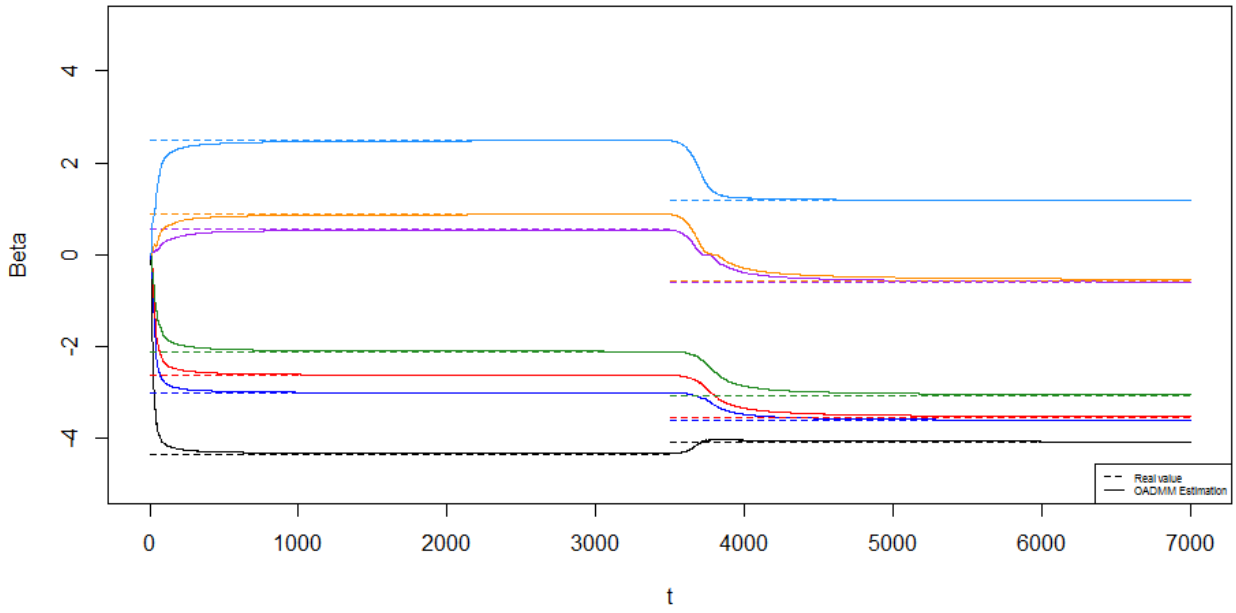

Figure II.7: Average estimation of the non-zero coefficients of β_t with a single structural break, with dynamic forgetting factor

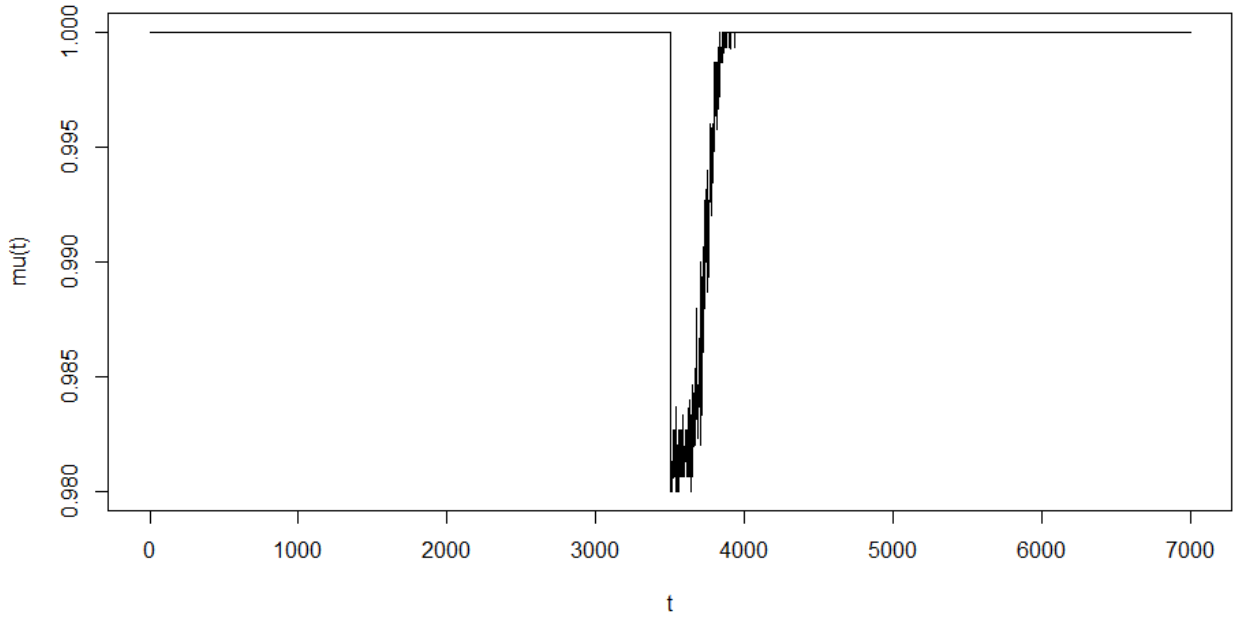

Figure II.8: Average value over the $N = 20$ simulations of the dynamic forgetting factor μ_t

tuning extremely tedious. This matter is addressed in detail in II.4.1 and may be one of the biggest problems with our online ADMM method.

II.4 A complete test case on the Danish data set

Since the results on synthetic data were satisfying, we will now apply OADMM on a real data set. It consists of power measurements of 349 Danish wind power plants performed every 15 minutes between 01/01/2006 and 31/03/2012, yielding a total of 219066 observations for each farm. The data was been scaled, which means that the measurements of a farm have been divided by the corresponding nominal power of that farm. The data has also been cleaned from all its measurement errors or corrupted data.

We will consider three test situations, which correspond to different sizes of portefolio. The first will correspond to the owner of a single wind farm (*i.e.* $m_p = 1$), the second will correspond to the owner of a small portefolio Ω_p of size $m_p = 15$. The final case will be the one of the system operator of the grid and will most farms of the grid. The process of the test case for the latter differs from the two other ones, and will not be explained here but in II.4.4 instead. Every time OADMM will compete with two state-of-the-art methods, namely batch ADMM presented in the first part and online least-squares. Since the latter does not provide any possibility of confidentiality, estimation for it will be performed only using the portefolio Ω_p and without exogenous input. The tuning of the parameters λ, ρ and of the dynamic forgetting factor will be adressed in the next sub-section.

We take $T = 40,000$ consecutive time steps from the Danish data set to form our working data set. For the case of batch estimation, a subset consisting of $T_{\text{batch}} = 25,000$ consecutive samples will serve as the learning set, yielding a batch estimator $\hat{\boldsymbol{\beta}}$. One step-ahead batch estimations of y_t for the next $15,000$ steps are then obtained by $\hat{y}_{t|t-1} = \boldsymbol{x}_t\hat{\boldsymbol{\beta}}$. OLS and OADMM estimations are naturally performed on-the-fly for the $T = 40,000$ time steps. Finally we assess the performance of every of the three methods. Again we proceed to $N = 20$ simulation. Boxplots are generally used in order to tone down errors which could be due to the specificities of a given portefolio of wind farms. In every of those N simulations a portefolio Ω_p of m_p close wind farms in generated. Concerning the set of contracted wind farms Ω_a, it will always consist of the m_a closest wind farms to the geographical barycenter of Ω_p. However to assess evolution of performance in function of the number of contracted agents, for every of the N simulations we will try sizes of Ω_a going from 5 to 70. Because of that the matrices $\mathbb{M}$ and $\mathbb{K}$ will be regenerated for every size m, though they will be the same for all N simulations of one given size m.

Evaluation of OADMM when compared to other methods will be done on the prediction error, sparsity of the estimator and finally on the computation time. Prediction error will be assessed through the means of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) which are defined as following :

$$\begin{aligned} \text{RMSE} &= \left(\frac{1}{T-t_0}\sum_{t=t_0+1}^{T}(y_t - \hat{y}_{t|t-1})^2\right)^{1/2} \\ \text{MAE} &= \frac{1}{T-t_0}\sum_{t=t_0+1}^{T}|y_t - \hat{y}_{t|t-1}| \end{aligned} \tag{II.15}$$

Where t_0 is a time step from which we start measuring the errors. In our case we take $t_0 = T_{\text{batch}} + 1$ since evaluation of batch ADMM must not be done on the learning set, which would bias the final result. The difference of interpretation between RMSE and MAE is not an easy point. Both yield similar results, but RMSE is generally more sensible to outliers than MAE. Since we always have RMSE $>$ MAE, the greater the difference between the two, the higher the number of outliers in estimation is for a given method. Once these prediction errors have been calculated for the three methods, we will calculate the relative improvement of ADMM / OADMM relatively to OLS which allow us to compare efficiently the

methods to one another:

$$\begin{aligned} \text{RMSE}_{\%} &= 100 \times \frac{\text{RMSE}_{\text{OLS}} - \text{RMSE}_{\text{method}}}{\text{RMSE}_{\text{OLS}}} \\ \text{MAE}_{\%} &= 100 \times \frac{\text{MAE}_{\text{OLS}} - \text{MAE}_{\text{method}}}{\text{MAE}_{\text{OLS}}} \end{aligned} \tag{II.16}$$

where "method" is either ADMM or OADMM. This quantity will be positive if the method considered is more performant than OLS, and negative if it is less. Its value will then correspond to the relative percentage of gain / loss relatively to the benchmark method OLS.

Sparsity of OADMM will be assessed through the percentage of coefficients of $\boldsymbol{\beta}$ which are zero and will be plotted over time. The sparsity of the ADMM estimator will also be plotted for comparison. It must be reminded that the contracted agents don't help the central one to improve his forecasts for free, and thus every non-zero coefficient may have a cost.

All the test have been performed on a 64 bytes Intel i7-4810 CPU, with a 2.80 GHz processor.

II.4.1 Tuning of λ, ρ and of the dynamic forgetting factor

Our online ADMM algorithm has 5 parameters to tune: λ and ρ which are involved in the shrinking of the coefficients as well as b, a and c which are defining the behavior of the forgetting factor. However those different coefficients cannot be tuned separately, since the two groups of parameters impinge on each others' effects. Too high values of λ (or too small of ρ) increase sparsity but slow down the convergence speed of the algorithm. Conversly, a high value of b or a low of a which correspond to easy forgetting may cause instabilities which need to be compensated through high values of λ to stabilize the algorithm. However in fact high values of both λ/ρ and b are not usually possible. Indeed, a high value of b means a high amount of forgetting, which leads after a few iterations to a decrease of $\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i$. If the threshold ratio λ/ρ is too high, this will have for consequence an almost sparse estimator $\boldsymbol{\beta}$, and lead to terrible forecasts. This issue is illustrated with C.4 which is in the appendix.
This means that it is not possible to tune the different parameters separately. Ergo one must try all the possible combinations of the parameters to find the one which satisfies him the most. Furthermore a specific tuning must correspond to a specific size m of a grid, which means that tuning must be redone when the size of the grid changes significantly.

The tuning process itself is rather classic. One tries to minimize prediction error (generally in terms of RMSE) on a separate set or through cross-validation to find optimal values of the parameters. However in our case we also try to promote sparsity, which means that the values of the parameters we chose may not be the ones which minimize RMSE but satisfy a certain compromise between sparsity and error minimization.

We therefore tried all possible combination of parameters in a certain range. Each yielded a RMSE value and a sparsity, and we chose the combination of λ, ρ, a, b and c which satisfied us the most. However since hereafter in every test we will try different values of m_a, a completely optimal value cannot be obtained for all the different simulations. We thus chose for the two first test cases hereafter to tune on the generic values $m_p = 15$ and $m_a = 30$, which yield generally good results for all sizes of m_p and m_a. The eventual non-optimality of the parameters may only slightly decrease the quality of results, but as one can see this does not hold back OADMM much. A second tuning is then done for the situation of the system operator ($m = m_p$) since it differs a lot from the cases $m_p = 1$ and $m_p = 15$. Finally for the

test cases $m_p = 1$ and $m_p = 15$ we chose the values $\lambda = 0.01$, $\rho = 1$, $b = 0.05$, $a = 0.1$ and $c = -80$, while for the last test case a good tuning is $\lambda = 9 \times 10^{-3}$, $\rho = 1$, $b = 0.05$, $a = 0.13$ and $c = -80$.

The tuning procedure for batch ADMM was the same, though obviously easier thanks to the presence of only two parameters. Good couples of values are respectively $(\lambda, \rho) = (0.01, 0.9)$ and $(\lambda, \rho) = (8 \times 10^{-3}, 0.9)$. Finally the tuning of the forgetting factor of online least-squares was done through cross validation too and $\mu = 0.9994$ yields the best results in most cases.

II.4.2 Case of a single windfarm ($m_p = 1$)

The situation of this section is the one of an owner of a single farm who contracts a varying number m_a of wind farms to improve his forecasts. The results of the simulation are presented in fig. II.9.

OADMM yields the best results out of the three methods. It improves online least-squares forecast from 6% to 12%, and improves forecast of batch ADMM at least by 2% in terms of RMSE. Similar (though a little inferior) results are obtained for MAE. This detail may mean that OLS has more often outliers than the two other methods of estimation, outliers whose significance is toned down in MAE. Improvements of forecast stagnates for OADMM over $m_a = 30$ contracted farms: this is probably due to the fact that contracted farms are chosen geographically, and that the further the contracted farms are, the less interesting information for forecast they hold. This is directly correlated with the increase of sparsity of the estimator when m_a gets bigger. Indeed, far away farms may not yield interesting information, and therefore the algorithm puts their corresponding coefficients to 0. Sparsity is very small for smaller size of contracted sets Ω_a since most close farms may hold good information for the contracted agent.
One may also see that the sparsity of the batch estimator (plotted in dashed lines) is quite inferior than the one of the online algorithm. Though we are not sure of it, this is probably due to the averaging over the learning set to which batch estimation proceeds. A few farms may be relevant at first and then not at all (or conversly), but batch ADMM can't adapt its estimation over time: thus it averages over the whole learning set. Besides a learning set of size $T_{\text{batch}} = 25,000$ is rather big, and it might be excessive for good batch estimation. These two points might be also the reason why for an excessive number of contracted agents the performance of batch ADMM drops.
Finally one can see that as expected batch ADMM has a calculation time which increases linearly with m, whereas it grows quadratically for the online algorithm. This is an issue, but since after a certain number of contracted agents m_a there is no more improvement of forecast, one may only be content with a limited amount of contracts. This way OADMM would be almost as quick as batch ADMM, and yield better results. In order to be sure what this critical number of contracts is, the central agent could increase the amount of the latter only progressively, and finally stop when he sees that progress becomes very small.

A possibility to have better results with batch ADMM would be through *sliding windows*. At every time step, batch estimation would be reperformed on the T_{batch} last iterations. This would somewhat solve the issue of time variations tracking. However this would increase dramatically the computational cost, and is why this method is hardly used in practice.

If one is interested in the aspect of the different estimations comparatively to the real data, the plot C.5 representing that can be found in Appendix D.

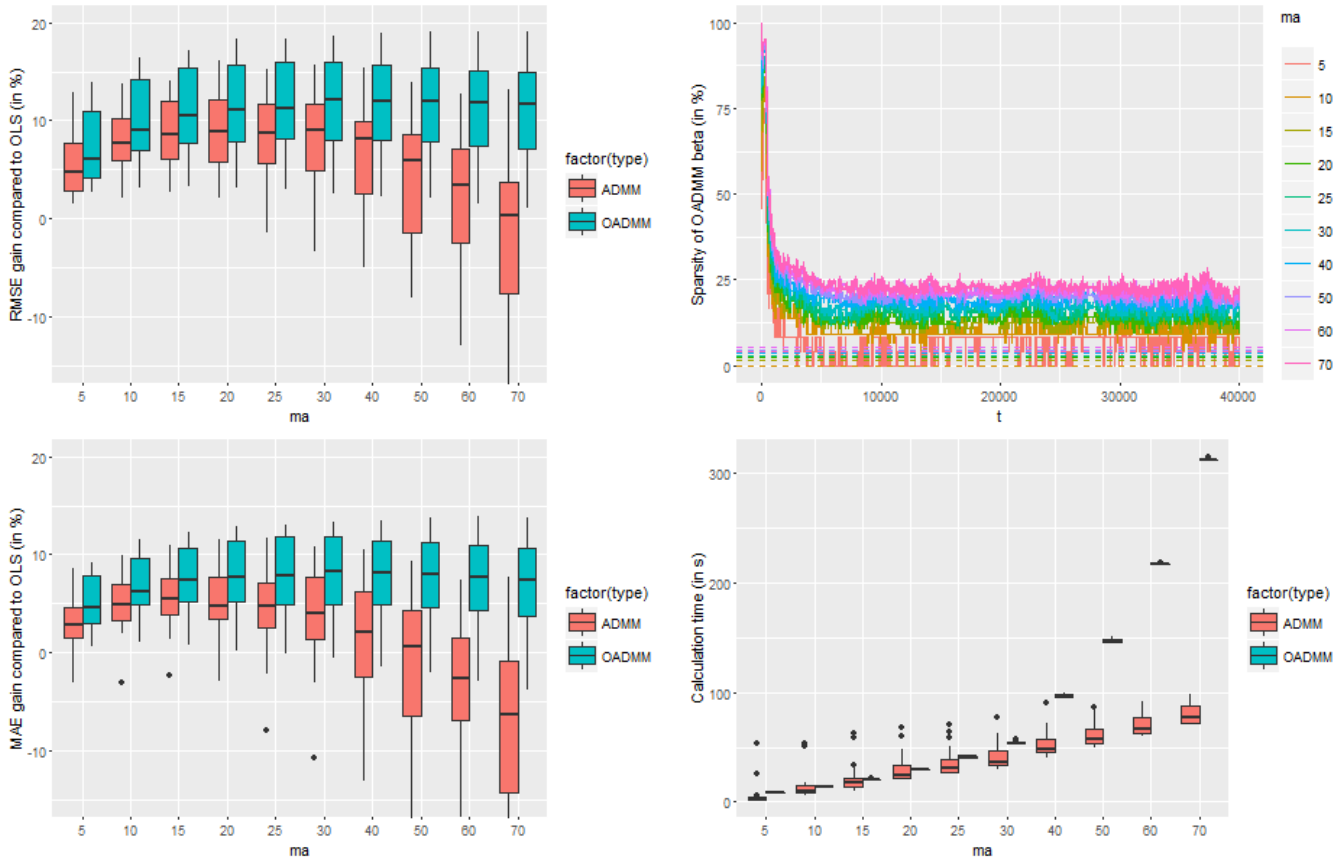

Figure II.9: Comparison of performance of OADMM with ADMM and OLS for single wind farms ($m_p = 1$)

II.4.3 Case of a portefolio of $m_p = 15$ wind farms

The same experiment was performed, but this time for small portefolios each of size $m_p = 15$. Again, OADMM performs better than its two competitors in accuracy. It yields similar results than for the previous part, with improvement between 7 and 12% when compared to OLS and at least 3% compared to batch ADMM in terms of RMSE. One can again see this "maximal progress" which can be achieved and which stagnates after a certain number of contracted farms.
Sparsity this time is a lot higher than for the case of a single wind farm, which seems logical. Indeed, since Ω_p is generated from close wind farms, and thus a lot of interesting information is already contained in the portefolio. Contracting further agents quickly becomes useless, which results in high sparsity of the estimator. One can see in the top-right plot two brutal changes in sparsity at $t \approx 23,000$ and $t \approx 28,000$. This is probably due to the dynamic forgetting factor, which means that at this moment there was a drop of forecasting accuracy.

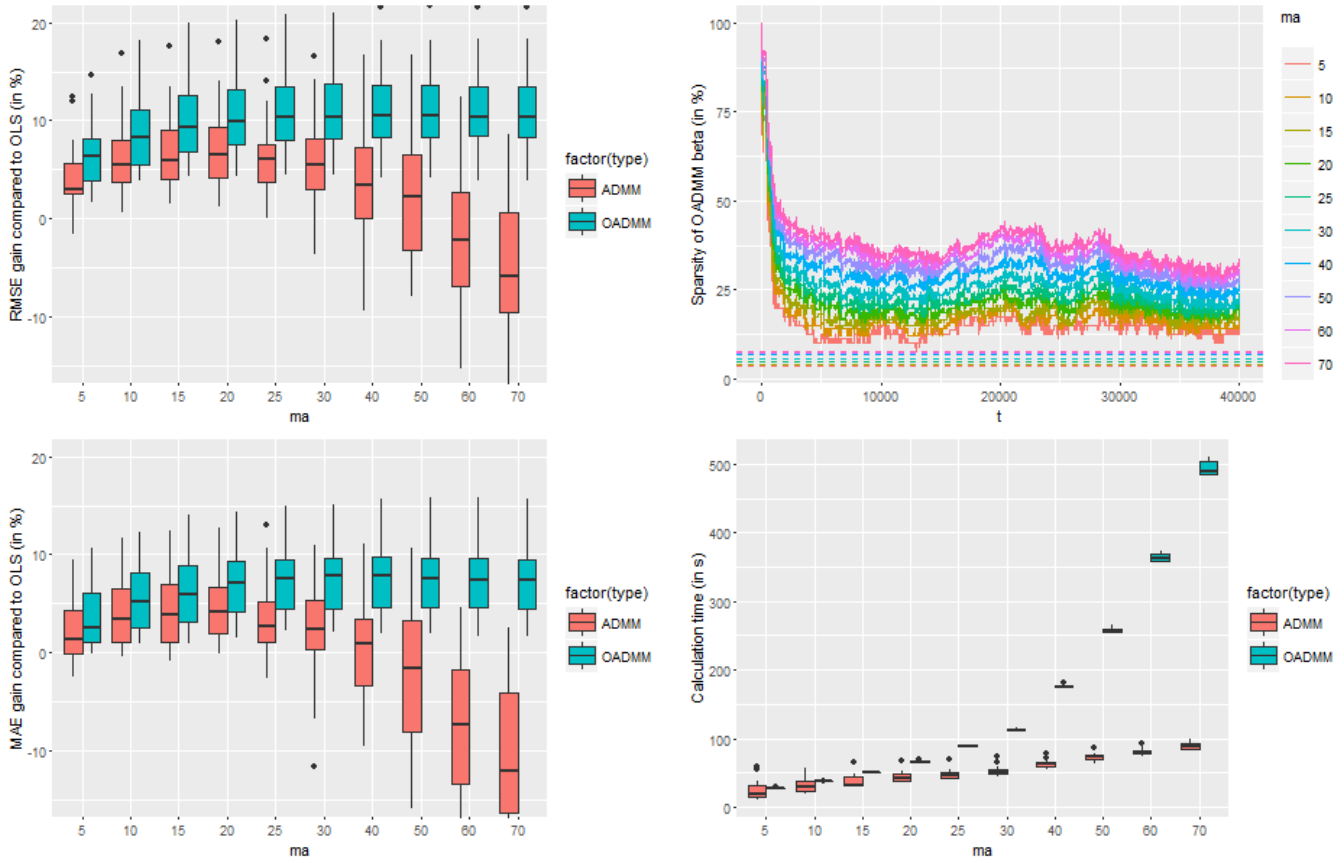

Figure II.10: Comparison of performance of OADMM with ADMM and OLS for portefolios of size $m_p = 15$

II.4.4 Case of the system operator

In this case we consider the case of the system operator, who aims at reducing his exploitation costs through better forecasting of renewable energy production. However this time the procedure of testing is slightly different.

Due to the $\mathcal{O}(m^2)$ of the OADMM algorithm, in the following example we randomly select only $m = 160$ wind farms from the 349 possible of the Danish data set. The resulting data set is then divided in $N = 10$ successive sub-sets of $14,000$ consecutive time steps (*i.e.* the first sub-set will have the observations from $t = 1$ to $14,000$, the second from $t = 14,001$ to $28,000$, and so on). Batch learning will be performed on the $T_{\text{batch}} = 10,000$ first steps of those sub-sets, and applied in a similar fashion as earlier on the next $4,000$ next one. As usual OLS and OADMM will be performed on-the-fly on the $T = 14,000$ time steps. The results obtained are plotted fig. II.11.

OADMM improves OLS estimation by 7% and ADMM one by 11% in terms of RMSE. The results for MAE are as usual similar though a little inferior, due to the lesser weight given to outliers in MAE. In this case the results for ADMM are really surprising, as it has troubles to compete with online methods here. A possible explanation is the following: in the case of the system operator, most farms of the data set have been taken in account. However due to the cleaning of the latter, a certain number of wind farms electricity production measurements have discontinuities in them (due to the removal of corrupted data). Batch ADMM, averaging over a given learning set, may have therefore troubles to find right coefficients due to these discontinuities, which results in poorer predictions.

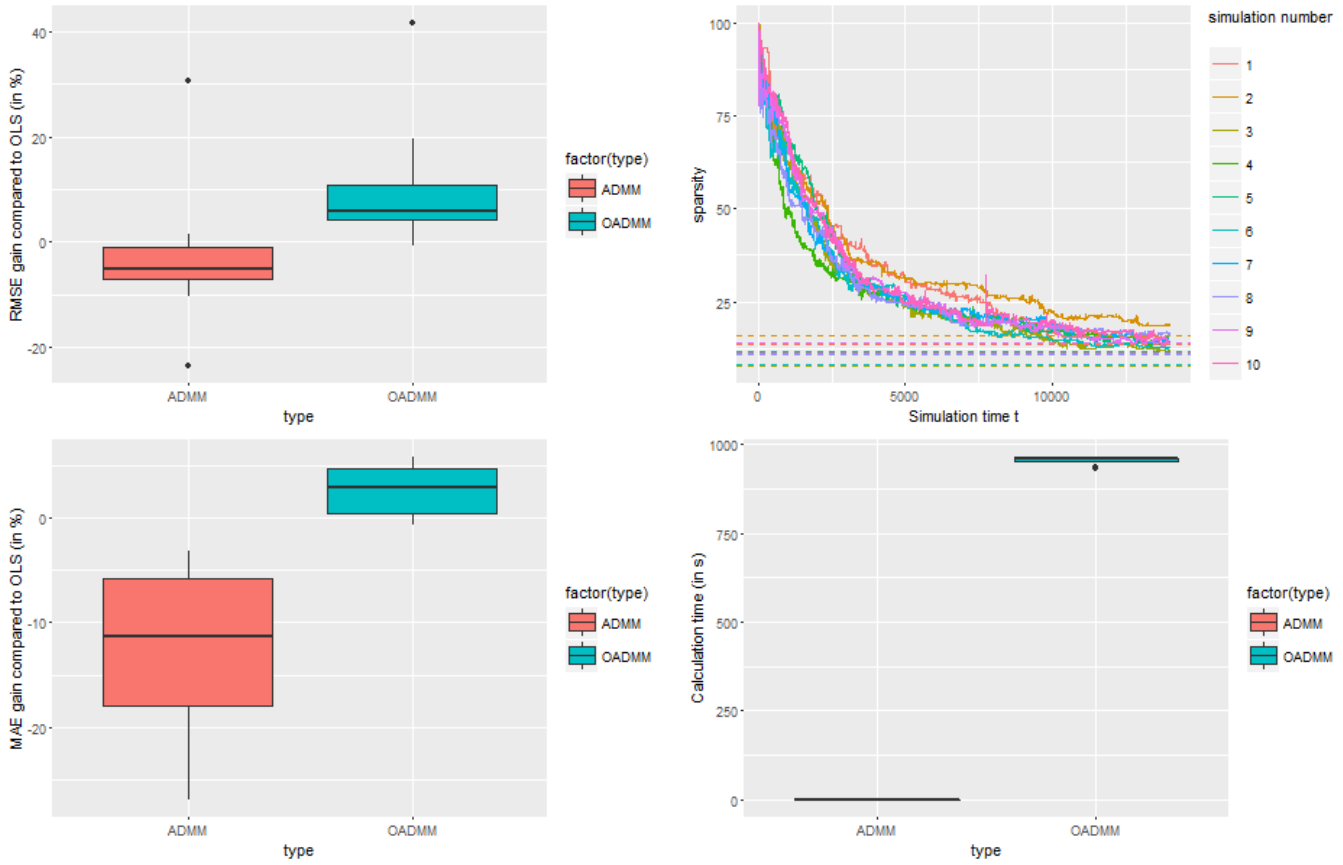

Figure II.11: Comparison of performance of OADMM with ADMM and OLS in the case of the system operator

Finally the top-right picture represents sparsity for each of the $N = 10$ simulations (the time steps represented in x-axis are only relative to a sub-set) over time. As one can see, achieved sparsity stagnates after some time around 20% for OADMM, which is analogous to batch ADMM. While this may seem poor, actually it is not bad at all. Indeed in the case of the system operator, the farms are dispatched all over a country and may thus have very different production from one to another. High sparsity of the estimator is therefore rarely achievable.

II.5 Future development possibilities

While the OADMM algorithm presented in this paper is an overall success, we are planning to develop it further, either to improve forecasting accuracy even more or to perform a different type of estimation than the one performed in this paper.

II.5.1 Improving of the compatibility with the dynamic forgetting factor

By adding the dynamic forgetting factor, one greatly improves the tracking abilities of the online algorithm. However a forgetting factor, even a dynamic one, always represents a risk. As the reader saw it earlier in II.4.1, simultaneously having high values of λ/ρ and b is not possible. Therefore there is a fundamental compromise between sparsity and tracking abilities for OADMM.
For now no solution has been found to that problem. The idea of variants of the lasso penalization

have been tried, such as penalizing comparatively to a batch estimation and not the zero vector (*i.e.* replacing the penalty $\lambda|\boldsymbol{\beta}|$ by a penalty $\lambda|\boldsymbol{\beta} - \boldsymbol{\beta}_0|$, where $\boldsymbol{\beta}_0$ could be for instance a batch estimation). However none of the tried methods have worked for now, and the best way to avoid this problem is to chose lower values of b when tuning. This matter is still being investigated, and we hope a solution will be found.

II.5.2 Quantile regression

In this paper, we made the choice to use a quadratic loss function to perform our regressions. Therefore implicitely we estimated the conditional mean of the average production y_t of the central agent respectively to a given set of regressors, namely the energy productions of the farms of $\Omega_p \cup \Omega_a$ at the previous time steps. This yields very limited possibilities of statistical interpretation, and one may thus be interested in quantile regression instead.

Quantile regression aims at estimating the quantiles of a random variable, conditionally to a given set of regressors. Estimating the quantiles of the average production would give more information about its variability, and therefore would allow a better management of risks linked to forecasts. The online minimization problem of the α-ieth quantile is written:

$$\min_{\boldsymbol{\beta}} \sum_{\tau=\ell+1}^{t-1} \rho_\alpha(\boldsymbol{x}_\tau \boldsymbol{\beta} - y_\tau) + \lambda|\boldsymbol{\beta}| \tag{II.17}$$

where $\rho_\alpha(u) := \Big(\alpha - \mathbb{1}_{u<0}\Big)u$.

Reformulating the problem in ADMM form, the optimality problem becomes:

$$\begin{cases} \min \sum_{\tau=\ell+1}^{t-1} \rho_\alpha(\boldsymbol{x}_\tau \mathbb{M}\boldsymbol{z} - y_\tau) + \lambda|\boldsymbol{\beta}| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0, \ \forall i \end{cases} \tag{II.18}$$

to which the augmented lagragian is

$$\mathcal{L}_\rho = \sum_{\tau=\ell+1}^{t-1} \rho_\alpha(\boldsymbol{x}_\tau \mathbb{M}\boldsymbol{z} - y_\tau) + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| + \frac{\rho}{2} \left\| \boldsymbol{\beta} - \mathbb{M}\boldsymbol{z} + \boldsymbol{u} \right\|^2$$

A $\boldsymbol{\beta}_t^i$ update would be obtained exactly like for the case of a quadratic loss. However the $\boldsymbol{z}$ update becomes problematic, since ρ_α is not differentiable at 0. Even if it is sub-differentiable, calculating the subdifferential would lead to a $\boldsymbol{z}$ update which cannot be performed simply by solving a linear system like II.6. Solutions like linear programming may be used to address this issue, and their compatibility with OADMM will be analyzed soon.

Planning of the internship

- Reading of [6] and of papers of state-of-the-art methods: 3 weeks.
- Programming of batch ADMM and online least-squares and application on synthetic data: 2 weeks.
- Searching for already existing methods of online ADMM: 2 week.
- Developing the OADMM algorithm presented in this paper and testing it on synthetic data: 4 weeks.
- Testing the OADMM algorithm and comparing it with other methods on the Danish data set: 3 weeks.

Conclusion

Therefore this project was overall a success. A (semi) online method using the Alternative Direction Method of Multipliers to proceed to distributed learning has been developed. It yields better results in forecasting than state-of-the-art methods such as batch ADMM for distributed learning or online least-squares for both RMSE and MAE criteria, as well as managing better to obtain sparsity in the estimators. As wanted the presented algorithm protects efficiently the data of the participating actors through two encryption matrices. The use of a dynamic forgetting factor, though difficult to tune, makes the developed algorithm good in learning from time varying data.
The online ADMM algorithm also needs very few exchanges between agents and of very little size, which make its implementation on a real life platform more realistic than its batch counterpart. It thus represents an interesting option for performing good forecasting of wind power production, for both an agent owning a few farms or for the system operator. Finally, even if in this paper we took the example of wind power production forecasting, since the presented method relies only mathematical tools it can be applied for other types of energy as well (solar for instance).

Notwithstanding those positive points, flaws exist in the developed algorithm. The major drawback is that it is not fully recursive. It is even slower than the batch method, and calculational burden was one of the motivations behind the development of an online scheme. While solutions have been investigated to address this issue, such as the introduction of a fusion center, none have yielded good results. It might be possible that developing a completely online scheme which protects information is not possible, in which case somebody who desires a purely recursive algorithm needs to use other methods than ADMM. Furthermore the tuning of the algorithm, with 5 parameters, is not easy, and there is a dilemma between having tracking abilities and achieving sparse estimations, which make of OADMM a complicated forecast method to use.

From a more personal point-of-view, this internship was a great opportunity broaden my knowledge in fields such as optimization and statistical learning, and has conforted me in my future choice of specialization. It was also an opportunity to develop autonomy: in research the goal is often blurry and unclear. It is often after many tries and failures that one finds the right path, something which is not the case in school projects I was used to do until now. This was especially true in this internship: many ideas proved to be dead ends, and some of them even seem ridiculous when I look back at them. This gives a light feeling of loss of time, but also enlarges the feeling of achievement when the results of the here presented OADMM were obtained.

However the work on the latter is not over. With Pierre Pinson we will work on a paper and try to publish the results we obtained either in IEEE or the International Journal of Forecasting, and I hope this will succeed. Furthermore, the possibility of performing quantile regression with OADMM would allow more opportunities of risk management and give more robust estimations, and therefore the compatibility of the presented algorithm with quantile regression will be analyzed soon.

Bibliography

[1] J. Dowell and P. Pinson. Very-short-term probabilistic wind power forecasts by sparse vector autoregression. *IEEE*, Transactions on smart grids, 2015.

[2] L. Cavalcante, R. Bessa, M. Reis, and J. Dowell. Sparse structures for very short-term wind power forecasting. *Wind Energy*, To be published.

[3] P. Pinson. Introducing distributed learning approaches in wind power forecasting. *International Conference on Probabilistic Methods Applied to Power Systems*, 2016.

[4] H. Wang and A. Banerjee. Online alternating direction method. 2012.

[5] T. Suzuki. Dual Averaging and Proximal Gradient Descent for online Alternating direction multiplier method. *Proceedings of the 30rd International Conference on Machine Learning*, 2013.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3, No 1, 2010.

Glossary

ADMM Alternative Direction Method of Multipliers. See [6] for more details.

Batch Learning Learning method performed on a stationary data set.

Central Agent The actor of interest in this paper who tries to improve the forecast of the average production of his portefolio Ω_p. See Portefolio for more information.

Contracted Agents Agents of other wind farms of the grid which have been contracted by the central agent to help him in his forecasts. The set of the contracted agents' wind farm is noted Ω_a and consists of m_a wind farms.

MAE Mean Absolute Error. See II.15 for a definition.

OADMM Online Alternative Direction Method of Multipliers. An online version of Alternative Direction Method of Multipliers. See [6] for more details. (ADMM).

OLS Online least-squares. See Appendix A for more detail.

Online Learning A recursive learning method. Learning is performed at every step t, making it more flexible than batch learning.

Portefolio The set of wind farms of the central agent. It will be noted Ω_p and consists of m_p wind farms.

RMSE Root Mean Squared Error. See II.15 for a definition.

Soft Thresholding The soft thresholding function $\mathcal{S}$ of threshold κ is the function defined as following:

$$\mathcal{S}_\kappa(a) = \begin{cases} a - \kappa & \text{if } a > \kappa \\ a + \kappa & \text{if } a < -\kappa \\ 0 & \text{else} \end{cases}$$

Its definition is generalized to the case of a vector where it is applied coefficient by coefficient.

System operator The operator of the smart grid.

WPP Wind Power Plant.

Appendices

Appendix A

Omitted calculations of the ADMM

A.1 Expression of the fusion center $\overline{z}$

After introducing the fusion center $\overline{\boldsymbol{z}}$ in I.11, one obtains equation I.12 for $\boldsymbol{z}_k^i$. The can be reinjected in the I.11, which yields:

$$\overline{\boldsymbol{z}}_k = \text{argmin}_{\boldsymbol{z}} \frac{1}{2} \|m\overline{\boldsymbol{z}} - \boldsymbol{Y}\|^2 + \frac{\rho}{2} \sum_{i=1}^{m} \left\| \mathbb{X}_i \boldsymbol{\beta}_k^i - \overline{\boldsymbol{z}} - \mathbb{X}_i \boldsymbol{\beta}_k^i + \overline{\boldsymbol{X}\boldsymbol{\beta}}_k - \boldsymbol{u}_{k-1}^i + \overline{\boldsymbol{u}}_{k-1} + \boldsymbol{u}_{k-1}^i \right\|^2$$

Since $\boldsymbol{u}_{k-1}^i$ does not depend of i and thus $\overline{\boldsymbol{u}}_{k-1} \equiv \boldsymbol{u}_{k-1}^i := \boldsymbol{u}_{k-1}$, this gives:

$$\overline{\boldsymbol{z}}_k = \text{argmin}_{\boldsymbol{z}} \frac{1}{2} \|m\overline{\boldsymbol{z}} - \boldsymbol{Y}\|^2 + \frac{m\rho}{2} \left\| \overline{\boldsymbol{z}} - \overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \boldsymbol{u}_{k-1} \right\|^2$$

The zero-gradient condition yields:

$$m \times (m\overline{\boldsymbol{z}} - \boldsymbol{Y}) + m\rho(\overline{\boldsymbol{z}} - \overline{\boldsymbol{X}\boldsymbol{\beta}}_k - \boldsymbol{u}_{k-1}) = 0$$

This finally yields the expected update of $\overline{\boldsymbol{z}}_{k+1}$ given by I.14:

$$\overline{\boldsymbol{z}}_k = \frac{1}{m+\rho}(\boldsymbol{Y} + \rho\overline{\boldsymbol{X}\boldsymbol{\beta}}_k + \rho\boldsymbol{u}_{k-1})$$

A.2 Shooting for calculating the β update

The $\boldsymbol{\beta}_k$ update I.15 still needs to solve an optimality problem. An iterative method has been chosen to do so, and is called shooting. The principle of the method is detailled hereafter. In order to lighten the notational burden, we will omit the indices and exponants k and i and will keep in mind though that what is presented below is the shooting to calculate $\boldsymbol{\beta}_k^i$, the estimated coefficients corresponding to agent i at iteration k. This means that we will note for instance $\mathbb{X}_i$ only $\mathbb{X}$ and $\boldsymbol{Y}_k^i$ as $\boldsymbol{Y}$.

Let $\varphi(\boldsymbol{\beta}) = \|\mathbb{X}\boldsymbol{\beta} - \boldsymbol{Y}\|^2$ and $\psi(\boldsymbol{\beta}) = |\boldsymbol{\beta}|$ be. Let us also define $f = \varphi + \frac{2\lambda}{\rho}$. φ and ψ are both convex, and thus f is convex too. It therefore has a subdifferential $\partial f(\boldsymbol{\beta})$ for every vector $\boldsymbol{\beta}$.

Furthermore, a vector $\boldsymbol{\beta}^*$ is an optimum of f iff $0 \in \partial f(\boldsymbol{\beta}^*)$. Since the subdifferential of a sum is the sum of subdifferential, this optimality conditions yields in our case:

$$\boldsymbol{\beta}^* \in \text{argmin}(f) \Leftrightarrow 0 \in 2\mathbb{X}^\top(\mathbb{X}\boldsymbol{\beta}^* - \boldsymbol{Y}) + \frac{2\lambda}{\rho}\partial\psi(\boldsymbol{\beta}^*)$$

Since ψ is separable in its coordinates, this gives for every coordinate β_j^* of $\boldsymbol{\beta}^*$:

$$0 \in \sum_{t=\ell+1}^{T} \mathbb{X}_{t,j}\Big(\sum_{k=1}^{\ell} \mathbb{X}_{t,k}\beta_k^* - Y_t\Big) + \frac{\lambda}{\rho}\partial\psi(\beta_j^*),\ \forall j \in \{1,2,\dots,\ell\}$$

$$\Leftrightarrow \quad (\boldsymbol{x}_j^\top \boldsymbol{x}_j)\beta_j^* + \underbrace{\sum_t \sum_{k\neq j} \mathbb{X}_{t,j}\mathbb{X}_{t,k}\beta_k^* - \boldsymbol{x}_j^\top \boldsymbol{Y}}_{=S_{0,j}} = -\frac{\lambda}{\rho}\partial\psi(\beta_j^*),\ \forall j \in \{1,2,\dots,\ell\}$$

where $\boldsymbol{x}_j$ represents the j-ieth column of matrix $\mathbb{X}$. Shooting consists in solving the above presented equations sequentially for every coordinate β_j^* of $\boldsymbol{\beta}^*$ until a convergence criterion is met or a maximum number of iterations performed. One solves the above presented equation sequentially by noticing that $\beta_j^* \mapsto \boldsymbol{x}_j^\top \boldsymbol{x}_j \beta_j^* + S_{0,j}$ is affine, and that therefore solving this equation for a given j consists in calculating the intersection between the corresponding affine function and the broken line representing $-\frac{\lambda}{\rho}\partial\psi(\beta_j^*)$. This fact is illustrated in fig. A.1 and yields the updates which are presented in the figure and in the algorithm below. This means that at every iteration p of the shooting algorithm, one obtains an estimate β_p^j of β_j^*.

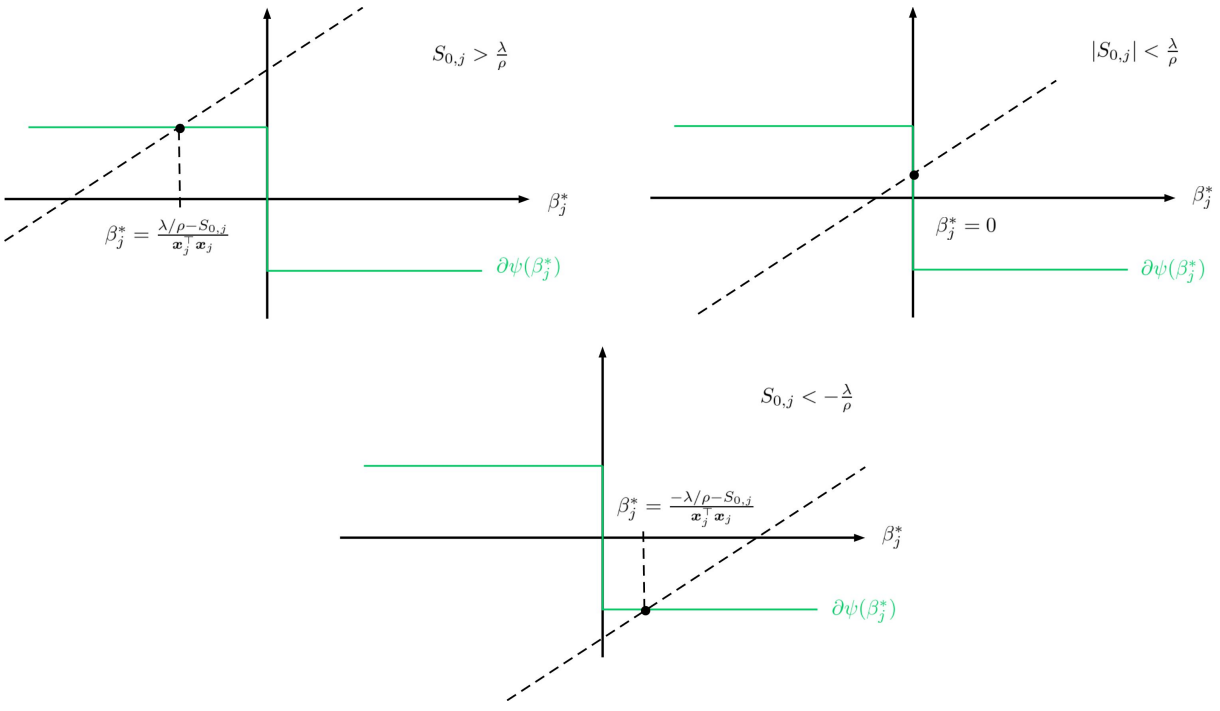

Figure A.1: Principle of shooting at one step p to calculate the approximation β_p^j of β_j^*

In the algorithm $\boldsymbol{\beta}$ may be initialized at the ridge estimator $\Big(\mathbb{X}_i^\top \mathbb{X}_i + \frac{\lambda}{\rho}\text{Id}\Big)^{-1}\mathbb{X}_i^\top \boldsymbol{Y}_{k-1}^i$ since it is often close to $\boldsymbol{\beta}^*$. Therefore the complete shooting algorithm is the following:

Algorithm 3 Shooting algorithm at outer loop k for agent i

Initialize $\boldsymbol{\beta}_0$ at $\left(\mathbb{X}_i^\top \mathbb{X}_i + \frac{\lambda}{\rho}\mathsf{Id}\right)^{-1} \mathbb{X}_i^\top \mathbf{Y}_{k-1}^i$

while $\|\boldsymbol{\beta}_p - \boldsymbol{\beta}_{p-1}\| > \eta$ and $p \leq M_{\mathsf{loop}}$ **do**
  for $j = 1, \ldots, \ell$ **do**

    Calculate $S_{0,j}$

    if $S_{0,j} > \lambda/\rho$ **then**
      $\beta_p^j \leftarrow \dfrac{\lambda/\rho - S_{0,j}}{\boldsymbol{x}_j^\top \boldsymbol{x}_j}$
    else if $S_{0,j} < -\lambda/\rho$ **then**
      $\beta_p^j \leftarrow \dfrac{-\lambda/\rho - S_{0,j}}{\boldsymbol{x}_j^\top \boldsymbol{x}_j}$
    else
      $\beta_p^j \leftarrow 0$
    end if
  end for
  $p \leftarrow p + 1$
end while

return $\boldsymbol{\beta}_p$

Appendix B

Calculations of the online least-squares algorithm

Online least-squares is the most simple form of online learning. It relies on a Newton-Raphson method to obtain a resursive learning scheme, which allows to have very few calculations. Here we directly introduce a constant forgetting factor μ.

Let us consider an autoregressive of lag ℓ model for a time-series $\{y_t\}$ namely $y_t = \boldsymbol{x}_t\boldsymbol{\beta}_t + \varepsilon_t$. The online least-squares problem at step $t+1$ of which $\boldsymbol{\beta}_{t+1}$ is solution is:

$$\min_{\boldsymbol{\beta}} \frac{1}{2} \underbrace{\sum_{\tau=\ell+1}^{t} \mu^{t-\tau}(\boldsymbol{x}_\tau\boldsymbol{\beta} - y_\tau)^2}_{=S_t(\boldsymbol{\beta})}$$

For $\boldsymbol{\beta}_0$ close to $\boldsymbol{\beta}$ one may perform the Taylor development:

$$S_t(\boldsymbol{\beta}) = S_t(\boldsymbol{\beta}_0) + \left\langle \overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}_0), \boldsymbol{\beta} - \boldsymbol{\beta}_0 \right\rangle + \left\langle H_t(\boldsymbol{\beta}_0) \cdot (\boldsymbol{\beta} - \boldsymbol{\beta}_0), \boldsymbol{\beta} - \boldsymbol{\beta}_0 \right\rangle$$

Where $H_t(\boldsymbol{\beta}_0)$ is the hessian matrix of S_t evaluated in $\boldsymbol{\beta}_0$. By taking the gradient of this equation with respect to $\boldsymbol{\beta}$, one gets:

$$\overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}) = \overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}_0) + H_t(\boldsymbol{\beta}_0)(\boldsymbol{\beta} - \boldsymbol{\beta}_0)$$

For $\boldsymbol{\beta} = \boldsymbol{\beta}_{t+1}$ and $\boldsymbol{\beta}_0 = \boldsymbol{\beta}_t$ (which are supposed close), the optimality of $\boldsymbol{\beta}_{t+1}$ makes that the hand-left term of the equation is zero. This yields $\vec{0} = \overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}_t) + H_t(\boldsymbol{\beta}_0)(\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t)$. We will furthermore make the supposition that the hessian is non-singular. In practice one adds a sufficient weight on its diagonal in order to make this hypothesis valid. Therefore $\boldsymbol{\beta}_{t+1}$ is given by the equation:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - \Big(H_t(\boldsymbol{\beta}_0)\Big)^{-1} \overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}_t)$$

One has $H_t(\boldsymbol{\beta}_0) = \sum_{\tau=\ell+1}^{t} \mu^{t-\tau} \boldsymbol{x}_\tau^\top \boldsymbol{x}_\tau$ and is thus independent of the vector in which it is evaluated. Furthermore since $S_t(\boldsymbol{\beta}) = S_{t-1}(\boldsymbol{\beta}) + \frac{1}{2}(\boldsymbol{x}_t\boldsymbol{\beta} - y_t)^2$, derivation with respect to $\boldsymbol{\beta}$ and evaluating the resulting expression for $\boldsymbol{\beta} = \boldsymbol{\beta}_t$ yields:

$$\overrightarrow{\nabla}_{\boldsymbol{\beta}} S_t(\boldsymbol{\beta}_t) = \overrightarrow{\nabla}_{\boldsymbol{\beta}} S_{t-1}(\boldsymbol{\beta}_t) + \boldsymbol{x}_t^\top(\boldsymbol{x}_t\boldsymbol{\beta} - y_t)$$

Since $\boldsymbol{\beta}_t$ is the solution to the optimality problem at step t, $\overrightarrow{\nabla}_{\beta} S_{t-1}(\boldsymbol{\beta}_t) = \vec{0}$. One has therefore the recursive online least-squares algorithm:

$$\begin{cases} \boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t - H_t^{-1} \boldsymbol{x}_t^\top (\boldsymbol{x}_t \boldsymbol{\beta}_t - y_t) \\ H_t = \mu\, H_{t-1} + \boldsymbol{x}_t^\top \boldsymbol{x}_t \end{cases} \tag{B.1}$$

Appendix C

Omitted calculations for the online ADMM scheme

C.1 Detailled calculations of the β_t^i update

At step t, the regression vector $\boldsymbol{\beta}_t^i$ for agent i is obtained by minimizing the function $\mathcal{L}_{\beta^i} = \lambda|\boldsymbol{\beta}^i| + \frac{\rho}{2}\left\|\boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}_{t-1}^i + \boldsymbol{u}_{t-1}^i\right\|^2$ with respect to $\boldsymbol{\beta}^i$. Hence $\boldsymbol{\beta}_t^i$ is a minimum of this objective function iff $0 \in \partial\mathcal{L}_{\beta^i}(\boldsymbol{\beta}_t^i)$. If ψ again denotes the ℓ_1 norm $|\cdot|$, we have the equivalences:

$$\begin{aligned} 0 \in \partial\mathcal{L}_{\beta^i}(\boldsymbol{\beta}_t^i) \quad &\Leftrightarrow \quad 0 \in \frac{\lambda}{\rho}\partial\psi(\boldsymbol{\beta}_t^i) + \boldsymbol{\beta}_t^i - \mathbb{M}_i \boldsymbol{z}_{t-1}^i + \boldsymbol{u}_{t-1}^i \\ &\Leftrightarrow \quad \boldsymbol{\beta}_t^i + \frac{\lambda}{\rho}\partial\psi(\boldsymbol{\beta}_t^i) = \mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i \end{aligned}$$

Similarly to the proof for the shooting method from Appendix A, since the ℓ_1 norm is seperable in its coordinates one obtains the scalar equivalent of the latter equation for every coefficient j. Ergo calculating the coefficients of $\boldsymbol{\beta}_t^i$ consists in finding the intersection between the broken lines representing the components of $\boldsymbol{\beta}_t^i \mapsto \boldsymbol{\beta}_t^i + \frac{\lambda}{\rho}\partial\psi(\boldsymbol{\beta}_t^i)$ and the horizontal lines having for intercepts the coefficients of $\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i$. Graphical representation given by fig. C.1 makes it clear that finally $\boldsymbol{\beta}_t^i$ is given by:

$$\begin{cases} \boldsymbol{\beta}_t^i = \mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i - \dfrac{\lambda}{\rho} & \text{if} \quad \mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i > \dfrac{\lambda}{\rho} \\ \boldsymbol{\beta}_t^i = \mathbb{M}_i \boldsymbol{z}_{t-1}^i + \boldsymbol{u}_{t-1}^i - \dfrac{\lambda}{\rho} & \text{if} \quad \mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i < -\dfrac{\lambda}{\rho} \\ \boldsymbol{\beta}_t^i = 0 \quad \text{if} \quad |\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i| < 0 \end{cases}$$

Which is precisely $\boldsymbol{\beta}_t^i = \mathcal{S}_{\lambda/\rho}(\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i)$, with $\mathcal{S}_{\lambda/\rho}$ being the vectorial soft-thresholding operator.

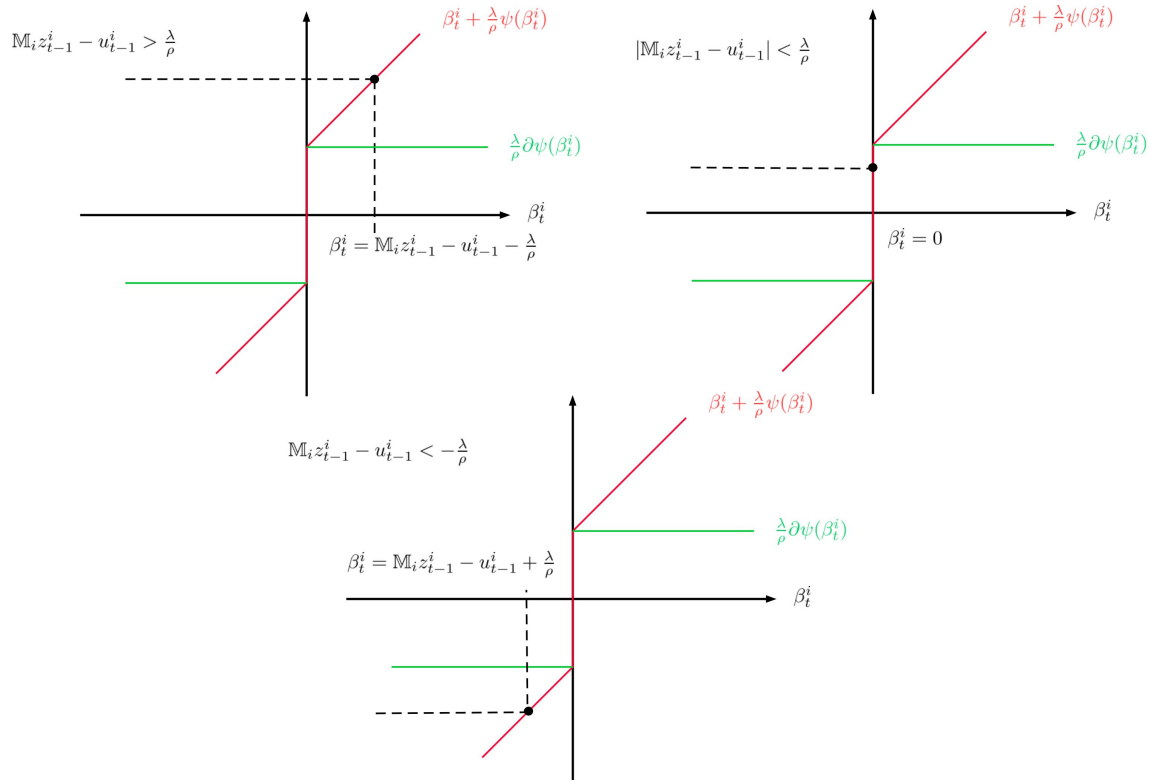

Figure C.1: Soft-thresholding operation for the calculation of a coefficient of $\boldsymbol{\beta}_t^i$

C.2 Proof of the OADMM algorithmic scheme with dynamic forgetting factor

Adding a dynamic forgetting factor to reduce inertia of the learning process consists in reformulating the optimality problem as following at step t:

$$\begin{cases} \min \dfrac{1}{2} \displaystyle\sum_{\tau=\ell+1}^{t-1} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\sum_{i=1}^{m} \boldsymbol{x}_\tau^i \mathbb{M}_i \boldsymbol{z}^i - y_\tau)^2 + \lambda \sum_{i=1}^{m} |\boldsymbol{\beta}^i| \\ \text{s.t.} \quad \boldsymbol{\beta}^i - \mathbb{M}_i \boldsymbol{z}^i = 0, \; \forall i \in \{1, 2, \ldots, m\} \end{cases}$$

Since the calculation of $\boldsymbol{\beta}_t^i$ does not need the two left terms of the objective function, it remains the same as without forgetting factor.

However derivation with respect to $\boldsymbol{z}$ yields:

$$\begin{aligned} & \sum_{\tau=\ell+1}^{t-1} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M} \boldsymbol{z} - y_\tau) - \rho \mathbb{M}^\top (\boldsymbol{\beta}_t - \mathbb{M}\boldsymbol{z} + \boldsymbol{u}_{t-1}) = 0 \\ \Leftrightarrow \quad & \underbrace{\Big(\sum_{\tau=\ell+1}^{t-1} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M}) + \rho \mathbb{M}^\top \mathbb{M} \Big)}_{:= \mathbb{H}_{t-1}} \boldsymbol{z} = \underbrace{\sum_{\tau=\ell+1}^{t-1} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top y_\tau + \rho(\boldsymbol{\beta}_t + \boldsymbol{u}_{t-1})}_{:= \boldsymbol{P}_{t-1}} \end{aligned} \tag{C.1}$$

Since for $\tau = t-1$ the product $\prod_{k=\tau+1}^{t-1} \mu_k = 1$ (it is empty), one has:

$$\begin{cases} \mathbb{H}_{t-1} = \sum_{\tau=\ell+1}^{t-2} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M}) + (\boldsymbol{x}_{t-1} \mathbb{M})^\top (\boldsymbol{x}_{t-1} \mathbb{M}) \\ \boldsymbol{P}_{t-1} = \sum_{\tau=\ell+1}^{t-2} \Big(\prod_{k=\tau+1}^{t-1} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top y_\tau + (\boldsymbol{x}_{t-1} \mathbb{M})^\top y_{t-1} \end{cases}$$

$$\Leftrightarrow \begin{cases} \mathbb{H}_{t-1} = \mu_{t-1} \underbrace{\sum_{\tau=\ell+1}^{t-2} \Big(\prod_{k=\tau+1}^{t-2} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top (\boldsymbol{x}_\tau \mathbb{M})}_{\mathbb{H}_{t-2}} + (\boldsymbol{x}_{t-1} \mathbb{M})^\top (\boldsymbol{x}_{t-1} \mathbb{M}) \\ \boldsymbol{P}_{t-1} = \mu_{t-1} \underbrace{\sum_{\tau=\ell+1}^{t-2} \Big(\prod_{k=\tau+1}^{t-2} \mu_k \Big) (\boldsymbol{x}_\tau \mathbb{M})^\top y_\tau}_{=\boldsymbol{P}_{t-2}} + (\boldsymbol{x}_{t-1} \mathbb{M})^\top y_{t-1} \end{cases}$$

This proves the updates of the OADMM algorithm in the case of a dynamic forgetting factor:

$$\begin{cases} \Big(\mathbb{H}_{t-1} + \rho \mathbb{M}^\top \mathbb{M} \Big) \boldsymbol{z}_t = \boldsymbol{P}_{t-1} + \rho(\boldsymbol{\beta}_t + \boldsymbol{u}_{t-1}) \\ \mathbb{H}_t = \mu_t \mathbb{H}_{t-1} + (\boldsymbol{x}_t \mathbb{M})^\top (\boldsymbol{x}_t \mathbb{M}) \\ \boldsymbol{P}_t = \mu_t \boldsymbol{P}_{t-1} + (\boldsymbol{x}_t \mathbb{M})^\top y_t \end{cases}$$

C.3 Additional plots for the test cases and the case study

C.3.1 Bi-weekly structural breaks for the gaussian data test case

The context is the one of II.3.2, where OADMM was applied on a gaussian generated data with coefficients to estimate $\boldsymbol{\beta}$ having structural breaks. Here we take the case of a bi-weekly change, which corresponds to $T_{\text{period}} = 1344$ steps. The forgetting parameter b was a bit increased when compared to the case with only one single jump. As such, we set $b = 0.04$ to have an effective memory of 25 samples. The values of a and c remain unchanged. As earlier, we repeat the generation of the design matrix $\mathbb{X}$ and the response vector $\boldsymbol{Y}$ $N = 20$ times.

As one can see, again the dynamic forgetting factor enters in action shortly after the structural breaks happened and helps the estimators to converge again quickly.

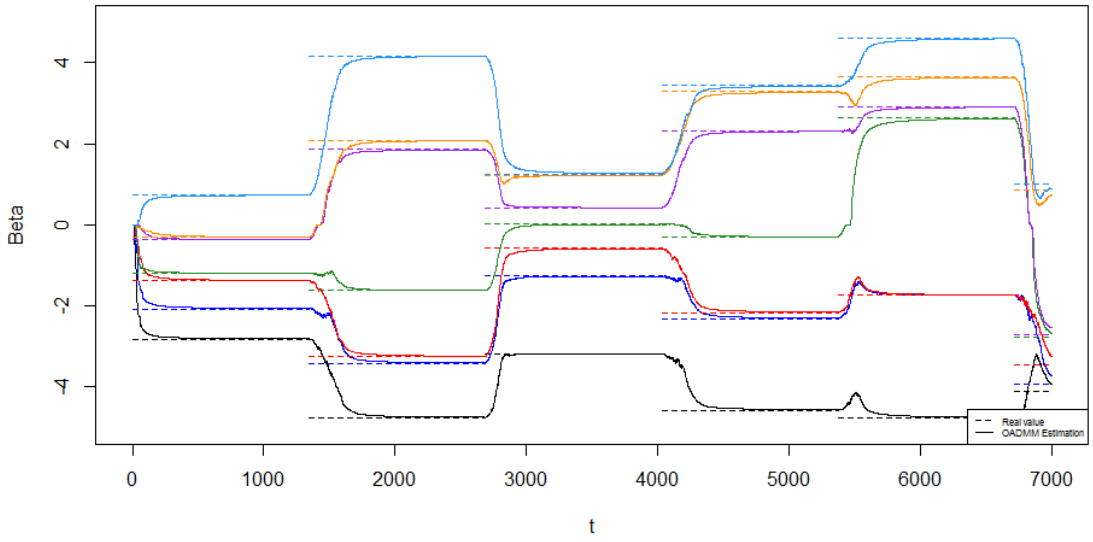

Figure C.2: Average estimation of the non-zero coefficients of $\boldsymbol{\beta}_t$, with seasonally happening structural breaks

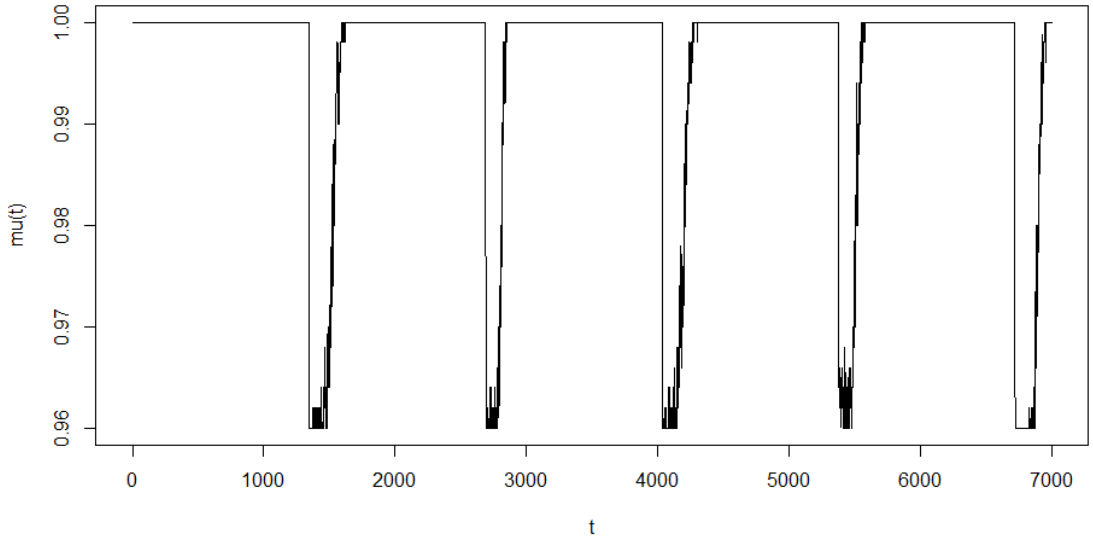

Figure C.3: Average value over the $N = 20$ simulations of the dynamic forgetting factor μ_t

C.3.2 Issues in estimation for b and λ/ρ simultaneously set at high values

In order to illustrate the problem which happens when b and λ/ρ are simultaneously set at high values, we take again the synthetic data of II.3.2 with structural breaks. As usual estimation will be performed on $N = 20$ data sets and then averaged. We set $b = 0.1$ which corresponds to a memory of 10 samples, $a = 0.9$ again and $c = -800$. As one can see with C.4, after managing initially to estimate the right values of the non-zero coefficients of $\boldsymbol{\beta}$, after the structural break OADMM's estimation collapses. This is due to the excessive amount of forgetting: it leads to a low value of $\mathbb{M}_i \boldsymbol{z}_{t-1}^i - \boldsymbol{u}_{t-1}^i$, and finally to a $\hat{\boldsymbol{\beta}}_t^i$

which is almost entirely sparse. The estimation cannot go back to regular values because the prediction error has increased since the structural break, and does not go down. As such the dynamic forgetting factor remains active, and thus the algorithm cannot learn efficiently from new incoming data.

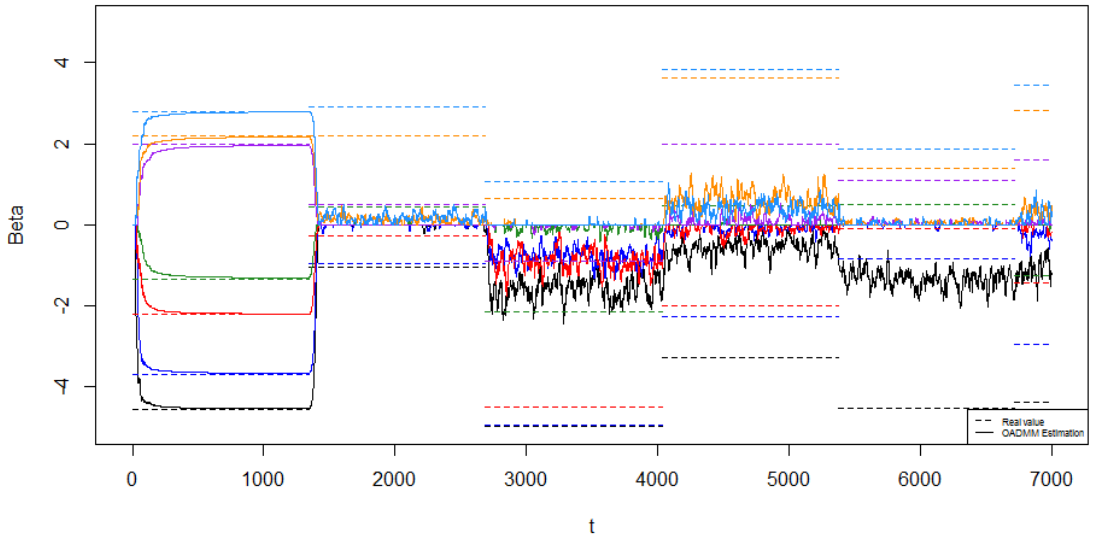

Figure C.4: Unstable estimation of the non-zero coefficients of $\boldsymbol{\beta}_t$ due to an excessive b and threshold λ/ρ

C.3.3 Overlapping of real data y_t and its different estimations $\hat{y}_t$ for the Danish data set

Fig. C.5 represents a typical case of overlapping of the real data y_t and its one step-ahead estimation $\hat{y}_t$, for all three compared methods for $m_p = 15$ and $m_a = 20$. One can see that the estimated data fits very well the real data, although it is lagging behind the real data. This is normal and is very common in estimation: it is due to the fact that when brutal variations happen, it may take a few additional steps (due to the autoregressive nature) to make a major change on the estimation.

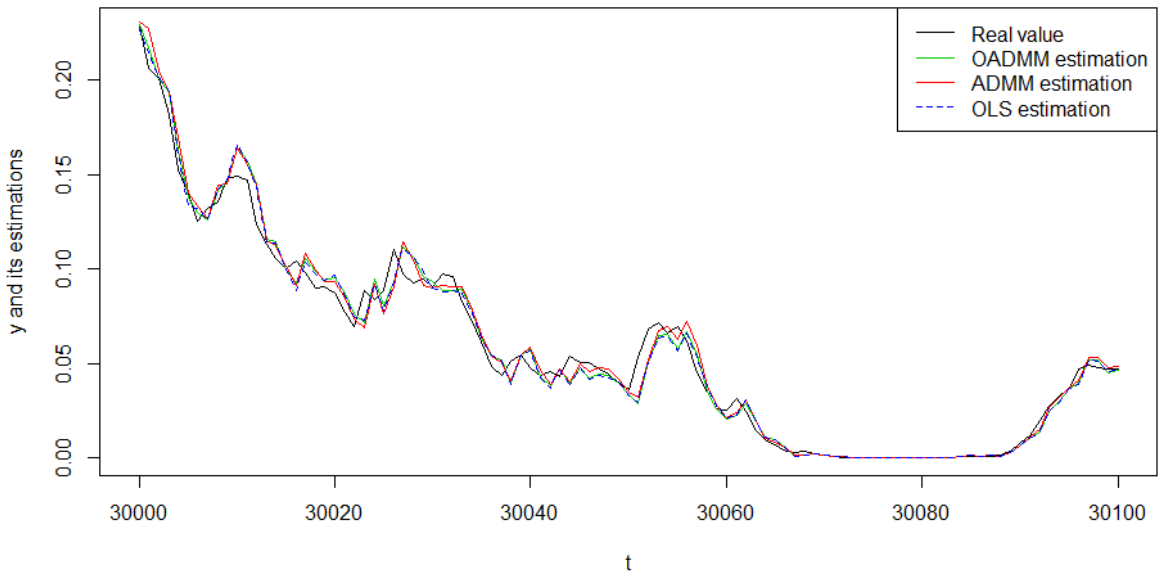

Figure C.5: Overlapping of real data $\{y_t\}$ and its estimates $\{\hat{y}_t\}$ for the three different methods, for $m_p = 15$ and $m_a = 20$