Online Convex Optimization for Time Series Prediction*

Research Report: Internship Summer 2018

Saad Christophe Ackerer Damien†

August 31, 2018

Abstract

This report is a description of the project I realized during my summer internship at Swissquote Group Holding SA in Gland, Switzerland as a quantitative researcher intern. In this project, I worked on a novel approach of time series prediction based on online learning and convex optimization. I am going to show the efficiency and reliability of some algorithms along with their capability of replacing traditional and usually used techniques. In particular, I am going to study a new approach of univariate and multivariate ARMA models based on online convex optimization. [Haz16] [OAS13] [CXF17].

1 Motivation

A time series is a sequence of signals taken at successive equally spaced points in time. Given some time series, predicting the next values according to the previous ones can be very useful in the field of data analysis. Many works have been done on this purpose, the **AutoRegressive Moving Average models** (ARMA) are often used for time series prediction. They assume that each new signal is a noisy linear combination of the last k observations and q last independent noise terms.

$$X_t = \sum_{i=1}^{k} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t$$

where $\alpha \in \mathbb{R}^k$ is the autoregressive (AR) coefficient and $\beta \in \mathbb{R}^q$ is the moving-average (MA) coefficient. There are many techniques for estimating these parameters in order to make the best ARMA prediction. The most commonly used : the **Maximum-Likelihood Estimation**. This estimation technique operates in maximizing the (log)likelihood function. There are many disadvantages of using this method as shown in Table 1.

Maximum-Likelihood Estimation	Online Learning approach
• **Huge compuational cost** • **Strong assumptions on noise terms (independence and identically distributed Gaussian distribution)**	• **Very low compuational cost** • **Approach more general, capacity of coping with a wider range of time series** • **Noise terms arbitrarily generated** • **Handle any type of loss functions**

Table 1: Disadvantages of Maxiumum-Likelihood estimation and advantages of an Online Learning approach

In which situations is it better to use online convex optimization algorithms and make better predictions than the traditional maximum-likelihood estimation?

*Quantitative Asset Management, Swissquote Bank. Email: qam@swissquote.ch
†Internship supervisor

2 Introduction to online convex optimization

In this section, we give a introduction to the concept of online convex optimization(OCO) along with an example.

2.1 Basic Definitions

Here are some basic definitions needed to understand online convex optimization.

Convex Optimization : It is a subfield of optimization that studies the problem of minimizing **convex functions** over **convex sets**.

Online Learning : It a method of machine learning in which data becomes available in **a sequential order** and is used to **update our best predictor** for future data at each step.

2.2 The Online Convex Optimization model

We explain now the idea behind the Online Convex Optimization model along with an Example.

Online convex optimization is defined in a game-theoretic framework, it can be seen as a structured repeated game where:

1. At each iteration t, an online player chooses an element γ^t from a convex bounded set of decisions $\kappa \in \mathbb{R}^n$ without knowing the outcome of his choice.
2. After commiting his decision, the adversary chooses a convex loss function $l_t : \kappa \mapsto \mathbb{R}$ from a bounded family of loss functions.
3. The player then suffer loss $l_t(\gamma^t)$ and adds this loss to his total sum of losses $L = \sum_{i=1}^{T} l_i(\gamma^i)$

Our goal is to minimize the sum of losses over a number T of iterations. This can be achieved by trying to minimize the convex loss function over the set at each iteration t in order to make better choices at $t+1$.

A good OCO algorithm is one with a regret (defined below) that grows sublinearly as a function of T, this implies that on average the algorithm is performing as the best strategy we can have in hindsight. In other words,

$$R_T = \sup_{\{l_1,\dots,l_T\}} \left(\sum_{t=1}^{T} l_t(\widetilde{\gamma}^t) - \min_{\gamma\in\kappa} \sum_{t=1}^{T} l_t(\gamma) \right) = o(T)$$

where $\min_{\gamma\in\kappa} \sum_{t=1}^{T} l_t(\gamma)$ is the total loss of the best decisions we could have made with the best strategy.
To evaluate the performance of an algorithm, we look at an upper bound of its worst case regret.

Example 1 (The weighted majority (WM) algorithm)**.** *Let us imagine a game where an online player needs to make one of two choices A and B, he has the assistance of N advisers. This situation can be modeled as a choice from the set κ of advisors with $|\kappa| = N$. The player has to choose at iteration t the best advice.*
One method is to attribute at each iteration t, to each advisor i a weight $w_t(i)$ which will define the quality of his advice.
We then group the advisors in two subsets $\kappa_A, \kappa_B \subseteq \kappa$, the advisors that chose respectively A and B.
For choices A and B we give weights $W_t(A) = \sum_{a\in\kappa_A} w_t(a)$ and $W_t(B) = \sum_{b\in\kappa_B} w_t(b)$

See Algorithm 1

Algorithm 1 WM algorithm

Initialization $w_1(i) = 1$, $i = 1, ..., N$ and Total Loss $L = 0$
for $t = 1$ **to** T
  Predict $\widetilde{c} \leftarrow \begin{cases} \text{A} & W_t(A) > W_t(B) \\ \text{B} & \text{otherwise.} \end{cases}$
  Observe c and suffer loss $l_t(\widetilde{c})$, $L = L + l_t(\widetilde{c})$
  Update $w_{t+1}(i) \leftarrow \begin{cases} w_i(t) & w_i(t) = c \\ (1-\epsilon)w_i(t) & \text{otherwise for some } \epsilon > 0 \end{cases}$
  Update $W_{t+1}(A)$ and $W_{t+1}(B)$
end for
return L

3 Online convex optimization algorithms

In this section, we present two OCO algorithms, the Online Gradient Descent and the Online Newton step algorithms.

3.1 Settings

We give some setting and notations for the rest of this report.

$\kappa \subseteq \mathbb{R}^n$	convex set
l_t	loss function revealed at time t
T	total number of iterations (run)
$\{\gamma^t\}_{t=1}^T$	time series
$\{\widetilde{\gamma}^t\}_{t=1}^T$	predictions series
$\prod_\kappa(y)$	Euclidean projection of y onto κ $\quad \underset{x\in\kappa}{\arg\min}\ \|\|y-x\|\|_2$
$\prod_\kappa^A(y)$	Euclidean projection of y onto κ in the norm induced by a matrix A $\quad \underset{x\in\kappa}{\arg\min}\ (y-x)^\top A(y-x)$
η	learning rate, step sizes we make at each iteration towards the minimum of the convex function
G	upper bound of the norm of the gradient $\quad \|\|\nabla l_t(x)\|\| \leqslant G$
D	upper bound of the diameter of set κ $\quad \underset{x,y\in\kappa}{\sup}\ \|\|x-y\|\|_2 \leqslant D$
λ	the exp-concavity parameter of the loss functions $\{l_t\}_{t=1}^T$, this means that $\forall t, e^{-\lambda l_t}$ is concave

Remark. $\prod_\kappa(y) = \prod_\kappa^I(y)$

3.2 Online Gradient Descent (OGD)

Online Gradient Descent is the first OCO algorithm we are going to present.

Gradient descent, represented in Figure 1, is an iterative optimization method which consists of iteratively moving the current point in the direction of the function's gradient to reach its extrema. In our case, this function will be the loss function and we will try to reach its minimum.
Online gradient descent, Figure 2, is a **first order method**, this means that it only takes into consideration the first order derivative of the convex function (its gradient). Algorithm 2 describes how it operates.

Algorithm 2 Online Gradient Descent

Input convex set κ, T, $\gamma_1 \in \kappa$, step sizes $\{\eta_t\}$
for $t = 1$ **to** T
  Predict $\widetilde{\gamma^t}$
  Observe γ^t and $l_t(\widetilde{\gamma^t})$
  Project and **Update:** $\quad \widetilde{\gamma_{t+1}} = \prod_\kappa(\gamma^t - \eta_t \nabla l_t)$
end for

OGD grantees a regret bound of

$$R_t \leqslant \frac{3}{2} GD\sqrt{T} = \mathcal{O}(GD\sqrt{T})$$

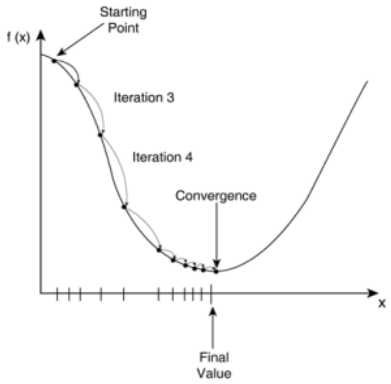


Figure 1: Gradient descent, taking steps in the direction of the gradient to reach the minimum of the convex function

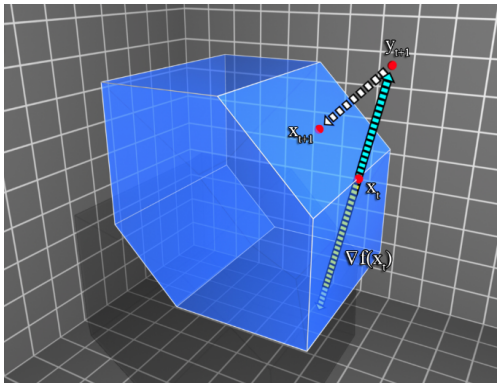


Figure 2: Online gradient descent, moving in the direction of the gradient and then projecting back to the set

3.3 Online Newton Step (ONS)

Next, we look at the second algorithm, the Online Newton Step.

Newton-Raphson is an iterative method for finding the roots of a differentiable function by moving in a direction defined by the function and its derivative

$$x_{k+1} = x_k - (\nabla f(x_k))^{-1} f(x_k)$$

In our case, we are interested in finding an extrema, so the steps will be taken in the direction defined by the first and second order derivatives (the gradient and the Hessian)

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k)$$

Since the Hessian is too expensive to compute, this OCO algorithm will be based on the **quasi-newton method** which uses an approximation of the hessian.

Online newton step is a second order method which will indirectly take into consideration the second order derivative of the function (the Hessian) by moving in the direction of the product between the inverse of a matrix A_t **related to the hessian** and the gradient. Algorithm 3 describes how it operates.

Algorithm 3 Online Newton Step

1: **Input** convex set κ, T, $\gamma_1 \in \kappa$, step size η, an initial matrix $A_0 = \frac{1}{\eta^2 D^2} I$
2: **for** $t = 1$ **to** T
3: **Predict** $\widetilde{\gamma^t}$
4: **Observe** γ^t and $l_t(\widetilde{\gamma^t})$
5: **Update** $A_t = A_{t-1} + \nabla l_t \nabla l_t^\top$
6: **Project** and **Update:** $\widetilde{\gamma_{t+1}} = \prod_\kappa^{A_t}(\gamma^t - \frac{1}{\eta} A^{-1} \nabla l_t)$
7: **end for**

ONS grantees a regret bound of

$$R_t \leqslant 5(\frac{1}{\lambda} + GD) n log(T) = \mathcal{O}((GD + \frac{1}{\lambda}) \log(T))$$

Remark. *There is an efficient way of computing A_t^{-1} at each iteration using Sherman-Morrison formula*

$$(A+uu^\top)^{-1} = A^{-1} - \frac{A^{-1}uu^\top A^{-1}}{1+u^\top A^{-1}u}$$

where $A \in \mathbb{R}^{n\times n}$ and $u \in \mathbb{R}^n$

4 Application to univariate ARMA model

These machine learning techniques are applicable to the ARMA model in order to make an estimation of the parameters and make good predictions. But since the sequence of noise terms are never revealed at any time, we cannot use online convex optimization over the space of AR and MA coefficients (α, β). We use an improper learning approch by approximating the ARMA(p,q) model by an AR(M) where $M = m+k$ for some $m \in \mathbb{N}$

$$\widetilde{X_t} = \sum_{i=1}^{M} \gamma_i^t X_{t-i}$$

Remark. *In all this project, only the squared loss function is considered $l_t(\widetilde{x}) = (x-\widetilde{x})^2$, the parameters are calculated according to the squared loss.*

4.1 Settings

These are the settings of the online learning models:

1. $\kappa = \{\gamma \in \mathbb{R}^M : |\gamma_i| \leqslant c \quad i = 1,...,M\}$ such that $|\alpha_i| \leqslant c \quad i = 1,...,k$
2. $l_t(\widetilde{X}) = (X_t - \widetilde{X}_t)^2 = (X_t - \sum_{i=1}^M \gamma_i^t X_{t-i})^2 = l_t(\gamma^t)$
3. $\nabla l_t(\gamma^t) = \begin{pmatrix} \frac{\partial}{\partial\gamma_1}\left((X_t - \sum_{i=1}^M \gamma_i^t X_{t-i})^2\right) \\ \frac{\partial}{\partial\gamma_2}\left((X_t - \sum_{i=1}^M \gamma_i^t X_{t-i})^2\right) \\ \vdots \\ \frac{\partial}{\partial\gamma_M}\left((X_t - \sum_{i=1}^M \gamma_i^t X_{t-i})^2\right) \end{pmatrix} = -2(X_t - \widetilde{X}_t)\begin{pmatrix} X_{t-1} \\ X_{t-2} \\ \vdots \\ X_{t-M} \end{pmatrix}$
4. $G = 2c\sqrt{M}.(X_{max})^2$
5. $D = 2c\sqrt{M} = 2\max_{x\in\kappa} ||x||_2 \geqslant \sup_{x,y\in\kappa} ||x-y||_2$
6. $\lambda = \frac{1}{M}$

4.2 ARMA OGD

We apply the Online Gradient Descent on the ARMA model as shown in Algorithm 4.

Algorithm 4 ARMA online gradient descent

1: **Input** Horizon M, first step size $\eta_1 = \frac{D}{G}$, set constraint c, $\gamma^1 \in \kappa$, T, in-sample data $\{X\}$
2: **for** $t = 1$ **to** T
3: **Update** $\eta_t \leftarrow \frac{1}{\sqrt{t}}\eta_1$
4: **Predict** $\widetilde{X}_t = \sum_{i=1}^M \gamma_i^t X_{t-i}$
5: **Observe** X_t and suffer loss $l_t(\gamma^t)$
6: **Project** and **Update:** $\gamma^{t+1} \leftarrow \prod_\kappa(\gamma^t - \eta_t \nabla l_t)$
7: **end for**

4.3 ARMA ONS

We apply the Online Newton Step algorithm on the ARMA model as shown in Algorithm 5.

Algorithm 5 ARMA online newton step

Input Horizon M, step size $\eta = \frac{1}{2}\min\{\lambda, \frac{1}{4GD}\}$, set constraint c, $\gamma^1 \in \kappa$, initial $(M \times M)$ matrix $A_0 = \frac{1}{\eta^2 D^2} I_M$, T, in-sample data $\{X\}$
for $t = 1$ **to** T
  Predict $\widetilde{X}_t = \sum_{i=1}^{M} \gamma_i^t X_{t-i}$
  Observe X_t and suffer loss $l_t(\gamma^t)$
  Update $A_t \leftarrow A_{t-1} + \nabla l_t \nabla l_t^\top$
  Project and **Update:** $\gamma^{t+1} \leftarrow \prod_{\kappa}^{A_t}(\gamma^t - \frac{1}{\eta} A_t^{-1} \nabla l_t)$
end for

4.4 Experiments

We now test these algorithms on synthetic and real data, we compare their predictions with those done by the ARMA MLE by using the root-mean-square-error.

$$RMSE_t = \sqrt{\frac{\sum_{i=1}^{t}(X_i - \widetilde{X_i})}{t}}$$

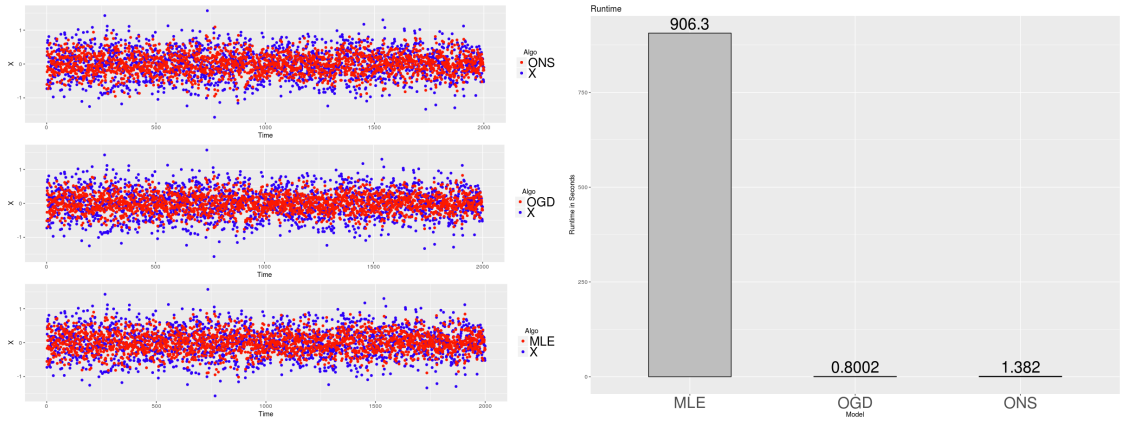
We set $M = 10$ and $c = 1$ for all experiments.

a Setting 1: Sanity Check

We generate data using the R function arima.sim with gaussian noise normally distributed $\mathcal{N}(0, \sigma^2)$. The parameters used are in Table 2 and the results are shown in Figure 3 and Figure 4.

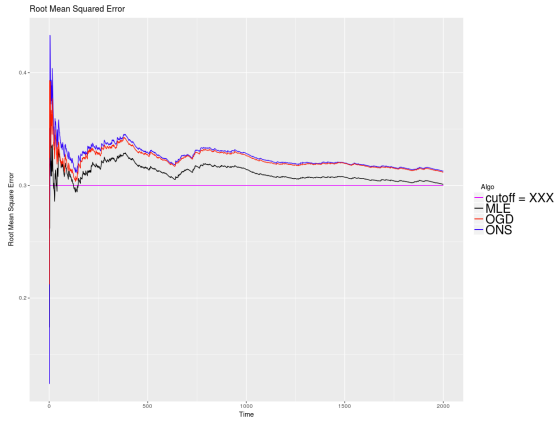
α	$(0.6, -0.5, 0.4, -0.4, 0.3)$
β	$(0.3, -0.2)$
σ	0.3
run	2000
training run (ONS and OGD)	2000

Table 2: Settings 1



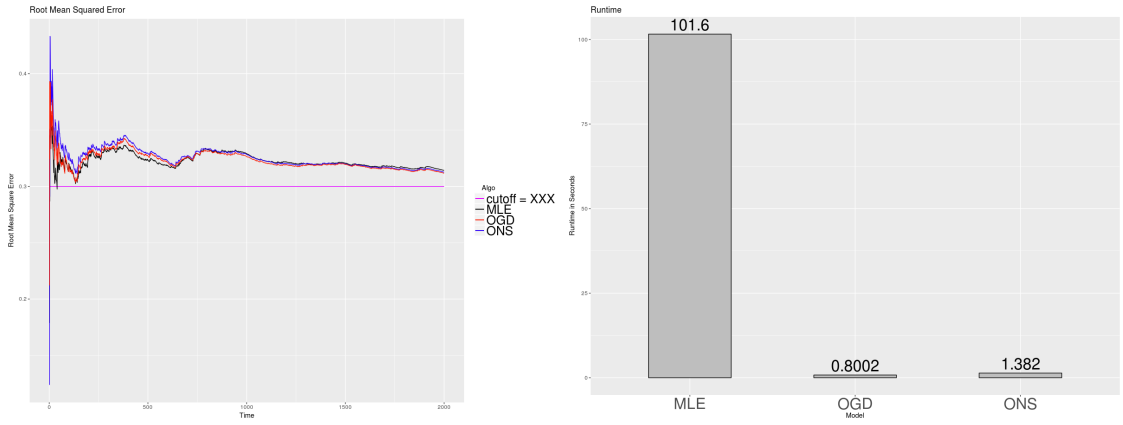
(a) Predictions

(b) Runtime (in s)



(c) RMSE

Figure 3: Setting 1 results with MLE having a growing window for predictions



(a) RMSE

(b) Runtime (in s)

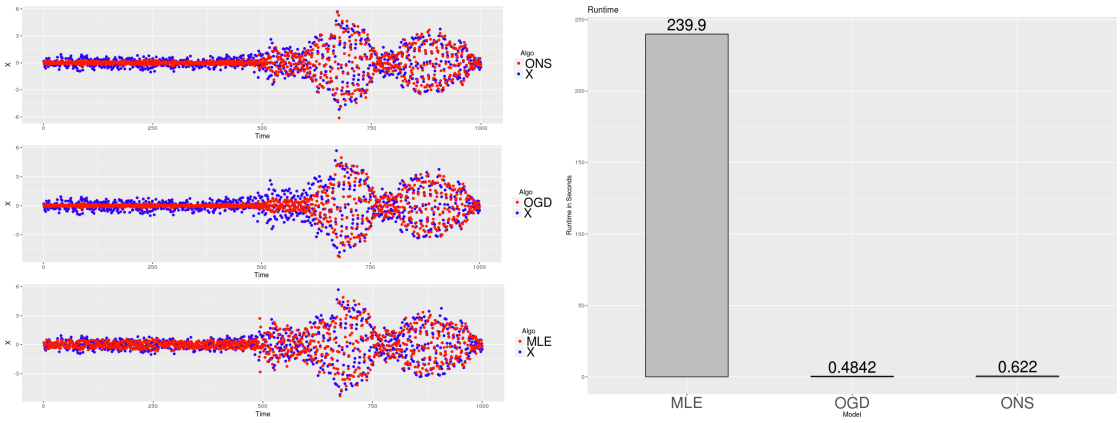
Figure 4: Setting 1 results with MLE having a fixed window of 252 for predictions

b Setting 2: Abrupt change

We combine two series of synthetic data using arima.sim with different parameters to check how the algorithms react to an abrupt change. The noise are normally distributed $\mathcal{N}(0,\sigma^2)$. The parameters used are in Table 3 and the results are shown in Figure 5 and Figure 6.

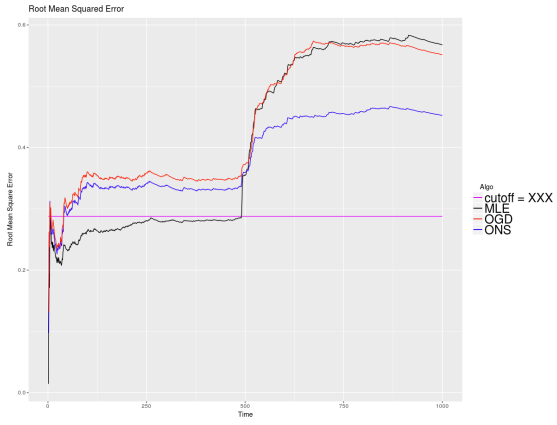
α_1	$(0.6, -0.5, 0.4, -0.4, 0.3)$
α_2	$(-0.4, -0.1, 0.65, -0.1, 0.1)$
β_1	$(0.3, -0.2)$
β_2	$(-0.7, 0.4)$
σ_1	0.3
σ_2	0.1
run	1000
training run (ONS and OGD)	1000

Table 3: Settings 2



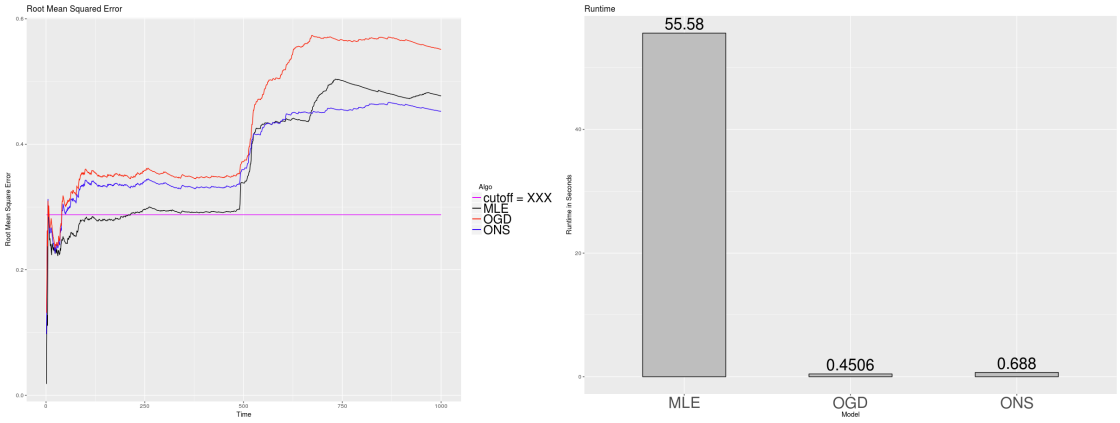
(a) Predictions

(b) Runtime (in s)



(c) RMSE

Figure 5: Setting 2 results with MLE having a growing window for predictions



(a) RMSE

(b) Runtime (in s)

Figure 6: Setting 2 results with MLE having a fixed window of 252 for predictions

c Setting 3: Real Data, Apple Daily Volumes

We are now interested and testing these algorithm on real data, apple daily volumes (Figure 7). The parameters used are in Table 4 and the results are shown in Figure 8.

run	1000
training run (ONS and OGD)	1200
MLE window size	252

Table 4: Settings 3

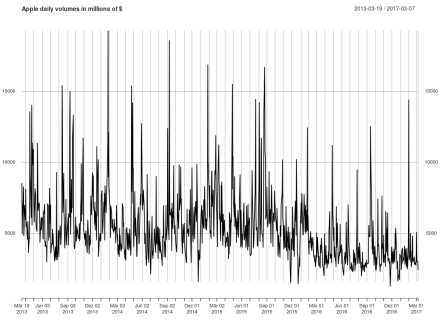
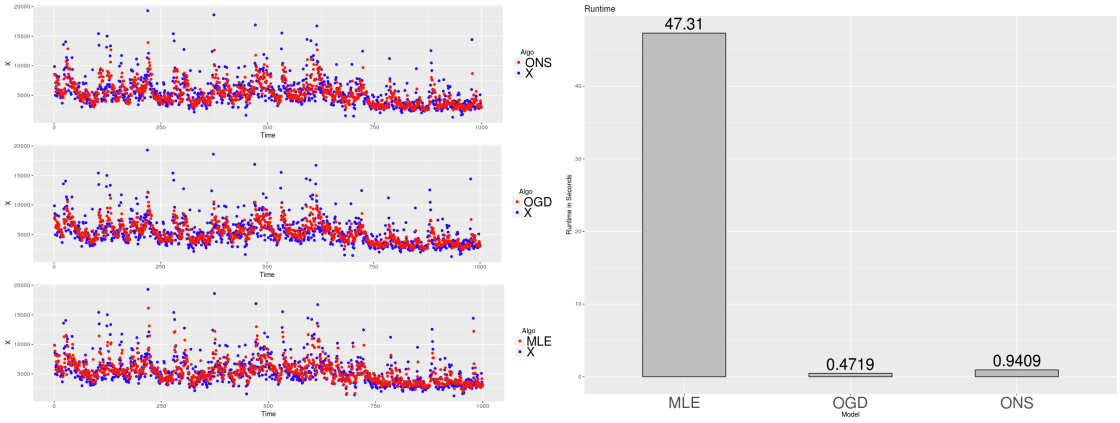
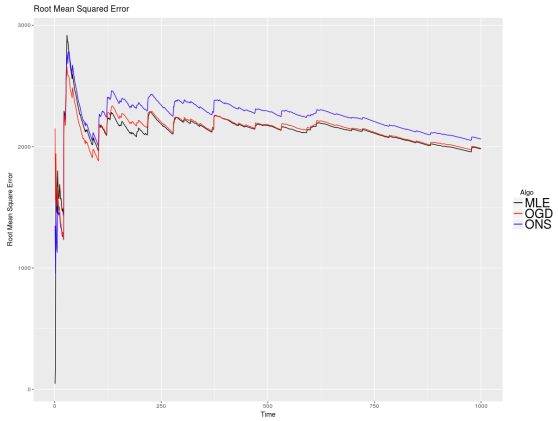


Figure 7: Apple Daily Volumes in millions of $



(a) Predictions

(b) Runtime (in s)



(c) RMSE

Figure 8: Setting 3 results

d Setting 4: Real Data, half-hourly electricity demand in England and Wales

Another example of real data time series prediction, half-hourly electricity demand in England and Wales (Figure 9). The parameters used are in Table 5 and the results are shown in Figure 10 and in Figure 11.

run	1500
training run (ONS and OGD)	2000

Table 5: Settings 4

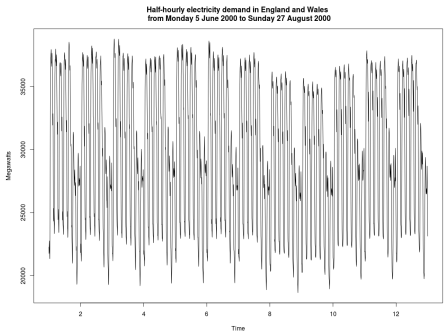
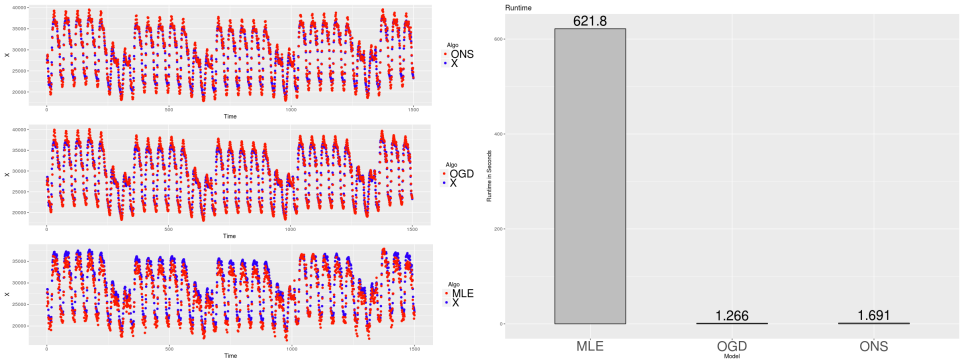
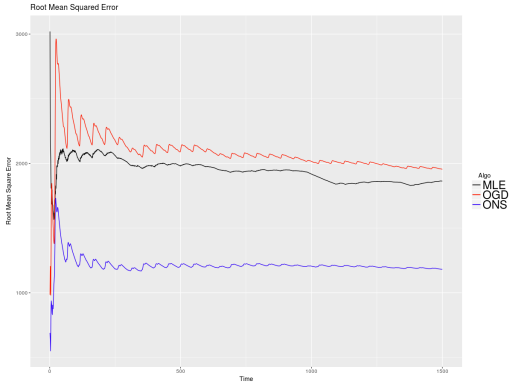


Figure 9: Half-hourly electricity demand in England and Wales in Megawatts

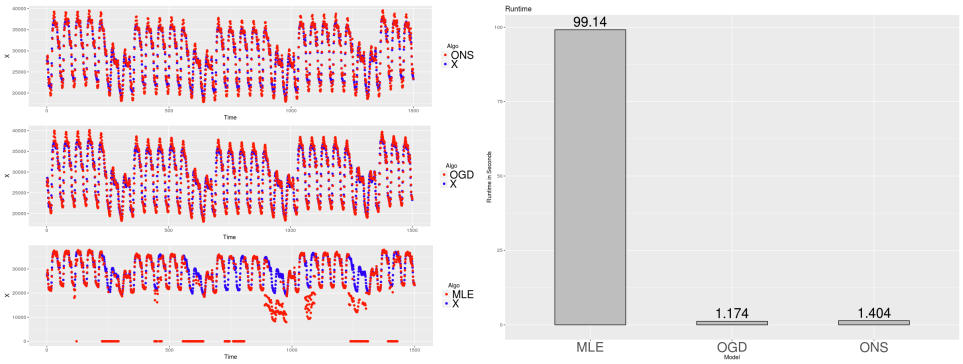


(a) Predictions (b) Runtime (in s)

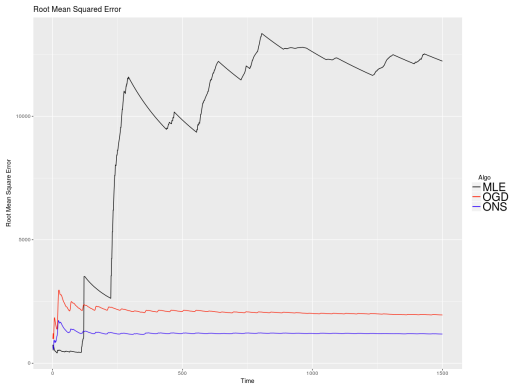


(c) RMSE

Figure 10: Setting 4 results with MLE having a growing window for predictions



(a) Predictions (b) Runtime (in s)



(c) RMSE

Figure 11: Setting 4 results with MLE having a fixed window of 252 for predictions

5 Application to multivariate ARMA process

In this section, we apply Online Convex Optimization to multivariate ARMA models. We test our algorithms on both synthetic and real data.

5.1 Some notations

Here are some notations and definitions that we will use in the rest of the report.

$A[i,j]$	is the i^{th} row and j^{th} column element of an $(m \times n)$ matrix A
$\|\|A\|\|_2$	Frobenius norm of a matrix A $\quad \|\|A\|\|_2 = \sqrt{\sum_i \sum_j A[i,j]^2}$
$\|\|A\|\|_W$	Frobenius norm of a matrix A induced by a matrix W $\quad \|\|A\|\|_W = \|\|W^{\frac{1}{2}}(A-X)W^{-\frac{1}{2}}\|\|_2$
$\|\|A\|\|_*$	nuclear norm of a matrix A $\quad \|\|A\|\|_* = \sum_{i=1}^{\min(m,n)} \sigma_i(A)$
$\|\|A\|\|_{\max}$	norm max of a matrix A $\quad \|\|A\|\|_{\max} = \max_{i,j} \|A[i,j]\|$
$\prod_\kappa^W(A)$	Euclidean projection of A onto κ in the norm induced by a matrix W $\quad \underset{X\in\kappa}{\arg\min} \|\|A-X\|\|_W$
$\prod_\kappa(A)$	Euclidean projection of A onto κ $\quad \underset{X\in\kappa}{\arg\min} \|\|A-X\|\|_2$
$\prod_{\mathscr{B}(*,\rho)}(A)$	Projection of A onto the nuclear norm ball of radius ρ $Procedure$: 1-Make a singular values decomposition of $A = U\Sigma V^\top$ 2-Project greatest $\min(m,n)$ singular values onto an $l1$-ball of radius ρ (Algorithm 10) 3-Make a new diagonal matrix Σ^* with diagonal the projection of the singular values 4-Return the matrix $U\Sigma^* V^\top$ The returned matrix may have a lower rank, this is what we call low rank approximation.

Low rank approximation: It is a technique of machine learning where a matrix of rank r is approximated to another matrix of lower rank. The best approximation is measured by minimizing a predefined cost function. This technique is used to simplify the model.

Definition 1 (Cointegration)**.** *Let x be a collection of n time series. We say that x_t is cointegrated of order (d,p) if each of its components are integrated of order d and there exists a linear combination $\mu_1 x_{1,t} + \mu_2 x_{2,t} + ... + \mu_n x_{n,t}$ which is of order $(d-p)$, the vector μ is called the cointegration vector.*

$$x_t \sim CI(d,p) \iff x_{1,t}, x_{2,t}, ...x_{n,t} \sim I(d) \ \wedge \ \exists_{\mu\neq 0} \ \mu^\top x_t \sim I(d-p)$$

In general, we have $x_{i,t} \sim I(1)$ and $\Delta x_t = x_t - x_{t-1} \sim I(0)$ (stationary), in this case, the variables follow a common trend.

5.2 The VARMA process (Stationary Time Series)

If we wish to forecast a **stationary** series not only based upon its own past realizations, but additionally taking realizations of $n-1$ other stationary series into account, then we can model the series as **Vector AutoRegressive Moving Average model** VARMA(k,q). It can be represented by the following equation:

$$\widetilde{x_t} = \sum_{i=1}^{k} \Phi_i x_{t-i} + \sum_{i=1}^{q} \Theta_i \epsilon_{t-i} + \epsilon_t$$

where $x_t, \epsilon_t \in \mathbb{R}^n$ and $\Phi_i, \Theta_i \in \mathbb{R}^{n\times n}$ are the coefficients matrices which introduce cross-dependencies between the series.
Each prediction will be an n-dimensional vector containing the forecast of each time series.
As realized for the univariate ARMA process, we approximate the VARMA(k,q) to a VAR(M) process:

$$\widetilde{x_t} = \sum_{i=1}^{M} \Gamma_i x_{t-i}$$

a Settings

1. $\kappa = \left\{\Gamma_i \in \mathbb{R}^{n\times n} : ||\Gamma_i||_{\max} \leqslant 1 \quad i = 1,...,M\right\} \quad ||A||_{\max} \leqslant 1 \Longleftrightarrow \max_{i,j} |A[i,j]| \leqslant 1$
2. Let $\psi = \{\Gamma_1, \Gamma_2, ..., \Gamma_M\}$
3. $l_t(\widetilde{x}_t) = ||x_t - \widetilde{x}_t||_2^2 = (x_{1,t} - \widetilde{x}_{1,t})^2 + ... + (x_{n,t} - \widetilde{x}_{n,t})^2 = (x_{1,t} - \sum_{i=1}^{M} \Gamma_i[1,]x_{t-i})^2 + ... + (x_{n,t} - \sum_{i=1}^{M} \Gamma_i[n,]x_{t-i})^2$
4. $\nabla_{\Gamma_i} l_t = \frac{\partial l_t}{\partial \Gamma_i} = \frac{\partial (x_{1,t} - \widetilde{x}_{1,t})^2}{\partial \Gamma_i} + ... + \frac{\partial (x_{n,t} - \widetilde{x}_{n,t})^2}{\partial \Gamma_i} = -2(x_{1,t} - \widetilde{x}_{1,t}) \frac{\partial \widetilde{x}_{1,t}}{\partial \Gamma_i} - ... - 2(x_{n,t} - \widetilde{x}_{n,t}) \frac{\partial \widetilde{x}_{n,t}}{\partial \Gamma_i}$
$= -2(x_{1,t} - \widetilde{x}_{1,t}) \begin{pmatrix} \frac{\partial \Gamma_i[1,]x_{t-i}}{\partial \Gamma_i[1,1]} & \frac{\partial \Gamma_i[1,]x_{t-i}}{\partial \Gamma_i[1,2]} & \cdots \\ \frac{\partial \Gamma_i[1,]x_{t-i}}{\partial \Gamma_i[2,1]} & \cdots & \\ \vdots & & \end{pmatrix} - ... - 2(x_{n,t} - \widetilde{x}_{n,t}) \begin{pmatrix} \frac{\partial \Gamma_i[n,]x_{t-i}}{\partial \Gamma_i[1,1]} & \frac{\partial \Gamma_i[n,]x_{t-i}}{\partial \Gamma_i[1,2]} & \cdots \\ \frac{\partial \Gamma_i[n,]x_{t-i}}{\partial \Gamma_i[2,1]} & \cdots & \\ \vdots & & \end{pmatrix}$
Since $\Gamma_i[j,]x_{t-i} = \Gamma_i[j,1]x_{1,t-i} + \Gamma_i[j,2]x_{2,t-i} + ... + \Gamma_i[j,n]x_{n,t-i}$
$= -2(x_{1,t} - \widetilde{x}_{1,t}) \begin{pmatrix} x_{1,t-i} & x_{2,t-i} & \cdots & x_{n,t-i} \\ 0 & 0 & \cdots & \\ 0 & \cdots & & \\ \vdots & & & \end{pmatrix} - ... - 2(x_{1,t} - \widetilde{x}_{1,t}) \begin{pmatrix} 0 & 0 & \cdots & \\ 0 & \cdots & & \\ \vdots & & & \\ x_{1,t-i} & x_{2,t-i} & \cdots & x_{n,t-i} \end{pmatrix}$
$= -2 \begin{pmatrix} (x_{1,t} - \widetilde{x}_{1,t})x_{t-i}^\top \\ (x_{2,t} - \widetilde{x}_{2,t})x_{t-i}^\top \\ \vdots \\ (x_{n,t} - \widetilde{x}_{n,t})x_{t-i}^\top \end{pmatrix}$
5. $G = 2n.(X_{max})^2$ where $X_{max} = \max_i x_{i,t}$
6. $D = 2n = 2\max_{X\in\kappa} ||X||_2 \geqslant \sup_{X,Y\in\kappa} ||X - Y||_2$
7. $\lambda = \frac{1}{n}$

b VARMA OGD

Online Gradient Descent for multivariate ARMA is described in Algorithm 6.

Algorithm 6 VARMA online gradient descent

Input Horizon M, first step size $\eta_1 = \frac{D}{G}$, $\psi^1 \in \kappa$, T, in-sample data $\{x\}$
for $t = 1$ **to** T
  Update $\eta_t \leftarrow \frac{1}{\sqrt{t}}\eta_1$
  Predict $\widetilde{x}_t = \sum_{i=1}^{M} \Gamma_i^t x_{t-i}$
  Observe x_t and suffer loss $l_t(\psi^t)$
  Project and **Update:** $\Gamma_i^{t+1} \leftarrow \prod_\kappa(\Gamma_i^t - \eta_t \nabla_{\Gamma_i} l_t)$
end for

c VARMA ONS

Online Newton Step for multivariate ARMA is described in Algorithm 7

Algorithm 7 VARMA online newton step

Input Horizon M, step size $\eta = \frac{1}{2}\min\left\{\lambda, \frac{1}{4GD}\right\}$, $\psi^1 \in \kappa$, initial list of M $(n \times n)$ matrices $A_i = \frac{1}{\eta^2 D^2} I_n$, T, in-sample data $\{x\}$
for $t = 1$ **to** T
  Predict $\widetilde{x}_t = \sum_{i=1}^{M} \Gamma_i^t x_{t-i}$
  Observe x_t and suffer loss $l_t(\psi^t)$
  Update $A_i^t \leftarrow A_i^{t-1} + \nabla_{\Gamma_i} l_t \nabla_{\Gamma_i} l_t^\top$
  Project and **Update:** $\Gamma_i^{t+1} \leftarrow \prod_{\kappa}^{A_i^t}(\Gamma_i^t - \frac{1}{\eta}(A_i^t)^{-1}\nabla_{\Gamma_i} l_t)$
end for

5.3 The EC-VARMA process (Nonstationary Cointegrated Time Series)

The EC-VARMA model can be defined by the following equation (error-correction VARMA):

$$\Delta x_t = \Pi x_{t-1} + \sum_{i=1}^{k-1} \Gamma_i \Delta x_{t-i} + \sum_{i=1}^{q} \Theta_i \epsilon_{t-i} + \epsilon_t$$

where $\Pi = -I + \sum_{i=1}^{k} \Phi_i$ is low rank and $\Gamma_j = -(\Phi_{j+1} + ... + \Phi_k)$ for $j = 1, ..., k-1$
and with x_t cointegrated (Definition 1) $x_t \in \mathbb{R}^n \sim I(1)$ and $\Delta x_t \sim I(0)$
The error correction term Πx_{t-1} is used to keep and control the long-run dynamic of the time series. An example of how error-correction models operate in Example 2.
We can also apply online convex optimization on EC-VARMA:

$$\Delta \widetilde{x}_t = \widehat{\Pi} x_{t-1} + \sum_{i=1}^{M} \Gamma_i \Delta x_{t-i}$$

$$\widetilde{x}_t = x_{t-1} + \widehat{\Pi} x_{t-1} + \sum_{i=1}^{M} \Gamma_i \Delta x_{t-i}$$

Example 2 (How do error-correction models work)**.** *Imagine two $I(1)$ time series x_t and y_t with a long-run equilibrium $x_t = \beta y_t$, $x_t - \beta y_t = 0 \sim I(0)$, $x_t, y_t \sim CI(1,1)$ with cointegration vector $(1, -\beta)$. Now if at time t, $x_t > \beta y_t$, there will be a disturbance in the equilibrium. We face 3 choices to recover:*

1. *a decrease in x_t and/or an increase in y_t*
2. *an increase in x_t and a faster increase in y_t*
3. *a decrease in x_t and a slower decrease in y_t*

This can be modeled as:

$$\Delta x_t = -\alpha_1(x_{t-1} - \beta y_{t-1}) + \epsilon_1$$
$$\Delta y_t = \alpha_2(x_{t-1} - \beta y_{t-1}) + \epsilon_2$$

Where $\alpha_i > 0$ represents the speed of adjustment to the long run equilibrium. In this case we will have $\begin{pmatrix}\Delta x_t \\ \Delta y_t\end{pmatrix} = \Pi \begin{pmatrix} x_{t-1} \\ y_{t-1}\end{pmatrix} + \epsilon_t$ with $\Pi = \begin{pmatrix}-\alpha_1 & \beta\alpha_1 \\ \alpha_2 & -\beta\alpha_2\end{pmatrix}$ which will bear the cointegrating vector and the speed of adjustment information.

a Settings

1. $\kappa = \left\{\Gamma_i, \widehat{\Pi} \in \mathbb{R}^{n\times n} : ||\widehat{\Pi}||_* \leqslant \rho, ||\Gamma_i||_{\max} \leqslant 1 \quad i = 1, ..., M\right\}$
2. Let $\psi = \left\{\widehat{\Pi}, \Gamma_1, \Gamma_2, ..., \Gamma_M\right\}$
3. $l_t(\widetilde{x}_t) = ||x_t - \widetilde{x}_t||_2^2 = (x_{1,t} - \widetilde{x}_{1,t})^2 + ... + (x_{n,t} - \widetilde{x}_{n,t})^2$
   $= (x_{1,t} - x_{t-1} - \widehat{\Pi}[1,]x_{t-1} - \sum_{i=1}^{M} \Gamma_i[1,]\Delta x_{t-i})^2 + ... + (x_{n,t} - x_{t-1} - \widehat{\Pi}[n,]x_{t-1} - \sum_{i=1}^{M} \Gamma_i[n,]\Delta x_{t-i})^2$

4. $\nabla_{\Gamma_i} l_t = \frac{\partial l_t}{\partial \Gamma_i} = -2 \begin{pmatrix} (x_{1,t} - \widetilde{x}_{1,t}) \Delta x_{t-i}^\top \\ (x_{2,t} - \widetilde{x}_{2,t}) \Delta x_{t-i}^\top \\ \vdots \\ (x_{n,t} - \widetilde{x}_{n,t}) \Delta x_{t-i}^\top \end{pmatrix} \qquad \nabla_{\widehat{\Pi}} l_t = \frac{\partial l_t}{\partial \widehat{\Pi}} = -2 \begin{pmatrix} (x_{1,t} - \widetilde{x}_{1,t}) x_{t-1}^\top \\ (x_{2,t} - \widetilde{x}_{2,t}) x_{t-1}^\top \\ \vdots \\ (x_{n,t} - \widetilde{x}_{n,t}) x_{t-1}^\top \end{pmatrix}$

5. $G = 2n.(X_{max})^2$ where $X_{max} = \max_i x_{i,t}$

6. $D = 2n = 2 \max_{X \in \kappa} ||X||_2 \geqslant \sup_{X,Y \in \kappa} ||X - Y||_2$

7. $\lambda = \frac{1}{n}$

b EC-VARMA OGD

Algorithm 8 describes the Online Gradient Descent technique applied to the ec-varma model.

Algorithm 8 EC-VARMA online gradient descent

Input Horizon M, first step size $\eta_1 = \frac{D}{G}$, nuclear norm ball radius ρ, $\psi^1 \in \kappa$, T, in-sample data $\{x\}$
for $t = 1$ **to** T
  Update $\eta_t \leftarrow \frac{1}{\sqrt{t}} \eta_1$
  Predict $\widetilde{x}_t = x_{t-1} + \widehat{\Pi} x_{t-1} + \sum_{i=1}^{M} \Gamma_i \Delta x_{t-i}$
  Observe x_t and suffer loss $l_t(\psi^t)$
  Project and **Update:** $\Gamma_i^{t+1} \leftarrow \prod_\kappa (\Gamma_i^t - \eta_t \nabla_{\Gamma_i} l_t)$
  Project and **Update:** $\widehat{\Pi}^{t+1} \leftarrow \prod_{\mathscr{B}(*,\rho)} (\widehat{\Pi}^t - \eta_t \nabla_{\widehat{\Pi}} l_t)$
end for

c EC-VARMA ONS

Algorithm 9 describes the Online Newton Step technique applied to the ec-varma model.

Algorithm 9 EC-VARMA online newton step

Input Horizon M, step size $\eta = \frac{1}{2} \min\{\lambda, \frac{1}{4GD}\}$, $\psi^1 \in \kappa$, initial list of M $(n \times n)$ matrices $A_i = \frac{1}{\eta^2 D^2} I_n$, T, in-sample data $\{x\}$
for $t = 1$ **to** T
  Predict $\widetilde{x}_t = x_{t-1} + \widehat{\Pi} x_{t-1} + \sum_{i=1}^{M} \Gamma_i \Delta x_{t-i}$
  Observe x_t and suffer loss $l_t(\psi^t)$
  Update $A_{i+1}^t \leftarrow A_{i+1}^{t-1} + \nabla_{\Gamma_i} l_t \nabla_{\Gamma_i} l_t^\top$
  Update $A_1^t \leftarrow A_1^{t-1} + \nabla_{\widehat{\Pi}} l_t \nabla_{\widehat{\Pi}} l_t^\top$
  Project and **Update:** $\Gamma_i^{t+1} \leftarrow \prod_\kappa^{A_{i+1}^t} (\Gamma_i^t - \frac{1}{\eta} (A_{i+1}^t)^{-1} \nabla_{\Gamma_i} l_t)$
  Project and **Update:** $\widehat{\Pi}^{t+1} \leftarrow \prod_{\mathscr{B}(*,\rho)} (\widehat{\Pi}^t - \frac{1}{\eta} (A_1^t)^{-1} \nabla_{\widehat{\Pi}} l_t)$
end for

5.4 Experiments

We test these mulrivariate algorithms on synthetic and real data. In all the experiments, we set $M = 10$ and $\rho = 1$.

Remark. *The MLE fails to predict very often, here we choose an appropriate part of the data in which it can make predictions without throwing errors.*

a Setting 5: Sanity Check 1

Synthetic data generated by the R function arima.sim with correlated noise: noise terms are normally distributed with mean the value of the previous noise term (Figure 12). The parameters used are in Table 6 and the results are shown in Figure 13.

α_1	$(0.11, -0.5)$
α_2	$(0.31, -0.2)$
α_3	$(0.6, -0.4)$
β_1	$(0.41, -0.39, -0.685, 0.1)$
β_2	$(0.1, -0.1, -0.5, 0.2)$
β_3	$(0.41, -0.1, -0.5, 0.2)$
σ_i	0.05
σ_2	0.01
σ_3	0.15
run	500
training run	5000
MLE window size	252

Table 6: Settings 5

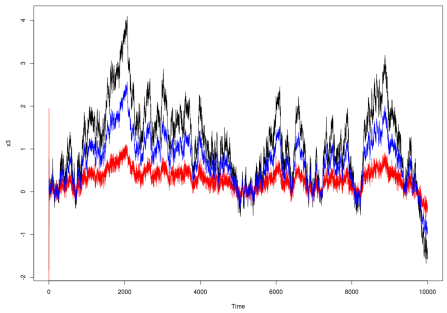
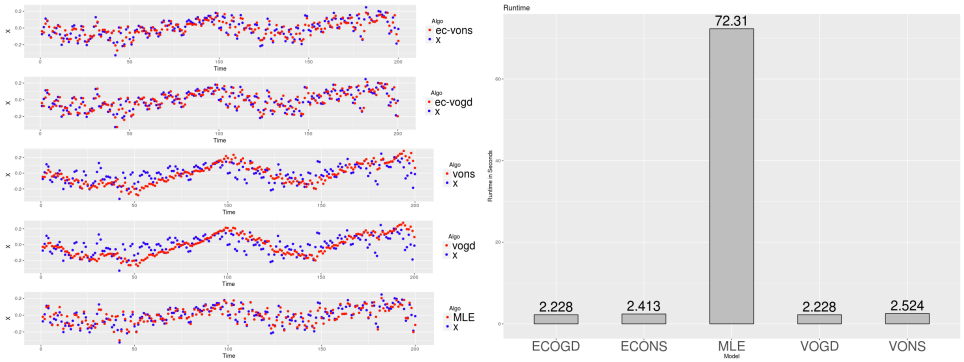
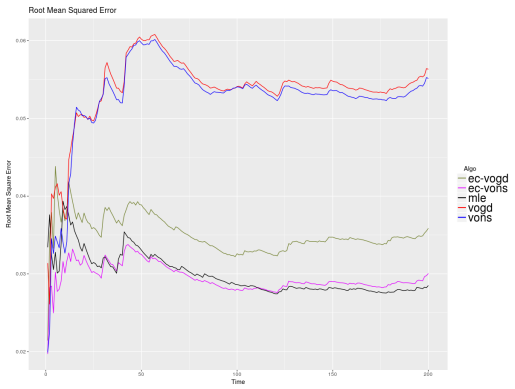


Figure 12: Sythetic data



(a) Predictions for first series

(b) Runtime (in s)



(c) RMSE

Figure 13: Setting 5 results

b Setting 6: Sanity Check 2

Synthetic data generated by the R function arima.sim with gaussian noise(Figure 14). The parameters used are in Table 7 and the results are shown in Figure 15.

α_1	$(0.11, -0.5)$
α_2	$(0.31, -0.2)$
α_3	$(0.6, -0.4)$
β_1	$(0.41, -0.39, -0.685, 0.1)$
β_2	$(0.1, -0.1, -0.5, 0.2)$
β_3	$(0.41, -0.1, -0.5, 0.2)$
σ_i	0.05
run	200
training run	5000
MLE window size	252

Table 7: Settings 6

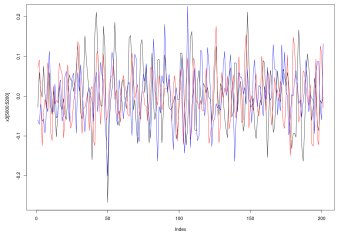
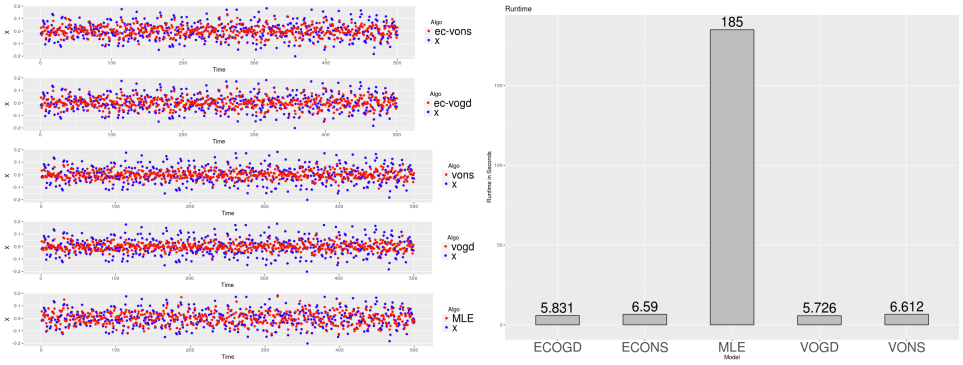
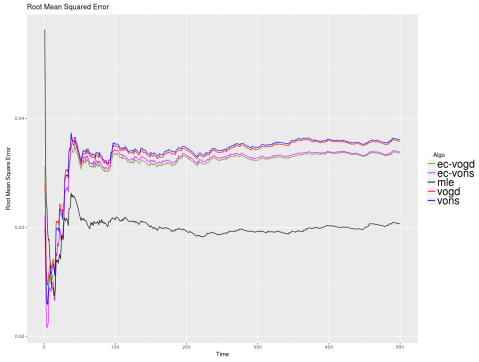


Figure 14: Sythetic data



(a) Predictions for first series

(b) Runtime (in s)



(c) RMSE

Figure 15: Setting 6 results

c Setting 7: Real Data, bitcoin volume traded in EUR per 15 min on different markets

We test the univariate and multivariate algorithms on real data: Bitcoin volume traded in EUR per 15 min on different markets (Coinbase, Bitstamp, Btcde), data represented in Figure 16. The parameters used are in Table 8 and the results are shown in Figure 17.

run	500
training run	3000
MLE window size	252

Table 8: Setting 7

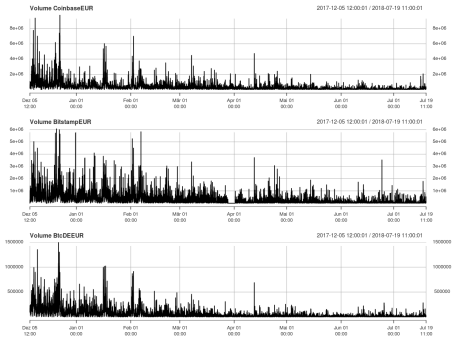
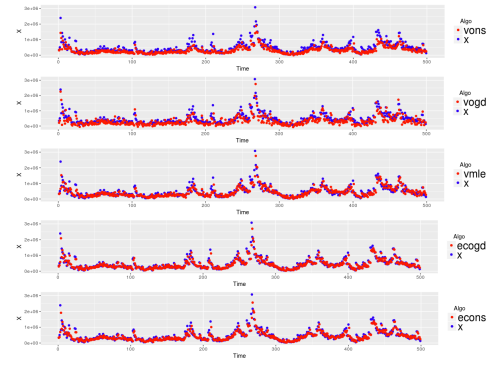
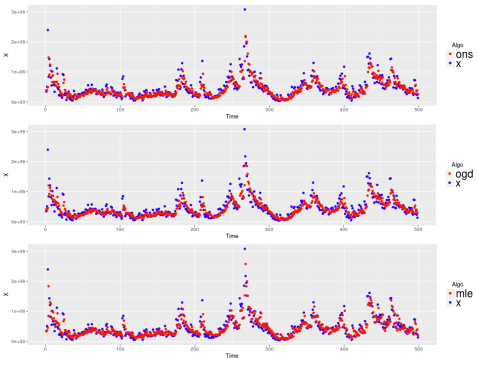


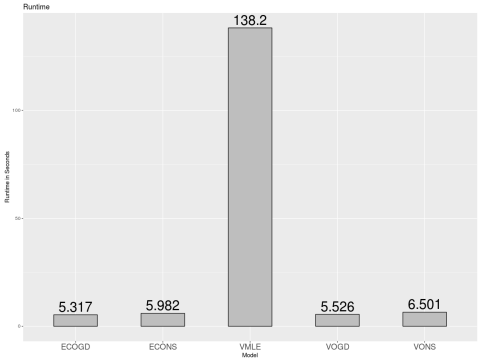
Figure 16: Bitcoin volume traded in EUR per 15 min in $



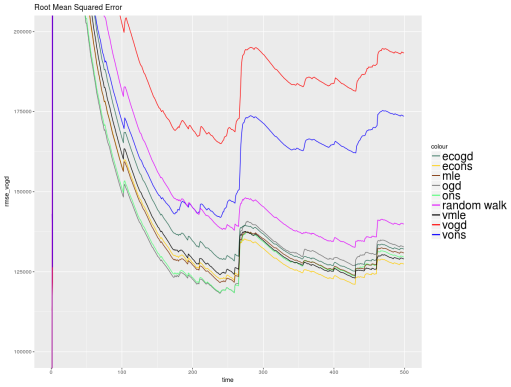
(a) Multivariate algorithms predictions for first series



(b) Univariate algorithms predictions for first series



(c) Runtime (in s)



(d) RMSE

Figure 17: Setting 7 results

6 Package presentation and content (more details for usage below)

6.1 Utilitiy functions

`squared_loss`	Computes the squared loss function of two vectors
`inv_sm`	Computes efficiently $(A+uu^\top)^{-1}$ using the Sherman-Morrison formula
`sqrtm_diag`	Computes the square root of a matrix using diagonalization
`sqrtm_dbit`	Computes the square root of a matrix using Denman-Beavers iteration
`projection`	Computes $\prod_\kappa^A(v)$ with κ a set with with constraint c, $\kappa = \left\{v \in \mathbb{R}^n : \lvert v_i\rvert \leqslant c \quad i = 1,...,n\right\}$
`mprojection`	Computes $\prod_\kappa^W(A)$ with κ a set with with constraint c, $\kappa = \left\{A \in \mathbb{R}^{n\times n} : \lvert\lvert A\rvert\rvert_{max} \leqslant c\right\}$
`nnbprojection`	Computes $\prod_{\mathscr{B}(*,\rho)}(A)$ with $\mathscr{B}(*,\rho)$ a nuclear norm ball of radius ρ
`l1projection`	Computes the projection of a vector into an $l1$-ball of radius ρ $\left\{v \in \mathbb{R}^n : \Sigma_i \lvert v_i\rvert \leqslant \rho\right\}$ **(details of this function below in Algorithm 10)**

6.2 OCO functions

`arma_ogd`	ARMA Online Gradient Descent
`arma_ons`	ARMA Online Newton Step
`varma_ogd`	VARMA Online Gradient Descent
`varma_ons`	VARMA Online Newton Step
`ecvarma_ogd`	EC-VARMA Online Gradient Descent
`ecvarma_ons`	EC-VARMA Online Newton Step

Algorithm 10 Projection of a vector onto an l1-ball $\left\{v \in \mathbb{R}^n : \Sigma_i |v_i| \leqslant \rho\right\}$

Input Vector v, radius ρ
Settings $y = v$, $p = 0$, $z = -\rho$
While $y \neq \emptyset$
  Choose pivot $k \leftarrow$ Random element from y
  Set $\mathscr{H} \leftarrow \{t \in v : t > k\}$, $\mathscr{L} \leftarrow \{t \in v : t < k\}$, $\mathscr{M} \leftarrow |\{t \in v : t = k\}|$
  if $(s + Mk + \sum_{t \in \mathscr{H}} t)/(p + M + |\mathscr{H}| < k)$
    $s \leftarrow s + Mk + \sum_{t \in \mathscr{H}} t$
    $p \leftarrow p + M + |\mathscr{H}|$
    $y \leftarrow \mathscr{L}$
  else $y \leftarrow \mathscr{H}$
end while
$\theta = s/p$
for $i = 1$ **to** n
  Set $\text{proj}_i = max(v_i - \theta, 0)$
end for
Return proj

7 Conclusion

We considered a new approach of time series prediction based on online learning, we presented the online convex optimization model along with two algorithms and their applications on univariate and multivariate ARMA models. We realized some experiences to test the reliability of these algorithms. The main results of this report can be resumed as follows:

1. There is an uncomparable difference of running time between the Maximum-Likelihood estimation and the OCO algorithms.
2. The Maximum-likelihood estimation makes better predictions on synthetic data (provided a growing window)
3. On real data, the OCO algorithms make better performance since this implies fast adaptation from noisy data and regime shifts.

7.1 Future research

Online convex optimzation can be studied for many other applications such as the ARCH model and universal portfolio selection.

References

[CXF17] Aveleen Bijral Christopher Xie and Juan Lavista Ferres. Online Prediction Methods for Nonstationary Time Series. 2017.

[Haz16] Elad Hazan. *Introduction to Online Convex Optimization.* 2016.

[OAS13] Shie Mannor Oren Anava, Elad Hazan and Ohad Shamir. Online Learning for Time Series Prediction. *JMLR: Workshop and Conference Proceeding*, 2013, 2013.