Università di Pisa

Department of Mathematics
Master's Degree in Mathematics

Markov Switching Quantile Regression

Master thesis

Candidate:
Mario Correddu

Supervisor:
Prof. **Andrea Agazzi**

Academic year 2023/2024

Contents

Introduction

This thesis investigates the Hidden-Markov-Switching Quantile Regression model, a hybrid approach that combines quantile regression and Hidden Markov models.

Quantile regression is a statistical method for modeling conditional quantile functions. In contrast to the emphasis of classical least-squares regression on the conditional mean, quantile regression provides an approach to exploring the impact of covariates estimates to various quantiles of the response variable distribution, providing insights into conditional relationships, particularly valuable in fields like economics and finance. Economic variables often depend on the prevailing economic state, making Hidden Markov Models (HMMs) suitable for capturing changing dependencies. HMM assumes that the regime transition of our observed data is a finite-state Markov chain, that is an unobserved process characterized by the hidden state variables S_t. By integrating these models, we aim to achieve a better understanding of both conditional variation and hidden dynamics in the data.

We consider a model where the response variable depends linearly on the predictors, but the coefficients of such linear model depend, in turn, on an underlying finite-state, hidden Markov chain S_t. The theory of HMMs can thus be applied to our framework. In particular, parameter estimation is performed through the Expectation-Maximization algorithm (EM), alternating between computing the expectation of the log-likelihood of the model with respect to the estimated distribution of the hidden states, and maximizing the log-likelihood with respect to the model parameters.

For data with homogeneous frequency, the maximization step of the quantile regression side is carried out through linear quantile regression. However in the case of multifrequency data, over-parametrization is encountered and in order to overcome this problem, we introduce a reparametrization of the linear coefficient in the form of Almon exponential polynomials. Due to the now nonlinear nature of the optimization, Adam and the Nelder-Mead algorithm are chosen to solve the maximization step of parameter fitting: Adam is a stochastic-gradient-based optimization algorithm, that computes individual adaptive learning rates for different parameters; Nelder-Mead is a direct search method algorithm that iteratively generates a sequence of simplices to approximate an optimal point.

Another challenge inherent in such models lies in their sensitivity to the initialization point during estimation. To tackle this issue, we adopted the following strategy: an initial state partition is chosen using one of the initialisation strategies given at the end of this paragraph; the initial probability of the Markov model is then computed based on this partition, and the transition matrix elements are calculated as proportions of transitions. Concerning the initial parameters related to the Laplace distribution, quantile regression is performed separately to the observations in each hidden state. In order to obtain the initial state partition, we propose three methods: a random initialization, which computes several possibilities and selects the best-performing one based on likelihood after a small number of steps of EM; a clustering-based approach using the k-means algorithm; and a data-driven technique where we leverage information on recessions to initialize our economic models.

Additionally, we discuss and implement various statistical tests to assess the model's robustness and reliability. These tests rely on a particular function of the prediction of the quantiles, the $Hit(\beta^0)_t$ function, whose conditional expectation given any information known at $t-1$ must be 0. We evaluate the efficacy of the tests using simulated data and delineate the requisite conditions under which the tests are applicable.

We then perform a simulation study to investigate the behaviour of the proposed model where we present results from the estimation of quantiles of GDP for both the EU area and the US.

Finally, we extend our model by incorporating a quantile autoregressive (QAR) structure on the response variable, allowing for dependence on past quantiles along with the hidden regime transitions. Within this extended framework assumptions, we prove the consistency of the quantile regression estimates.

This work has been developed in collaboration with the European Central Bank, which provided the data and economic insights about the work. The ideas of this thesis are thought to be helpful in gaining a better understanding of the relationship that interest rates decided by the ECB may have on the economic situation.

Acknowledgements

I would like to express my gratitude to Max Lampe for providing the data and its invaluable support and guidance throughout this project. Additionally, I would like to express my sincere appreciation to my supervisor, Professor Agazzi, whose assistance and insights were crucial for the completion of this thesis.

Chapter 1

Quantile Regression

The classical theory of linear models is essentially a theory for models of conditional expectations. However, conditional mean might not be a satisfactory end in itself, even for statistical analysis of a single sample. Measures of spread, skewness, kurtosis as well as boxplots, histograms, and more sophisticated density estimation are all frequently employed to gain further insight. A way to go beyond these models is provided by quantile regression, that may represent a comprehensive approach to the statistical analysis of linear and nonlinear response models. Quantile regression supplements the exclusive focus of least squares based methods on the estimation of conditional mean functions with a general technique for estimating families of conditional quantile functions. This greatly expands the flexibility of both parametric and nonparametric regression methods, and has found multiple applications in the field of econometrics and finance. This chapter builds on the work of [1], which provides a detailed discussion on quantile regression. For a deeper exploration of regularization techniques, we recommend referring to [2].

1.1 Definition of quantile regression

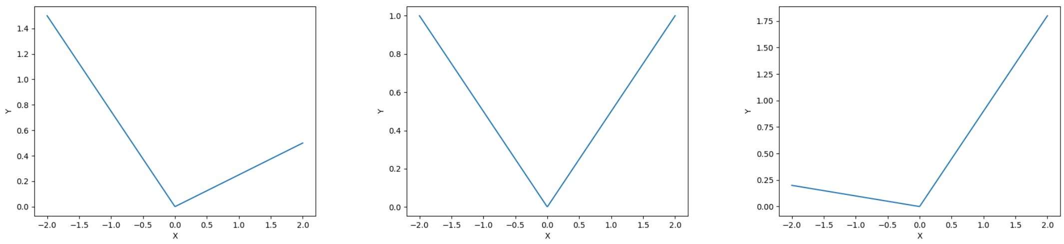
Definition 1.1.1. Let X be a real valued random variable with cumulative distribution function $F(x) = \mathcal{P}(X \leq x)$. The quantile at level τ is given by

$$Q_X(\tau) = \inf \{x | F(x) \geq \tau\} \tag{1.1.1}$$

While this is the usual definition of quantile, one can see this as the solution of an optimization problem. We firstly define the loss function:

$$\rho_\tau(u) = u(\tau - \mathbb{1}_{(u<0)}) = u\left((\tau - 1)\mathbb{1}_{(u<0)} + \tau\mathbb{1}_{(u\geq 0)}\right) \tag{1.1.2}$$

Which is a function of the form:



(a) $\tau = 0.25$ (b) $\tau = 0.5$ (c) $\tau = 0.9$

Figure 1.1: plot of the ρ_τ function for different choices of τ

Then given a random variable X our claim is that the quantile at level τ is the given by the solution $\hat{x}$ of the following optimization problem:

$$\underset{x\in[0,1]}{\operatorname{argmin}}\, \mathbb{E}\left[\rho_\tau(X-x)\right] \tag{1.1.3}$$

In order to show the relation between $\hat{x}$ and the actual τ-th quantile of X (which we suppose to exist), we compute the derivative of the loss with respect to $\hat{x}$. Thus we have

$$(1-\tau)\int_{-\infty}^{\hat{x}} dF(x) + \tau \int_{\hat{x}}^{\infty} dF(x) = F(\hat{x}) - \tau = 0 \tag{1.1.4}$$

We observe that any element of $\{x : F(x) = \tau\}$ minimizes the expected loss, thus in practical application its smallest element must be chosen to adhere to the convention that the empirical quantile function be left-continuous.

In applications, where we have to manage real data, this is applied using the empirical cumulative distribution. Suppose we have a sample of size n, $\{X_i\}_{i=1}^n$, then its empirical cumulative distribution function is defined as:

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{X_i \le x\}} \tag{1.1.5}$$

and thus the expected loss with the actual data becomes:

$$\frac{1}{n}\sum_{i=1}^{n} \rho_\tau(x_i - x) \tag{1.1.6}$$

where x_i are the observed values of X_i for $i = 1, \dots, n$.

We can now formulate quantile regression starting from this formulation. Given some input variable $\{x_i\}_{i=1}^n$ and some output variable $\{y_i\}_{i=1}^n$, performing a quantile regression at level τ means solving the following optimization problem:

$$\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} \rho_\tau(y_i - \xi(x_i,\beta)) \tag{1.1.7}$$

where $\xi(x, \beta)$ represents some class of functions parameterized by $\beta \in \mathbb{R}^p$ for some $p \in \mathbb{N}$.

In this framework what we are trying to learn is the τ-th conditional (w.r.t. X) quantile function of Y defined as

$$Q_Y(\tau|X) = \inf\{y : P(Y \leq y|X) \geq \tau\} \tag{1.1.8}$$

A special case of quantile regression is when the function $\xi(x_i, \beta)$ is linear of the form $X\beta$, where we assume we incorporated the intercept through adding a column of ones to the data X.

Thus the problem can be reformulated as a problem of linear programming and it's possible to solve it through the known algorithms such as the simplex method. With some calculation one can deduce the following expression for such optimization problem:

$$\min_{(\beta,u,v)\in\mathbb{R}^p\times\mathbb{R}^{2n}_+} \{\tau\mathbf{1}_n u + (1-\tau)\mathbf{1}_n v | W\beta + u - v = y\} \tag{1.1.9}$$

,

where $\mathbf{1}_n$, is the vector of all 1's and length n, and W is the matrix whose columns consist of all the different realizations of the X data.

1.2 Properties

Some interesting properties specific to the linear quantile regression, are the so-called equivariance properties. Their usefulness stands in how they help model interpretation, as they encode some relationships between changes in data and changes in regression estimates

Theorem 1.2.1. *Let A be a $p \times p$ nonsingular matrix, $\gamma \in \mathbb{R}^p$, and $a > 0$. In order to highlight the relationship with the underlying variables, let's call $\hat{\beta}(\tau; y, W)$, the solution of problem 1.1.7. Then for any $\tau \in [0, 1]$,*

1. $\hat{\beta}(\tau; ay, W) = a\hat{\beta}(\tau; y, W)$
2. $\hat{\beta}(\tau; -ay, W) = -a\hat{\beta}(1-\tau; y, W)$
3. $\hat{\beta}(\tau; y + W\gamma, W) = \hat{\beta}(\tau; y, W) + \gamma$
4. $\hat{\beta}(\tau; y, WA) = A^{-1}\hat{\beta}(\tau; y, W)$

Proof. The proof relies on the known properties of the solutions for linear programming problems. □

Going back to the more general framework, we now observe that another equivariance property holds, one much stronger than those already discussed. Let h be a nondecreasing function on $\mathbb{R}$. Then, for any random variable Y ,

$$Q_{h(Y)}(\tau) = h(Q_Y(\tau)) \tag{1.2.1}$$

that is, the quantiles of the transformed random variable $h(Y)$ are transformed quantiles of the original Y.

This follows immediately from the elementary fact that, for any monotone h,

$$P(Y \leq y) = P(h(Y) \leq h(y)) \tag{1.2.2}$$

We highlight that the mean of standard linear regression does not share this property:

$$E\left[h(Y)\right] \neq h(E[Y]) \tag{1.2.3}$$

except for affine h or other exceptional circumstances.

This translates into an easier interpretation, when we train our model not directly on Y but on some monotonic transformation $h(Y)$ of the data. Thus, thanks to the equivariance property, one can straight think of $h^{-1}(x\beta)$ as an estimate of the conditional quantile of Y given X.

A useful application of this property consists in dealing with censored data.

Let y_i^* denote a latent (unobservable) variable assumed to be generated from the linear model

$$y_i^* = x_i\beta + u_i \tag{1.2.4}$$

for $i = 1, ..., n,$, where $\{u_i\}$ is independently and identically distributed (iid) from a distribution function F with density f . Censoring, consists in not observing the y_i^* -s directly, but instead we see:

$$y_i = \max\{0, y_i^*\} \tag{1.2.5}$$

the equivariance of the quantiles to monotone transformations implies a fairly simple expression for the conditional quantile functions of the response, y_i , in model 1.2.4

$$Q_{y_i}(\tau|x_i) = \max\{0, x_i\beta + F_u^{-1}(\tau)\}. \tag{1.2.6}$$

We observe that it is also straightforward to accommodate observation-specific censoring from the right and left.

Another useful property that is derived from the monotone equivariance is in terms of the interpretation of the coefficients of the regression. We have that in standard linear regression and quantile regression the following equalities hold respectively:

$$\frac{\partial E\left[Y|X=x\right]}{\partial x_j} = \beta_j \tag{1.2.7}$$

$$\frac{\partial Q_Y\left(\tau|X=x\right)}{\partial x_j} = \beta_j \tag{1.2.8}$$

.

However, only in the framework of quantile regression, if we assume that instead

$$Q_{h(Y)}\left(\tau|X=x\right) = x^{\intercal}\beta$$

the following equality is also true:

$$\frac{\partial Q_Y\left(\tau|X=x\right)}{\partial x_j} = \frac{\partial h^{-1}(x^{\intercal}\beta)}{\partial x_j} \tag{1.2.9}$$

This grants an easier interpretation of those models where what we see is a monotone function of our variable interest. However we stress also that interpreting coefficients of linear quantile regression is generally a harder task than those of standard linear regression.

1.3 Regularization

There are two reasons why we are often not satisfied with results of our regression:

- The first is prediction accuracy: having too many features may decrease the quality of our prediction, leading to overfitting, or in other words making our model learn the noise instead of the structure of the data
- The second reason is interpretation. With a large number of predictors, we often would like to determine a smaller subset that exhibit the strongest effects. In order to get the "big picture," we are willing to sacrifice some of the small details.

A way to tackle this problem is through Shrinkage, that is to fit a model containing all predictors using a technique that constrains or regularizes the coefficient estimates, or equivalently, that shrinks the coefficient estimates towards zero.

The most natural way to do so is to use lasso. Lasso is defined as:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \rho_\tau(y_i - \beta' x_i + \beta_0)) - \alpha ||\beta'||_1 \tag{1.3.1}$$

Where $\beta = (\beta_0, \beta')$, and β_0 is the intercept of the regression. In fact we have that for $\tau = 1/2$ this estimator can be computed with a simple data augmentation device and for $\tau \neq 1/2$ the situation is just slightly more complicated since we want asymmetric weighting of the residual term and symmetric weighting of the penalty term. The key property of lasso is that it acts more like a model selection penalty, shrinking β coefficients all the way to their 0 coordinates when α is sufficiently large. For this reason we say that the lasso yields sparse models, that is it involves only a subset of the variables.

Chapter 2

Hidden Markov Models

A hidden Markov model (HMM) is a statistical framework used to describe observable events that are influenced by internal, unobservable factors. An HMM comprises two interconnected stochastic processes: a series of hidden states forming a Markov chain, and a corresponding series of observable variables whose distribution is determined by these hidden states. Due to their ability to model hidden states influencing observable sequences, HMMs have diverse applications in various fields. A prominent example is speech recognition. where, the HMMs help identify the sequence of hidden states (phonemes) that underlie the spoken word (observable sequence) based on the audio signal. Another application is found in bioinformatics where they are used to analyze DNA sequences to identify gene structures. In finance and econometrics, HMMs have also been extensively employed for regime detection tasks. This chapter is a reelaboration of the presentation of hidden Markov models provided in[3] with some computational insights recovered from [4].

2.1 Definitions and notations

Definition 2.1.1. (Transition Kernel). Let $(\mathbb{S}, \mathcal{S})$ and $(\mathbb{Y}, \mathcal{Y})$ be two measurable spaces. A transition kernel from $(\mathbb{S}, \mathcal{S})$ to $(\mathbb{Y}, \mathcal{Y})$ is a function $Q : \mathbb{S} \times \mathcal{Y} \to [0, \infty]$ that satisfies:

(i) for all $x \in \mathbb{S}, Q(s, \cdot)$ is a positive measure on $(\mathbb{Y}, \mathcal{Y})$;

(ii) for all $A \in \mathcal{Y}$, the function $x \mapsto Q(s, A)$ is measurable.

(iii) $Q(s, Y) = 1$ for all $s \in \mathbb{S}$

If $\mathbb{S} = \mathbb{Y}$ for all $s \in \mathbb{S}$, then Q will be referred to as a Markov transition kernel on $(\mathbb{S}, \mathcal{S})$.

A transition kernel Q is said to admit a density with respect to the positive measure μ on $\mathbb{Y}$ if there exists a non-negative function $q : \mathbb{S} \times \mathbb{Y} \to [0, \infty]$, measurable with respect to the product σ-field $\mathcal{S} \otimes \mathcal{Y}$, such that

$$Q(s, A) = \int_A q(s, y)\mu(dy), \quad A \in \mathcal{Y}$$

The function q is then referred to as a transition density function. When $\mathbb{S}$ and $\mathbb{Y}$ are countable sets it is customary to write $Q(s, y)$ as a shorthand notation for $Q(s, \{y\})$, and Q is generally referred to as a transition matrix (whether or not $\mathbb{S}$ and $\mathbb{Y}$ are finite sets).

If Q is an (unnormalized) Markov transition kernel on $(X, \mathcal{X})$, its iterates are defined inductively by

$$Q_0(x, \cdot) = \delta_x \quad \text{for } x \in X,$$
$$Q_k = QQ_{k-1} \quad \text{for } k \geq 1.$$

These iterates satisfy the Chapman-Kolmogorov equation: $Q_{n+m} = Q_n Q_m$ for all $n, m \geq 0$. That is, for all $x \in X$ and $A \in \mathcal{X}$,

$$\int Q_{n+m}(x, A)\, dx = \int Q_n(x, dy) Q_m(y, A). \tag{2.1.1}$$

If Q admits a density q with respect to the measure μ on $(X, \mathcal{X})$, then for all $n \geq 2$, the kernel Q_n is also absolutely continuous with respect to μ. The corresponding transition density is

$$q_n(x, y) = \int q(x, x_1) \cdots q(x_{n-1}, y)\, \mu(dx_1) \cdots \mu(dx_{n-1}). \tag{2.1.2}$$

Definition 2.1.2 (stochastic process)**.** Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space and let $(\mathbb{X}, \mathcal{X})$ be a measurable space. An $\mathbb{X}$-valued (discrete index) stochastic process $\{X_n\}_{n\geq 0}$ is a collection of $\mathbb{X}$-valued random variables. A filtration of $(\Omega, \mathcal{F})$ is a non-decreasing sequence $\{\mathcal{F}_n\}_{n\geq 0}$ of sub- σ-fields of $\mathcal{F}$. A filtered space is a triple $(\Omega, \mathcal{F}, \mathbb{F})$, where $\mathbb{F}$ is a filtration; $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ is called a filtered probability space. For any filtration $\mathbb{F} = \{\mathcal{F}_n\}_{n\geq 0}$, we denote by $\mathcal{F}_\infty = \vee_{n=0}^{\infty} \mathcal{F}_n$ the σ-field generated by $\mathbb{F}$ or, in other words, the minimal σ-field containing $\mathbb{F}$. A stochastic process $\{X_n\}_{n\geq 0}$ is adapted to $\mathbb{F} = \{\mathcal{F}_n\}_{n\geq 0}$, or simply $\mathbb{F}$-adapted, if X_n is $\mathcal{F}_n$-measurable for all $n \geq 0$ The natural filtration of a process $\{X_n\}_{n\geq 0}$, denoted by $\mathbb{F}^X = \left\{\mathcal{F}_n^X\right\}_{n\geq 0}$, is the smallest filtration with respect to which $\{X_n\}$ is adapted.

Definition 2.1.3. (Markov Chain). Let $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ be a filtered probability space and let Q be a Markov transition kernel on a measurable space (X, $\mathcal{X}$). An X-valued stochastic process $\{X_k\}_{k\geq 0}$ is said to be a Markov chain under $\mathbb{P}$, with respect to the filtration $\mathbb{F}$ and with transition kernel Q, if it is $\mathbb{F}$-adapted and for all $k \geq 0$ and $A \in \mathcal{X}$,

$$\mathbb{P}\left(X_{k+1} \in A \mid \mathcal{F}_k\right) = Q\left(X_k, A\right)$$

The distribution of X_0 is called the initial distribution of the chain, and $\mathbf{X}$ is called the state space.

Definition 2.1.4 (Stationary Process)**.** A stochastic process $\{X_k\}$ is said to be stationary (under P) if its finite-dimensional distributions are translation invariant, that is, if for all $k, n \geq 1$ and all $n_1, \ldots, n_k$, the distribution of the random vector $(X_{n_1+n}, \ldots, X_{n_k+n})$ does not depend on n

2.2 Hidden Markov Models

Given a time series $\{y_t\}_{t=0}^T \equiv y_0^T$ we would like to exploit the structure of discrete Markov chains to model its distribution as a process Y_0^T. One common practice is to assume a hidden process S_0^T that determines the distribution of the Y_t variables, and to assume that such process is a Markov chain, with a finite number of states.

Definition 2.2.1 (HMM). Let $(Y_t, S_t)_{t=0}^{\infty}$ be a discrete-time stochastic process such that, for each $t \in \mathbb{N}$, $S_t \in \mathbb{S} \equiv \{0, \ldots, k\}$ is the unobservable state and $Y_t \in \mathbb{Y} \subseteq \mathbb{R}^h$, for some $h \in \mathbb{N}$, is the observable state. A Hidden Markov Model is a statistical model such that for each $t \in \mathbb{N}$, the conditional distribution of Y_t, given Y_0^{t-1} and S_0^t, depends only on S_t, and the conditional distribution of S_t, given Y_0^{t-1} and S_0^{t-1}, depends only on S_{t-1}, so that

$$\begin{aligned} Y_t \mid \left(Y_0^{t-1}, S_0^t\right) &\sim P_{\theta^*}\left(S_t, \cdot\right) \\ S_t \mid \left(Y_0^{t-1}, S_0^{t-1}\right) &\sim Q_{\theta^*}\left(S_{t-1}, \cdot\right) \end{aligned} \tag{2.2.1}$$

where $\theta^* = (\pi^*, A^*, \beta^*) \in \mathbb{R}^k \times \mathbb{R}^{k \times k} \times \mathbb{R}^{d \times k}$ are the defining parameters of the model. In particular for $i, j = 1...k$:

- $\pi_i^* = \bar{P}_*^{\pi}(S_0 = i)$ are called the starting probabilities
- $a_{ij}^* = Q_{\theta^*}(S_{t-1} = j, S_t = i)$ are the entries of the transition matrix
- $\beta_i^* \in \mathbb{R}^d$ are the parameters that define the distribution of $P_\theta(S_t = i, Y_t)$, also called the emission distribution

where $A = (a_{i,j})_{i,j=1}^k$, $\beta = (\beta_1, \ldots, \beta_k)$ and $\bar{P}_*^{\pi}$ denote the probability distribution over $(Y_t, S_t)_{t=-\infty}^{\infty}$.

Finally it is often assumed that, for each $s \in \mathbb{S}$, $P_{\theta^*}(s, \cdot)$ admits a density $f_{\theta^*}(s, \cdot) \equiv f(\cdot\, ; \beta_s^*)$ with respect to some σ-finite measure μ on $\mathbb{Y}$. Thus β^* will be the parameters defining such density.

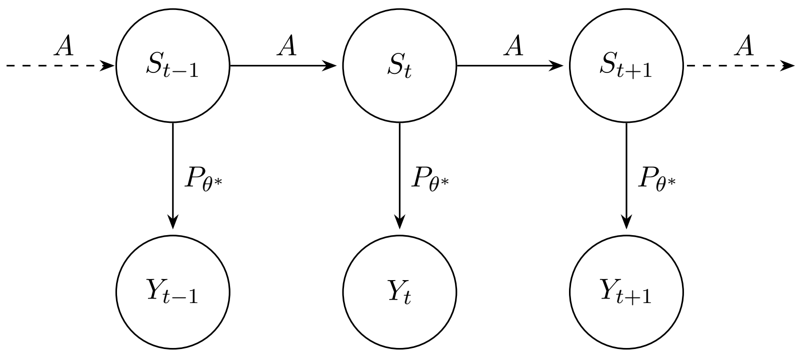


Figure 2.1: Visual representation of a HMM

A fundamental issue in hidden Markov modeling is: given a fully specified model and some observations $y_0, \ldots, y_n$, what can be said about the corresponding unobserved state sequence $s_0, \ldots, s_n$? More specifically, we shall be concerned with the evaluation of the conditional distributions of the state at index k, s_k , given the observations $y_0, \ldots, y_n$, a task that is generally referred to as smoothing.

Before dwelling into the computations, we fix the following notation. We will refer to $p(X = x, Y = y)$, and $p(X = x|Y = y)$ as the joint and the conditional densities with respect to some reference measure μ, of the random variables X and Y, evaluated at some points x, y. To make the notation more compact, given the sequences of hidden states s_0^n, and observed variables y_0^n, we will write $p(s_k^j, y_h^m)$ and $p(s_k^j|y_h^m)$, when meaning $p(S_k^j = s_k^j|Y_h^m = y_h^m)$ and $p(S_k^j = s_k^j|Y_h^m = y_h^m)$ respectively.

We first define what are the inference problems that we are interested in before deriving the basic results that form the core of the techniques discussed in the following section.

Definition 2.2.2 (Smoothing, Filtering, Prediction)**.** The problems that we are aiming to solve can all be formulated as the the problem of computing a specific conditional density of the hidden states given the sequence, with respect to the product of reference measures of the transition kernel P_θ, for each element of the sequence:

- Filtering: is the computation of $p(s_n|y_0^n)$, for $n \geq 0$;
- Smoothing is the computation of $p(s_k|y_0^n)$, for $n \geq k \geq 0$;
- Prediction: is the computation of $p(s_{n+p}|y_0^n)$, for $n, p \geq 0$.

Smoothing can thus be interpreted now as the problem of computing the distribution of a past state. Filtering is then computing the distribution of the present state. Prediction is finally computing the distribution of a future state .

In order to derive explicit algorithms to compute these quantities we first notice that for $j > 0$, the conditional density of S_j given $Y_0^n = y_0^n$ is proportional to the joint density of S_j and Y_0^n:

$$p(S_j|Y_0^n = y_0^n) \propto p(S_j, Y_0^n = y_0^n) \tag{2.2.2}$$

Second we recall that we assumed $\mathbb{S}$ to be finite thus the following property holds:

$$\sum_{s \in \mathbb{S}} p(S_j = s|Y_0^n = y_0^n) = 1 \tag{2.2.3}$$

Thus we can now just focus on the computation of $p(s_j, y_0^n)$ for $s_j \in \mathbb{S}$, and then we will only need to normalize the results to retrieve $p(s_j|y_0^n)$.

2.3 Algorithms for the inference problems

All the classical inference problems are computationally straightforward since the distribution is singly-connected, an can be solved in linear time using some so-called message passing algorithms.

Filtering

We concentrate first on the problem of filtering. The key observation is that through marginalization we can rewrite $p(s_j, y_0^n)$ as:

$$p(s_j, y_0^n) = \sum_{s_{j-1} \in \mathbb{S}} f(y_n, \beta_{s_j}^*) Q_{\theta^*}(s_{j-1}, s_j) p(s_{j-1}, y_0^{n-1}) \tag{2.3.1}$$

We now define:

$$\alpha_t(s) = p(S_t = s, Y_0^t = y_0^t) \tag{2.3.2}$$

where

$$\alpha_0(s) = p(S_0 = s, Y_0 = y_0) = f(y_0, \beta_s^*)\pi(s) \tag{2.3.3}$$

We observe that marginalizing in the same way as in the previous equation we can derive a recursive equation for the α_t as:

$$\alpha_t(s) = \sum_{s' \in \mathbb{S}} f(y_t, \beta_s^*) Q_{\theta^*}(s', s)\alpha_{t-1}(s') \tag{2.3.4}$$

we can rewrite as well the equation for the filtering problem as:

$$p(S_n = s, Y_0^n = y_0^n) = \sum_{s' \in \mathbb{S}} f(y_n, \beta *_s) Q_\theta(s', s)\alpha_{n-1}(s') \tag{2.3.5}$$

Thus in order to solve the filtering problem, one only needs to compute the values of $\alpha_t(s)$ for every s in $\mathbb{S}$, and $t \leq n$ through the recursion, and normalize the result.

Smoothing

For the smoothing problem instead we observe that thanks to the dependence structure of the model, the distribution $p(S_k, Y_0^n)$ can be written as :

$$p(s_k, y_0^n) = p(s_k, y_0^k)p(y_{k+1}^n | s_k) \equiv \alpha_k(s_k)\beta_k(s_k) \tag{2.3.6}$$

This simple splitting of the multiple integration in 2.3.6 constitutes the forward-backward decomposition.

We observe that also $\beta_t(s)$ satisfies a recursion given by:

$$\beta_t(s) = \sum_{s' \mathbb{S}} f(y_t, s) Q_\theta(s, s')\beta_{t+1}(s') \tag{2.3.7}$$

And β_n is naturally defined to be 1. Thus one can compute $p(s_k | y_0^n)$ as:

$$p\left(s_k \mid y_{1:n}\right) \equiv \gamma\left(s_k\right) = \frac{\alpha\left(s_k\right)\beta\left(s_k\right)}{\sum_{s_k \in \mathbb{S}} \alpha\left(s_k\right)\beta\left(s_k\right)} \tag{2.3.8}$$

Together the $\alpha - \beta$ recursions are called the Forward-Backward algorithm.

Most likely joint state

One problem related to smoothing is given by finding the most likely path s_0^n of $p\left(s_0^n \mid y_0^n\right)$, also known as Viterbi alignment.

$$p\left(s_1^n, y_1^n\right) = \prod_t f\left(y_t, \beta_{s_t}^*\right) Q_{\theta *}\left(s_{t-1}, s_t\right)$$

The problem can be easily solved, by exploiting the properties of $\alpha_t(s)$ and $\beta_t(s)$ by means of dynamic programming. In fact if we consider the problem of maximizing only the probability of the last state, given any past sequence of states, we get:

$$\max_{s_T}\prod_{t=1}^{T} p\left(y_t \mid s_t\right) p\left(s_t \mid s_{t-1}\right) = \left\{\prod_{t=1}^{T-1} p\left(y_t \mid s_t\right) p\left(s_t \mid s_{t-1}\right)\right\} \underbrace{\max_{s_T} p\left(y_T \mid s_T\right) p\left(s_T \mid s_{T-1}\right)}_{\mu(s_{T-1})}$$

$\mu\left(s_{T-1}\right)$ depends thus, only from the penultimate timestep. We can continue in this manner, defining the recursion

$$\mu\left(s_{t-1}\right) = \max_{s_t} p\left(y_t \mid s_t\right) p\left(s_t \mid s_{t-1}\right) \mu\left(s_t\right), \quad 2 \leq t \leq T$$

with $\mu\left(s_T\right) = 1$. This means that the effect of maximising over $h_2, \ldots, s_T$ is compressed into $\mu\left(s_1\right)$ so that the most likely state s_1^* is given by

$$s_1^* = \underset{s_1}{\operatorname{argmax}} p\left(y_1 \mid s_1\right) p\left(s_1\right) \mu\left(s_1\right)$$

Once computed, backtracking gives

$$s_t^* = \underset{s_t}{\operatorname{argmax}} p\left(y_t \mid s_t\right) p\left(s_t \mid s_{t-1}^*\right) \mu\left(s_t\right)$$

This way of solving the most likely hidden state problem, is also called Viterbi algorithm.

Prediction

The p-step ahead predictive distribution is finally given by

$$p\left(s_{t+p} \mid y_{1:t}\right) = \sum_{s_{t+p},\ldots,s_t} p\left(s_{t+p} \mid s_{t+p-1}\right) \ldots p\left(s_{t+1} \mid s_t\right) p\left(s_{t+1} \mid s_t\right) p\left(s_t \mid y_{1:t}\right)$$

That is for any given $y_0^n \in \mathbb{Y}^{n+1}$, the p-step predictive distribution may be obtained by marginalization of the joint distribution with respect to all variables s_k except the last one (the one with index $k = n + p$). Chapman-Kolmogorov equations are applied in order to compute the distribution until present time, where the remaining term represents a smoothing problem that is solved with the algorithm that we derived in the previous sections.

2.4 Estimation of Parameters and EM

Given that the state variables are hidden, the likelihood of the model will be a function of only the observable variables, computed through marginalization, by exploiting the past dependence properties of the model.

Definition 2.4.1 (Likelihood)**.** The likelihood of the observations is the probability density function of $Y_0, Y_1, \ldots, Y_n$ with respect to μ_n defined, for all $(y_0, \ldots, y_n) \in \mathbb{Y}^{n+1}$, by

$$\mathcal{L}_{\theta,n}\left(y_0^n\right) = \sum_{(s_1,\ldots,s_n)\in\mathbb{S}^n} \pi(s_0) P_\theta(s_0, y_0) \prod_{t=1}^{n} Q_\theta(s_{t-1}, s_t) P_\theta(s_t, y_t) \tag{2.4.1}$$

In addition,

$$\ell_{\theta,n}\left(y_0^n\right) \stackrel{\text{def}}{=} \log \mathcal{L}^{\theta}_{\pi,T}\left(y_0,\dots,y_n\right)$$

is referred to as the log-likelihood function.

We want to estimate the true parameter of our model given a sequence of observation, by means of the maximum likelihood method. However maximizing directly the quantity $\ell_{\theta,n}$, might be hard due to the presence of the summation inside the logarithm. One common method to tackle this problem is through the use of the Expectation Maximization algorithm.

Let's define the complete likelihood as the likelihood assuming that we observed the hidden states:

$$\mathcal{L}^{cmp}_{\theta,n}\left(s_0^n, y_0^n\right) = Q_\theta(s_0,s_1)P_\theta(s_0,y_0)\prod_{t=1}^{n} Q_\theta(s_{t-1},s_t)P_\theta(s_t,y_t) \tag{2.4.2}$$

We observe that we can interpret the quantities that we defined as $\mathcal{L}_{\theta,n}\left(y_0^n\right) = p(y_0^n|\theta)$ and $\mathcal{L}^{cmp}_{\theta,n}\left(s_0^n, y_0^n\right) = p(s_0^n, y_0^n|\theta)$

Then Expectation maximisation algorithm is:

Algorithm 1 EM algorithm

initialize: θ_0

- **E-Step:** Given the current estimate of the model parameters $\theta^{(t)}$, compute

$$\mathcal{Q}(\theta|\theta^{(t)}) \equiv \mathbb{E}_{S0^n|Y_0^n,\theta^{(t)}}[\log p(Y_0^n, S_0^n|\theta)].$$

- **M-Step:** Find the new estimate of the model parameters

$$\theta^{(t+1)} = \arg\max_\theta \mathcal{Q}^{(t+1)}(\theta|\theta^{(t)}).$$

where $\mathcal{Q}^{(t+1)}(\theta|\theta^{(t)})$ is often referred to as the intermediate quantity of EM algorithm.

The key observation to understand the role of the intermediate quantity is that since $p(y_0^n, s_0^n|\theta) = p(s_0^n|y_0^n,\theta)p(y_0^n,|\theta)$, then for any choice of parameters θ, then taking the expectation with respect to the variable $S_0^n|Y_0^n,\theta^{(t)}$, we get:

$$\mathcal{L}_{\theta,n}\left(s_0^n, y_0^n\right) = \sum_{s_0^n\in\mathbb{S}^n} \log p(y_0^n, s_0^n|\theta)p(s_0^n|y_0^n,\theta^{(t)}) - \sum_{s_0^n\in\mathbb{S}^n} \log p(s_0^n|y_0^n,\theta)p(s_0^n|y_0^n,\theta^{(t)}) \tag{2.4.3}$$

$$\equiv \mathcal{Q}(\theta|\theta^{(t)}) - \mathcal{H}(\theta|\theta^{(t)}) \tag{2.4.4}$$

We now observe that $\mathcal{H}(\theta|\theta^{(t)}) - \mathcal{H}(\theta^(t)|\theta^{(t)})$ is the Kullback-Leiber divergence of $p(s_0^n|y_0^n,\theta)$ with respect to $p(s_0^n|y_0^n,\theta^{(t)})$ and thus is always ≤ 0.

We can thus conclude that

$$\mathcal{L}_{\theta,n}\left(s_0^n, y_0^n\right) - \mathcal{L}_{\theta^(t),n}\left(s_0^n, y_0^n\right) \mathcal{Q}^{(t+1)}(\theta|\theta^{(t)}) \geq \mathcal{Q}(\theta|\theta^{(t)} - \mathcal{Q}(\theta^(t)|\theta^{(t)} \tag{2.4.5}$$

We now explicitate the relationship that connects our parameters θ^* and the Likelihood 2.4.1, in order to make the step of the algorithm more explicit in our framework:

$$\mathcal{L}_{\pi,T}(y_0^T, s_0^T) = \sum_{(s_1,\dots,s_T)\in\mathbb{S}^n} \pi_i^* p\left(y;\beta_{s_1}^*\right) \cdot \prod_{t=1}^{T} a_{s_{t-1},s_t}^* p\left(y;\beta_{s_t}^*\right) \tag{2.4.6}$$

And the complete likelihood:

$$\mathcal{L}_{\pi,T}^{comp}(y_0^T, s_0^T) = \prod_{i=0}^{k} [\pi_i^* p\left(y;\beta_i^*\right)]^{z_{i,1}} \cdot \prod_{t=1}^{T} \cdot \prod_{j,i=1}^{k} [a_{i,j}^*]^{z_{j,t}\cdot z_{i,t-1}} \left[p\left(y;\beta_j^*\right)\right]^{z_{j,t}} \tag{2.4.7}$$

where

$$z_{i,t} = \begin{cases} 1 \text{ if } s_t = i \\ 0 \text{ otherwise} \end{cases} \tag{2.4.8}$$

So the complete log-likelihood is:

$$\ell_{\theta,n}^{comp}\left(y_0^n\right) = \sum_{i=1}^{C} z_{i,1} log(\pi_i) + \sum_{t=2}^{T}\sum_{j,i=1}^{C} z_{j,t}\cdot z_{i,t-1} log A_{ji} + \sum_{t=1}^{T}\sum_{j=1}^{C} z_{j,t} log(f_j(y_t;\beta_j)) \tag{2.4.9}$$

Let's consider now the E-step of EM algorithm. Given that the only random variables inside the complete likelihood are the z_i's then taking the expectation is equivalent to compute the following:

$$\gamma_t(i) := E^\theta[z_{i,t}] = P^\theta(s_t = i|Y) \qquad \gamma_{t,t-1}(j,i) := E^\theta[z_{j,t}\cdot z_{i,t-1}] = P^\theta(s_t = j, s_{t-1} = i|Y) \tag{2.4.10}$$

which we recall can be computed in linear time using the forward-backward algorithm that we introduced previously.

On the other hand M-step we have to find $\bar{\theta}$ that provides the maximum likelihood, given the $\gamma_t(i)$ and $\gamma_{t,t-1}(j,i)$:

$$\bar{\theta} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{C} \gamma_1(i) log(\pi_i) + \sum_{t=2}^{T}\sum_{j,i=1}^{C} \gamma_{t,t-1}(j,i) log(A_{ji}) + \sum_{t=1}^{T}\sum_{j=1}^{C} z_{j,t} log(f_j(x_t,\beta_j)) \tag{2.4.11}$$

Thus we can look at each parameter separately:

$$\bar{\pi} = \underset{\pi}{\operatorname{argmin}} \left\{ \sum_{i=1}^{C} \gamma_1(i) \cdot log(\pi_i) \middle| \sum_{i=1}^{C} \pi_i = 1 \right\} \tag{2.4.12}$$

$$\bar{A} = \underset{A}{\operatorname{argmin}} \left\{ \sum_{t=1}^{T} \sum_{j,i=1}^{C} \gamma_{t,t-1}(j,i) \cdot log(A_{ji}) \middle| \sum_{j=1}^{C} A_{i,j} = 1 \text{ for } i = 1, \ldots, C \right\} \tag{2.4.13}$$

$$\bar{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{t=1}^{T} \sum_{j=1}^{C} \gamma_t(i) \cdot log(f_j(x_t, \beta_j)) \tag{2.4.14}$$

So $\bar{\pi}$ and $\bar{A}$ can be easily computed by standard methods such as Lagrange multipliers, while the computation of the maximum of the emissions is an optimization problem that depends on the expression of the density.

Chapter 3

Our model

In this section we combine hidden Markov models and quantile regression by following the same approach as [5]. There are several caveats to pay attention to and we will address most of them. Finally in every section we provide the code that was written to implement the model.

3.1 Combining quantile regression and HMM

Let $y_t, t = 1, \ldots, T$ denote a real-value observation and $x_t, t = 1, \ldots, T$, be our covariates. Let's denote with S_t the state of a finite-state semi-Markov chain, defined on the state space $\{1, ..., k, ..., K\}$ at time t. A Markov switching quantile regression model is a particular kind of Hidden Markov model where quantile regression is embedded in its emission distribution. Let τ be the quantile that we are trying to predict. Then we assume the following model on the data:

$$y_t = \beta_k(\tau)x_t + \epsilon_t(\tau) \tag{3.1.1}$$

with $\beta_k(\tau)$ being a vector of state-specific regression coefficients, $x_t = (1, x_t)$ and $\epsilon_t(\tau)$ is the error term whose τ quantile conditional to $\{x_1, \ldots, x_t\}$ equals zero.

Then we code quantile regression inside the HMM by means of a particular form of the Asymmetric Laplace distribution:

$$f_{QR}(y|\beta, \sigma, x, \tau) = \frac{\tau(1-\tau)}{\sigma} \exp\left(-\rho_\tau\left(\frac{y - \beta^T x}{\sigma}\right)\right) \tag{3.1.2}$$

where:

$$\rho_\tau(v) = \begin{cases} \tau v \text{ for } v \geq 0 \\ (\tau - 1)v \text{ for } v < 0 \end{cases} \tag{3.1.3}$$

and

$f_{QR}(y|\beta, \sigma, x)$ is the PDF distribution,
τ is the quantile,
β if the vector of the coefficient of the regression,
x is a vector that represent the covariates
σ is a scale parameter.

The main reason to pick such a distribution is to compute, during the maximization step, the following maximum:

$$\max_{\beta \in \mathbb{R}^p} - \sum_{i=1}^{n} \rho_\tau(y_i - \xi(x_i, \beta)) \tag{3.1.4}$$

Observe that this formulation is equivalent to computing the maximum likelihood estimator under the following model for the response variable:

$$y_t = \beta x_t + u_t \tag{3.1.5}$$

where u_t is a asymmetric Laplace variable with density:

$$f(u|\beta, \sigma, \tau) = \frac{\tau(1-\tau)}{\sigma} \exp\left(-\rho_\tau\left(\frac{u}{\sigma}\right)\right) \tag{3.1.6}$$

The HMM was implemented using the hmmlearn library [6] of Python. This library allowed to exploit all the algorithms already implemented inside the library with the only thing left to manage being the new probability distribution function. The asymmetric Laplace distribution was already implemented in Python, though in a different formulation, its probability density function is defined as follows:

$$f_{AL}(y; \mu, \kappa, \lambda) = \frac{1}{\lambda(\kappa + \kappa^{-1})} \exp\left(\phi_\kappa\left(\frac{y-\mu}{\lambda}\right)\right) \tag{3.1.7}$$

where:

$$\phi_\kappa(x) = \begin{cases} -x\kappa \text{ for } x \geq 0 \\ x/\kappa \text{ for } x < 0 \end{cases} \tag{3.1.8}$$

and

$f_{AL}(x; \mu, \sigma, \lambda)$ is the PDF of the asymmetric Laplace distribution,
κ controls the asymmetry of the distribution,
μ is the location parameter (median),
λ is the scale parameter (spread).

The parameter κ determines whether the distribution is left-skewed ($\kappa > 1$) or right-skewed ($0 < \lambda < 1$). When $\kappa = 1$, it reduces to the standard Laplace distribution.

With some easy calculations, we get the same distribution as f_{QR} if:

$$\kappa = \sqrt{\frac{\tau}{1-\tau}} \tag{3.1.9}$$

$$\sigma = \lambda\sqrt{\tau(1-\tau)} \tag{3.1.10}$$

$$\frac{\beta x}{\sigma} = \frac{\mu}{\lambda} \tag{3.1.11}$$

We already highlighted that the betas will be calculated during the maximization step through quantile regression, and κ is already completely determined by the quantile.

We are left with σ that is calculated as follows:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{C}\gamma_t(k)\cdot(\rho_\tau(y_t - \beta_k(\tau)x_t^*) \tag{3.1.12}$$

We now show how we implemented the functions. After defining the QRHMM subclass we wrote the following methods:

Example of code

```
[ ]:   def _compute_likelihood(self, X):
            '''Compute the likelihood of observations given the model␣
    ↪parameters.

        Args:
            X (np.array): Input data matrix

        Returns:
            np.array: Matrix of likelihoods for each observation and␣
    ↪component.
        '''
            # Initialize an empty matrix to store the likelihoods
            probs = np.empty((len(X), self.n_components))
            # Iterate over each component
            for c in range(self.n_components):
                  # Compute the likelihood for each observation
                probs[:, c] = np.array([al.pdf(X[:,0], self.k[q], loc=self.
    ↪betas_[c,0,q]+np.matmul(self.betas_[c, 1:,q], (X[:, 1:]).T)  ,␣
    ↪scale=self.scale_[q]) for q in range(len(self.quantile))]).prod(0)
            return probs
```

Example of code

```
[ ]:    def _compute_log_likelihood(self, X):
            '''Compute the loglikelihood of observations given the model␣
↪parameters.

    Args:
        X (np.array): Input data matrix

    Returns:
        np.array: Matrix of likelihoods for each observation and␣
↪component.'''
             # Initialize an empty matrix to store the loglikelihoods
            logprobs = np.empty((len(X), self.n_components))
             # Iterate over each component
            for c in range(self.n_components):
                # Compute the loglikelihood for each observation
                logprobs[:, c] = np.array([al.logpdf(X[:,0], self.k[q],␣
↪loc=self.betas_[c, 0,q]+np.matmul(self.betas_[c, 1:,q], (X[:, 1:]).T) ␣
↪, scale=self.scale_[q]) for q in range(len(self.quantile))]).sum(0)
            return logprobs
```

Example of code

```
[ ]:    def compute_betas(self, y, X, weights):
            """
    Compute quantile regression coefficients (betas) for each regime␣
↪according to self.type_of_reg
    and saves the respective quantile regression model for each regime

    Parameters:
    - y (np.array): The target variable for regression.
    - X (np.array): The matrix of features.
    - weights (array-like): Posterior probabilities for each regime.

    Returns:
    - betas (np.array): Regression coefficients for each regime.
            """
            # Initialize an array to store regression coefficients for each␣
↪regime and quantile
            betas = np.zeros([self.n_components, self.n_features, len(self.
↪quantile)])

            # Iterate through each regime
            for j in range(self.n_components):
                for q in range(len(self.quantile)):

                     # Use QuantileRegressor for linear quantile regression
                    if(self.type_of_reg=='linear'):
                        qr = QuantileRegressor(quantile=self.quantile[q],␣
↪alpha=self.alpha,solver='highs')
```

```
                    quant_reg_result = qr.fit( X, y,␣
↪sample_weight=weights[:,j])
                    betas[j,0,q]=quant_reg_result.intercept_
                    betas[j,1:,q]=quant_reg_result.coef_
                    self.qrmodel[j]= qr
        ⋮
        return betas
```

We remark that we omitted the other options for "type_ of_ reg" as they are going to be defined in the following chapter that deals with multifrequency data.

Example of code

```
[ ]:    def compute_scale(self, y, X, weights):
            '''Compute the scale parameter of the asymmetric Laplace␣
↪distribution formulation of quantile regression.

        Args:
            y (np.array): Target variables.
            X (np.array): Feature matrix.
            weights (np.array): Weights for each element in the loss function.

        Returns:
            np.array: Scale parameter for each quantile in the asymmetric␣
↪Laplace distribution formulation of quantile regression.
        '''
            # Initialize an array to store the scale parameter
            scale_=np.zeros(len(self.quantile))
            for q in range(len(self.quantile)):
                 # Calculate the estimation of the quantile
                aux= np.tile(self.betas_[:,0,q], (len(y),1)).T + np.
↪matmul(self.betas_[:, 1:,q], (X).T)
                # Compute the quantile loss
                aux_2=rho(np.tile(y, (self.n_components,1))-aux, self.
↪quantile[q])
                # Calculate the scale parameter using weighted residuals
                scale_[q]= np.sum(np.multiply(aux_2, weights.T)) / ((len(y))␣
↪*((1-self.quantile[q])*self.quantile[q])**0.5)
            return scale_
```

3.2 Initialization

A non trivial problem found in training hidden Markov models, is sensitivity to changes in initial conditions of Expectation Maximization. This was evident during preliminary tests, where we noticed that our model choice of regimes could be particularly sensitive to initialisation, leading not only to a shift in interpretation of the hidden classes but also to a completely different set of parameters of our regression model. Not of lesser importance is also the issue of the speed of convergence: points closer to the optimum, will also amount to less iterations of the EM algorithm. Due to these issues, initialization becomes a crucial step in the algorithm and we developed multiple methods to tackle the problem.

First, we highlight the general strategy that we adopted for initializing the parameter of the EM algorithm:

- the first step consist of getting an initial state partition $(S_t)_{t=0}^T$ of our data, through a method of our choice.
- In this step we only fix the parameters of the hidden Markov chain. From the partition $(S_t)_{t=0}^T$ is then computed the empirical probability of each state. This will be the way we initialise π. In order to get the parameters of the transition matrix A we use again our initial state partition and compute the empirical transition probabilities
- the remaining parameters, the ones of the emission distributions, are then computed through the maximisation step of the expectation maximisation algorithm

Random approach

One first natural way to obtain a state partition is through a random approach following [5]. The algorithm performs the classification as follows:

- following a uniform distribution in the number of hidden states, each observation is classified
- then all the parameters of the model are retrieved through a step of maximization
- finally 2 rounds of EM are performed using the current parameter as initialization and a likelihood score is computed

The algorithm is repeated a number of times and the initialization is thus given by the parameters that retrieved the highest score at the end of the second step of the procedure.

Example of code

[ ]:
```
    def rand_init(self,X,  lengths=None):
        '''Initialize model parameters using a random labeling approach.

    Args:
        X (np.array): Input data matrix.

    Returns:
        None
    '''

        # Extract the target variable from the input data
        y=X[:,0]

        # Iterate through multiple random initializations and select the␣
↪best model
        models=[]
        for j in range(100):
            if (j%100==0):
                print(j)
            # Initialize transition matrix and starting probabilities␣
↪based on random labels
```

```
            predictions=np.random.randint(0,self.n_components,y.shape[0],␣
↪)
            self.transmat_ = self.get_trans_emp( predictions)
            self.startprob_ = self.get_start_emp( predictions)

            ⋮

            #Initialize the betas with the empirical results, the␣
↪posteriors are 1 for the correct label and zeros otherwise
            pred_mat = np.zeros((len(predictions), self.n_components))
            for j in range(self.n_components):
                pred_mat[:, j] = (predictions == j)

            self.betas_= self.compute_betas(X[:,0],X[:,1:], pred_mat)

            self.scale_= self.compute_scale(X[:,0],X[:,1:], pred_mat)

            # Fit the model for a small number of steps and store it␣
↪along with its score
            model=self.init_fit(X,lengths)
            models.append([model, model.score(X)])

        # Select the model with the highest likelihood score
        max_model= max(models, key=lambda x: x[1])

        # Update the model parameters with the best model
        self.startprob_= max_model[0].startprob_
        self.transmat_= max_model[0].transmat_
        self.betas_= max_model[0].betas_
        self.scale_= max_model[0].scale_
```

Example of code

[ ]:
```
    def init_fit(self, X, lengths=None):
        '''Fit the model parameters using the Expectation-Maximization␣
↪(EM) algorithm for 2 steps.

    Args:
        X (np.array): Input data matrix with shape.
        lengths ( optional): variable inherited from the baseHMM class,␣
↪it will only be None.

    Returns:
        self: Updated model instance after fitting.
    '''

        # If lengths are not provided, use the entire dataset as a single␣
↪segment
        if lengths is None:
```

```
            lengths = np.asarray([X.shape[0]])


         # Check the validity of the model and reset the monitor
        self._check()
        self.monitor_._reset()


        # Iterate through the Expectation-Maximization (EM) algorithm␣
↪steps
        for iter in range(2):
             # Perform the E-step and compute the current log probability
            stats, curr_logprob = self._do_estep(X, lengths)

            # Compute the lower bound before updating model parameters
            lower_bound = self._compute_lower_bound(curr_logprob)

             # Update model parameters in the M-step
            self._do_mstep(stats)

            # Check for convergence based on the monitor
            if self.monitor_.converged:
                break

             # Warn if some rows of transmat_ have zero sum
            if (self.transmat_.sum(axis=1) == 0).any():
                _log.warning("Some rows of transmat_ have zero sum␣
↪because no "
                             "transition from the state was ever observed.
↪")
        return self
```

Although the initial concept seemed promising, real-world implementation revealed significant performance issues as the approach was computationally expensive.

k-means

The second approach that we implemented is based on the usage of a clustering algorithm performed on the $(y_t)_{t=0}^T$ variables. One first observation is that performing clustering on the whole dataset $(x_t, y_t)_{t=0}^T$ is also possible, but might be both computationally more expensive and possibly be susceptible to the curse of dimensionality for very wide datasets. To perform this task we chose k-means due to its simplicity and computational efficiency.

The k-means is clustering algorithm based on solving an optimization problem objective function. In k-means the data is partitioned into disjoint sets $C_1, \ldots, C_k$ where each C_i is represented by a centroidμ_i . It is assumed that the input set $\mathcal{Y}$ is embedded in some larger metric space $(\mathcal{Y}', d)$ (so that $\mathcal{Y} \subseteq \mathcal{Y}'$) and centroids are members of $\mathcal{Y}'$. The k-means objective function measures the squared distance between each point in $\mathcal{Y}$ to the centroid of its cluster. The centroid of C_i is defined to be:

$$\mu_i(C_i) = \underset{\mu \in \mathcal{X}'}{\operatorname{argmin}} \sum_{x \in C_i} d(x, \mu)^2 \tag{3.2.1}$$

Then, the k-means objective is

$$G_{k\text{-means}}((X, d), (C_1, \ldots, C_k)) = \sum_{i=1}^{k} \sum_{x \in C_i} d(x, \mu_i(C_i))^2 \tag{3.2.2}$$

Finding the optimal k-means solution is often computationally infeasible (the problem is NP-hard), thus the following iterative procedure is referred as the k-means algorithm:

Algorithm 2 k-means

input: $\mathcal{X} \subset \mathbb{R}^n$; Number of clusters k
initialize: Randomly choose initial centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
repeat until convergence:

- $\forall i \in [k]$ set $C_i = \{\mathbf{x} \in \mathcal{X} : i = \operatorname{argmin}_j \|\mathbf{x} - \boldsymbol{\mu}_j\|\}$ (break ties in some arbitrary manner)
- $\forall i \in [k]$ update $\boldsymbol{\mu}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$

The implementation of the k-means algorithm was already present in Python, in the library sklearn.cluster as the function Kmeans. Preliminary tests showed little difference between this approach and the random one in terms of convergence of the EM algorithm. However k-means was significantly faster, and was the main algorithm that was used whenever no more data was known.

Data driven approach

The final approach is derived simply by incorporating external information into the model through a state partition. For example, when dealing with economic data, the state partition can be given by a vector of 0 and 1, that represent if at a specific point in time economy was either or not in a recession state. In theory, this approach is the most effective, provided that the correct variable is utilized. It may lead to a more robust initialization for the model and, simultaneously, provide us with a reference variable to compare the model's results and facilitate their interpretation. Assuming this variable exists, this method will always be the preferred choice for training our models.

3.3 Quantile prediction

After fitting the Markov switching quantile regression model, one naturally wishes to be able to use the model to compute quantile estimates, especially in order to perform tests to assess the performance of the model. For this purpose we developed two strategies that depend on the assumptions on present knowledge.

If we assume that we know the whole sequence of observable variables and want to provide the model estimates of the quantiles of past data, then our approach consists in using the data and Viterbi algorithm to compute the most likely sequence of hidden states and from that perform the prediction using the fitted coefficients for each hidden state.

Example of code

```
[ ]:    def insample_predict(self, Y ,X):
            """
        Predict the quantile of Y given X, utilizing Y to guess the regime␣
    ↪probabilities and performing regression based on the selected regime.

        Parameters:
        - Y (np.array): The target variable for which the quantile is to be␣
    ↪predicted.
        - X (np.array): The features used to predict the quantile of Y.

        Returns:
        - predictions (np.array): Predicted quantiles for each observation in␣
    ↪Y."""

            t=X.shape[0]
            data=np.concatenate((Y.reshape(-1,1), X),axis=1)

            predictions=np.zeros(t)
            state_prob=self.predict_proba(data)

            for j in range(t):
                predictions[j] = state_prob[j,0]*(np.dot(X[j,:], self.
    ↪betas_[0,1:,0]) + self.betas_[0,0,0])+state_prob[j,1]*(np.dot(X[j,:],␣
    ↪self.betas_[1,1:,0]) + self.betas_[1,0,0])

            return predictions
```

If instead we are interested in predicting the p future-lagged quantile while being at time t in the present, the approach is different: first we perform a p-step prediction for the hidden state. Once we compute the distribution of s_{t+p}, we select the hidden state s that is the most likely and then we use quantile regression coefficients for state s to compute the estimate for the quantile. Assuming we always want to perform just a p future lagged estimate, in order to compute the $k + p$ future estimate, we can now assume to know all the information up until time $t + k + p$. This embodies real life situations, where one has information at a certain moment in time and wants to know how lagged information influences future outcomes.

Example of code

[ ]:
```
def time_horiz_predict(self, past_sequence, X, Y, time_offset=1):
    """
   Predict the future lags (quantiles of the target variable)␣
↪conditioned on the features for the given past sequence by guessing␣
↪the regime          first and then performing the prediction using the␣
↪quantile linear regression framework of the found regime. To retrieve␣
↪the regime we us all      past information (Y included) to estimate␣
↪the probability vector of the regimes at lag "timeoffset" with respect␣
↪to the time of the                 prediction, using the Hidden Markov␣
↪model framework. Then we multiply it by the transition matrix and we␣
↪picked the regime that has maximum        probability.



   Parameters:
   - past_sequence (np.array): The past sequence of observations, the␣
↪first column is the target variable.
   - X (np.array): The known features for which future lags are to be␣
↪predicted.
   - Y (np.array): The target variable for which the quantile is to be␣
↪predicted.
   - time_offset(int): lag between last information used for the␣
↪prediction and the time of the prediction

   Returns:
   - predictions (np.array): Predicted future lags for each row in X.

   Achtung! This function differs from running the whole model untile␣
↪time t. When computing the state probabilities
   for the past_sequence, we also include information about the target␣
↪variable. From then we keep adding future information to the␣
↪past_sequence
   to make future predictions.

   """
        k=X.shape[0]
        predictions=np.zeros((k))
        new_past_sequence=past_sequence
        time_offset_transmat=np.linalg.matrix_power(self.transmat_,␣
↪time_offset)
        states=[]
        for j in range(k):
            prob=np.matmul(self.
↪predict_proba(new_past_sequence)[-time_offset], time_offset_transmat)
            states.append(np.argmax(prob))
            predictions[j]=np.matmul(self.betas_[states[-1],1:,0], X[j,:].
↪T)+self.betas_[states[-1],0,0]
            new_past_sequence=np.vstack((past_sequence, np.
↪concatenate((Y[:j].reshape(-1,1), X[:j,]),axis=1) ))

        return predictions, np.array(states)
```

Chapter 4

Multifrequency

One common occurrence when dealing with economical data, is that the dataset contains many variables that are sampled at different frequencies, such as daily, weekly, monthly or quarterly. This translates into a choice for the researcher. On the one hand, the variables that are available at high frequency contain potentially valuable information. On the other hand, the researcher cannot use this high frequency information directly if some of the variables are available at a lower frequency, because most time series regressions involve data sampled at the same interval. The common solution in such cases is to "pre-filter" the data so that the all the variables are available at the same frequency. In this section we present another way of dealing with the problem with MIDAS (MIxed Data Sampling regression) and apply it to our Markov switching quantile regression model. By using this method we will no longer be able to formulate quantile regression as a linear programming optimization problem and for this reason we will introduce two alternative optimization algorithms: Adam and Nelder-Mead.

4.1 MIDAS

When dealing with time series regression, a situation that is often encountered is to have relevant information as high frequency data, while the variable of interest is sampled at a lower frequency. Suppose we have a stream of data Y_t, $t \in \mathbb{N}$ sampled at some fixed frequency. Suppose also that we have another stream of data $X_t^{(m)}$ that is sampled m times faster, that is in the interval of time $[t, t+1)$ there are exactly m observation of X that were collected, at intervals of length $\frac{1}{m}$. Simple linear MIDAS regression presents as follows:

$$Y_t = \beta_0 + \beta_1 \mathbf{B}\left(L^{\frac{1}{m}}\right) X_{t-1}^{(m)} + \epsilon_t \tag{4.1.1}$$

where ϵ_t is an error term and $\mathbf{B}(x) = \sum_{j=0}^{K} B(j)x^j$ is a polynomial of degree K, whose weights sum to one (this condition is imposed because we added the parameter β_1). $L^{\frac{1}{m}}$ is the lag operator defined as:

$$L^{\frac{1}{m}} X_t^{(m)} = X_{t-\frac{j}{m}}^{(m)} \tag{4.1.2}$$

We observe how this framework naturally extends to our quantile regression problem by simply assuming that ϵ_t's distribution has quantile at level τ equal to 0. Then generalization for more X-s sampled at multiple time frequencies is straightforward.

The assumption is for $\mathbf{B}$ to be of finite order, however, even if the number of parameters $B(j)$'s in the polynomial $\mathbf{B}(L^{1/m})$ is finite, it might be quite large. To capture daily fluctuations in the process over the last, say, 6 months, we would need to estimate 6×22, or 132 $B(k)$ parameters (assuming 22 trading days a month). To account for daily data over the last year, we would need approximately 264 parameters. It becomes rapidly clear that one must impose some structure upon the b_k 's in order to get sensible results.

Often what is done is to aggregate the data at higher frequency in order to reduce all data to the same frequency, and then fitting a standard regression model on the pre-filtered data. This practice can be interpreted as imposing some structure on the polynomial $B(x)$, thus MIDAS can be considered a generalization of this common practice.

Let's write $\mathbf{B}(x) = \mathbf{B}(x;\theta) = \sum_{j=0}^{K} B\left(j,\theta\right) x^j$ in order to highlight the dependence of the polynomial from learned parameters. One common way to tackle the parameter proliferation issue and impose some structure on the $B(x;\theta)$ is the following as suggested by [7]:

$$B\left(j,\theta\right) = \frac{e^{\theta_1 j + \cdots + \theta_Q j^Q}}{\sum_{k=1}^{K} e^{\theta_1 k + \cdots + \theta_Q k^Q}} \tag{4.1.3}$$

which we call the "Exponential Almon Lag," since it is related to "Almon Lags" that are popular in the distributed lag literature. The function $B(j;\theta)$ is known to be quite flexible and can take various shapes with only a few parameters.

Let's consider the case of $Q = 2$. Then we have:

$$B\left(j,\theta\right) = \frac{e^{\theta_1 j + \theta_2 j^2}}{\sum_{k=1}^{K} e^{\theta_1 k + \theta_2 k^2}} \tag{4.1.4}$$

Even though we are dealing with just two parameters it's possible to express a wide variety of functions.

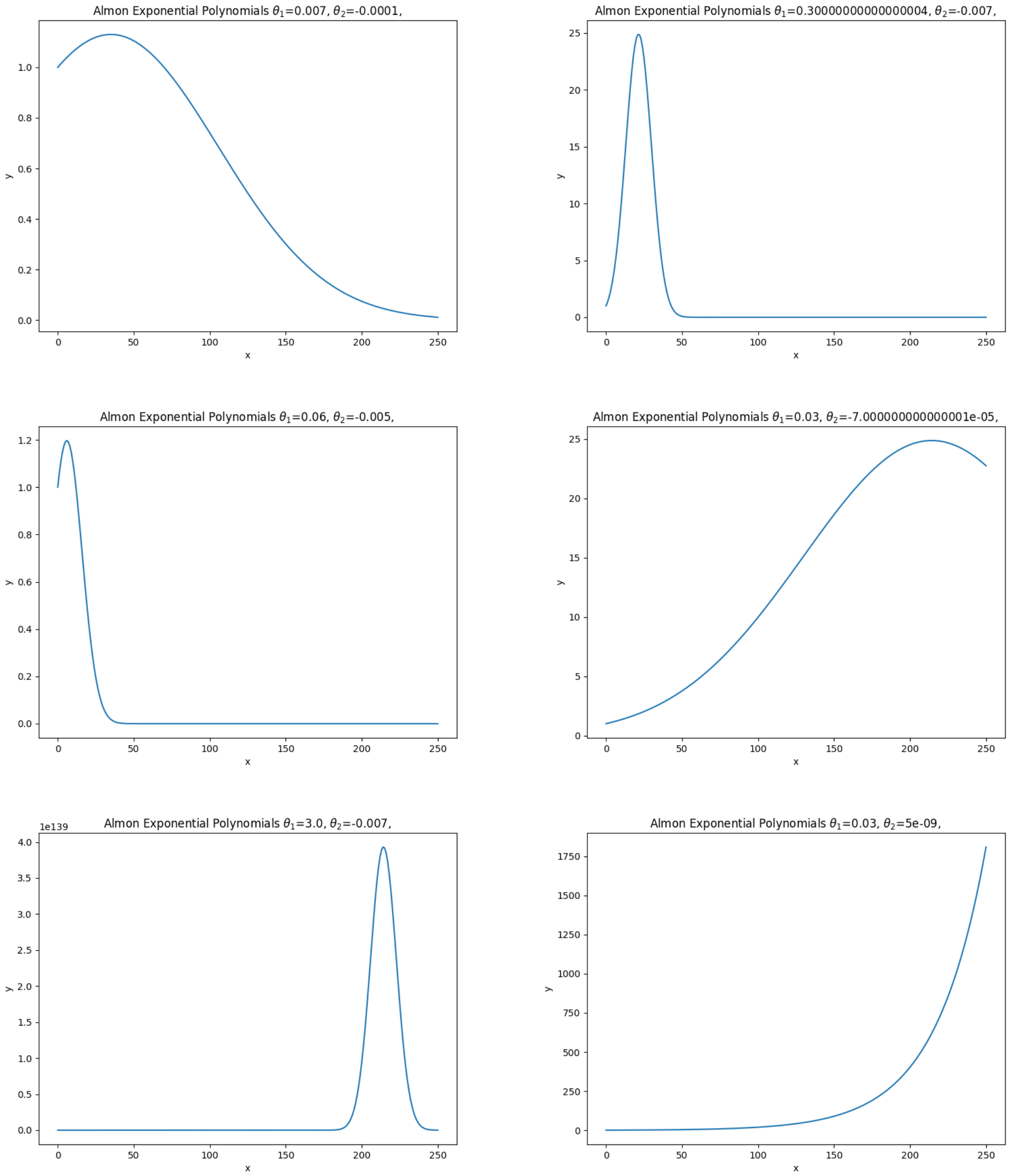


Figure 4.1: Plot of exponential almon lag functions for different values of θ_1 and θ_2

First, it is easy to see that for $\theta_1 = \theta_2 = 0$, we have equal weights (this case is not plotted). Then we can produce a wide variety of decreasing of increasing functions with peaks that can drastically change position according to the considered lag. This behavior then determines how many lags are included in regression. We observe that since the parameters are estimated from the data, once the functional form of $B(k;\theta)$ is specified, the lag length selection is purely data driven.

4.2 Almon optimization

We introduced exponential Almon polynomials to exploit the MIDAS framework without falling into an overparametrized regime. However the relationship that bonds the quantile loss together with the parameters of the model is now more complicated than the one with the linear coefficients and optimization can no longer be performed with linear programming. For this reason we resort to iterative optimization algorithms, that, starting from a initial point, update their estimates of the optimum point until a certain condition is satisfied. We considered two algorithms: a gradient base one, Adam [8] and a euristic one, Nelder-Mead [9]. Below we introduce the main features of these algorithms.

4.2.1 Adam

Stochastic gradient-based optimization is of core practical importance in many fields of science and engineering. If the function is differentiable w.r.t. its parameters, gradient descent is a relatively efficient optimization method, since the computation of first-order partial derivatives w.r.t. all the parameters is of the same computational complexity as just evaluating the function.

Often, objective are composed of a sum of subfunctions evaluated at different subsamples of data; in this case optimization can be made more efficient by taking gradient steps w.r.t. individual subfunctions, i.e. Stochastic Gradient Descent (SGD).

A possible extension of SGD is given by Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments, a method for efficient stochastic optimization that only requires first-order gradients with little memory requirements. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

The name Adam is derived from adaptive moment estimation, and is well suited for problems that are large in terms of data and/or parameters. We now give a description of the algorithm.

Let $f(\theta)$ be a stochastic objective function that is differentiable w.r.t. parameters θ. We are interested in minimizing the expected value of this function, $\mathbb{E}[f(\theta)]$ w.r.t. its parameters θ. With $f_1(\theta), \ldots, f_T(\theta)$ we denote the realisations of the stochastic function at subsequent timesteps $1, \ldots, T$. The stochasticity might come from the evaluation at random subsamples (minibatches) of datapoints. With $g_t = \nabla_\theta f_t(\theta)$ we denote the gradient, i.e. the vector of partial derivatives of f_t, w.r.t θ evaluated at timestep t. Adam algorithm is then defined as follows:

Algorithm 3 Adam

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ ▷ Initialize 1st moment vector
$v_0 \leftarrow 0$ ▷ Initialize 2nd moment vector
$t \leftarrow 0$ ▷ Initialize timestep
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Get stochastic gradients
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ ▷ Update biased first moment estimate
  $v_t \leftarrow \beta_2 \cdot y_{t-1} + (1 - \beta_2) \cdot g_t^2$ ▷ Update biased second raw moment estimate
  $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ ▷ Compute bias-corrected first moment estimate
  $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ ▷ Compute bias-corrected first moment estimate
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{v_t}+\epsilon}$ ▷ Update parameters
end while
return θ_t (resulting parameters)

The algorithm updates exponential moving averages of the gradient (m_t) and the squared gradient (y_t) where the hyper-parameters $\beta_1, \beta_2 \in [0, 1)$ control the exponential decay rates of these moving averages.

A possible interpretation of the ratio $\widehat{m}_t/\sqrt{\widehat{v}_t}$, is that it plays the role of assessing how much we trust that the direction $\widehat{m}_t$ corresponds to the direction of the true gradient: a greater ratio corresponds to a greater uncertainty about whether the direction of m_t corresponds to the direction of the true gradient. We observe that this property is not affected by scaling of the objective function, since the effective stepsize is invariant to the scale of the gradients; rescaling the gradients g with factor c will scale $\widehat{m}_t$ with a factor c and $\widehat{v}_t$ with a factor c^2, which cancel out: $(c \cdot \widehat{m}_t) / \left(\sqrt{c^2 \cdot \widehat{v}_t}\right) = \widehat{m}_t/\sqrt{\widehat{v}_t}$.

The moving averages themselves are estimates of the 1$^{\text{st}}$ moment (the mean) and the 2$^{\text{nd}}$ moment (of the gradient). However, these moving averages are initialized as (vectors of) 0 's, leading to moment estimates that are biased towards zero, especially during the initial timesteps, and especially when the decay rates are small (i.e. the β s are close to 1).

This initialization bias can be easily counteracted, resulting in bias-corrected estimates $\widehat{m}_t$ and $\widehat{v}_t$. Let g be the gradient of the stochastic objective f, then the algorithm computes the second moment estimate as:

$$y_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \tag{4.2.1}$$

Taking expectations of the left-hand and right-hand sides :

$$\begin{aligned}\mathbb{E}\left[y_t\right] &= \mathbb{E}\left[(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\cdot g_i^2\right] \\ &= \mathbb{E}\left[g_t^2\right]\cdot(1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}+\zeta \\ &= \mathbb{E}\left[g_t^2\right]\cdot\left(1-\beta_2^t\right)+\zeta\end{aligned}$$

where $\zeta = 0$ if the true second moment $\mathbb{E}\left[g_i^2\right]$ is stationary; otherwise ζ can be kept small since the exponential decay rate β_1 can (and should) be chosen such that the exponential moving average assigns small weights to gradients too far in the past. What is left is the term $\left(1-\beta_2^t\right)$ which is caused by initializing the running average with zeros. In algorithm 1 we therefore divide by this term to correct the initialization bias. The derivation for the first moment estimate is completely analogous.

In order to apply the algorithm to our problem of maximising the ρ_τ loss with linear linear coefficients parameterised by Almon exponential polynomials, we used the Adam implementation already present in the torch.optim library [10] . In order to be able to exploit the function, we defined the optimization problem in a sequential fashion using the torch modules and torch.tensors, as if we were programming a neural network.

Example of code

[ ]:
```
class InterceptModule(nn.Module):
    ''' auxiliary nn.module for computing a trainable intercept given␣
↪some data

    Attributes:
        self.weights: value of the intercept

    Methods:
        forward(self, x):
            returns the value of the intercept as the prediction of a␣
↪linear model'''

    def __init__(self):
        super(InterceptModule, self).__init__()
        self.weights = nn.Parameter(torch.zeros(1))

    def forward(self, x):
        return self.weights
```

Example of code

[ ]:
```

class MatrixVectorMultiplyLayer(nn.Module):
    '''Auxiliary module for almon specificied linear model: multiplies a␣
↪matrix with vectors of trainable parameters
    Attributes:
```

```
        self.input_dim (int) : dimension of the vector of trainable␣
↪parameters
        self.weights (tensor.torch) : vector of trainable parameters
    '''
    def __init__(self, input_dim):
        super(MatrixVectorMultiplyLayer, self).__init__()
        self.input_dim=input_dim
        self.weights = nn.Parameter(torch.zeros(1,self.input_dim,␣
↪dtype=torch.double) )

    '''Performs the forward pass computation on the input.
    Args:
            x(torch.tensor) : the matrix that multiplies the weights

        Returns:
            torch.tensor: return the product of the x with the vector of␣
↪trainable parameters
        """'''
    def forward(self, x):
        output=torch.matmul( self.weights, x)


        return output
```

Example of code

```
[ ]: class almon_coeff_SGD(nn.Module): #compute almon coeff and multiply them␣
↪with the data
    '''Auxiliary module for almon specificied linear model: multiplies␣
↪the proportion given by the exponential almon polynomials to the␣
↪corresponding data and sums the output
    Attributes:
        n (int) : number of coefficients to compute
        power(int): degree of the almon polynomial
        power_matrix(tensor.torch): auxiliary matrix with entries the␣
↪powers until power of the integers until n
        theta(torch.tensor): parameters of the almon polynomials'''

    def __init__(self, n, power, pow_mat):
        '''Initializes the custom layer.

        Args:
            n (int) : number of coefficients to compute
            power(int): degree of the almon polynomial
            power_matrix(tensor.torch): auxiliary matrix with entries the␣
↪powers of the integers
        '''
        super(almon_coeff_SGD, self).__init__()
```

```
        self.n = n
        self.power = power
        self.power_matrix=pow_mat[0:power,0:n ]
        self.theta =MatrixVectorMultiplyLayer(power)


    def forward(self, x):
      '''Performs the forward pass computation on the input.
        Args:
            x(torch.tensor) : data points

        Returns:
            torch.tensor: returns the sum of the proportion given by the␣
↪exponential almon polynomials multiplied the corresponding data
        '''
      alm_coeff=nn.functional.softmax(self.theta( self.power_matrix), ␣
↪dim=1)
      output=x*alm_coeff
      output=output.sum(dim=1,keepdim=True)

      return output
```

Example of code

```
[ ]: class almon_reg(nn.Module):
         '''Module that computes the prediction for a linear model that allows␣
↪almon specificied coefficients for some variables:
         Attributes:
             lags (list of int) : number of consecutive data that represent␣
↪lags of the inputs variables
             power(list of int): degree of the almon polynomial for each input␣
↪variable
             pow_mat(tensor.torch): auxiliary matrix with entries the powers␣
↪of the integers
             alm_bool (list of bool) : for each lagged feature True if the␣
↪variable is trained with exp almon coefficients (if False they will be␣
↪linear)
             layers(nn.ModuleList): list of modules containing all the␣
↪parameters of the model'''


    def __init__(self, lags, alm_bool, power, pow_mat):
        '''Initializes the custom layer.

        Args:
            lags (list of int) : number of consecutive data that␣
↪represent lags of the inputs variables
```

```
            alm_bool (list of bool) : for each lagged feature True if the␣
↪variable is trained with exp almon coefficients (if False they will be␣
↪linear)
            power(list of int): degree of the almon polynomial for each␣
↪variable
            pow_mat(tensor.torch): auxiliary matrix with entries the␣
↪powers of the integers

        '''
        super(almon_reg, self).__init__()
        self.lags = lags
        self.pow_mat=pow_mat
        self.alm_bool=alm_bool
        self.power=power


        # Create a list of layers
        self.layers = nn.ModuleList()
        j=0
        for (d,b,k) in zip(lags, alm_bool, power):
          if b==True:
            self.layers.append(almon_coeff_SGD(d,k, self.pow_mat))
            j=j+1
          else: self.layers.append(nn.Linear(d,1, bias=False).double())

        n=sum(self.alm_bool) #if there are no almon specified linear␣
↪coefficient use the intercept module to ad an intercept to the model
        if n > 0:
            self.final_linear = nn.Linear(n, 1).double()
        else:  self.final_linear =InterceptModule()




    def forward(self, x):
        '''Performs the forward pass computation on the input.
    Args:
            x(torch.tensor) : data points

        Returns:
            torch.tensor: return the prediction of the model given the␣
↪features x
        '''
        # Split the input into chunks
        input_chunks = np.split(x, np.cumsum(self.lags), axis=1)


        alm_output_chunks = []
        lin_output_chunks =[]
```

```
        # Process each chunk through a separate linear layer
        for chunk, linear_layer, b in zip(input_chunks, self.layers, self.
↪alm_bool):
            if b: alm_output_chunks.append(linear_layer(chunk))
            else: lin_output_chunks.append(linear_layer(chunk))

        # Concatenate the outputs from all linear and alm layers

        if alm_output_chunks:
            output_alm = torch.cat(alm_output_chunks, dim=1)
        else:  output_alm = torch.tensor([0])

        if len(lin_output_chunks)>0:
             output_lin= torch.stack(lin_output_chunks, dim=0)
        else:
            output_lin=torch.zeros(2, requires_grad=False)


        output=self.final_linear(output_alm)+torch.sum(output_lin, dim=0)

        return output
```

Example of code

[ ]:
```
def compute_betas(self, y, X, weights):
        """
    Compute quantile regression coefficients (betas) for each regime␣
↪according to self.type_of_reg
    and saves the respective quantile regression model for each regime

    Parameters:
    - y (np.array): The target variable for regression.
    - X (np.array): The matrix of features.
    - weights (array-like): Posterior probabilities for each regime.

    Returns:
    - betas (np.array): Regression coefficients for each regime.
        """
        # Initialize an array to store regression coefficients for each␣
↪regime and quantile
        betas = np.zeros([self.n_components, self.n_features, len(self.
↪quantile)])

        # Iterate through each regime
        for j in range(self.n_components):
            for q in range(len(self.quantile)):
```

```
                ⋮

            elif(self.type_of_reg=='almon_SGD'):
                # Use stochastic gradient descent for Almon quantile␣
↪regression
                model=self.almon_QR_SGD( y=y,X= X, posterior=weights[:
↪,j], lr=self.lr)
                self.tensor_init[j]=model.state_dict()

                #convert our values into linear parameters for the␣
↪data
                quant_reg_result=self.almon_to_linear_SGD(model)
                betas[j,:,q]=quant_reg_result
                self.qrmodel[j]= model


        return betas
```

Example of code

```
[ ]:    def almon_QR_SGD(self, y, X, posterior, init_tensor=None, lr=0.01):
            ''' Fit a quantile linear regression model allowing exp almon␣
↪parametrisation of coefficients, optimised with Adam optimiser.
           The model allows for L1 regularization.
        Args:
            y(np.array) : target variables
            X(np.array) = feature matrix
            init_tensor (torch.Tensor): optional initial tensor for model␣
↪weights.
            weights(np.array):  posterior probabilities that act as weights␣
↪for each element of y in the quantile loss
            lr (float): Learning rate for the Adam optimizer (default is 0.
↪01).



        returns:
            (almon_reg) : fitted model
                          '''
            #quantile regression performed through almon exp coefficients
            #relies on the assumption that a feature can either affect␣
↪positevely or negatively the return variable for all lags
            torch.manual_seed(self.rnd)
            # Create an instance of the model and define a quantile␣
↪regression loss function
            model = almon_reg(lags=self.lags, alm_bool=self.alm_bool,␣
↪power=self.alm_power, pow_mat=self.pow_mat)
            loss_fn =  rho_loss(tau=self.quantile[0])
```

```
        # Define a Lasso regularization loss
        lasso_loss = CustomLassoLoss(alpha=self.alpha,␣
↪target_layer='final_linear.weight')

        # Initialize the Adam optimizer
        optimizer =  optim.Adam(model.parameters(), lr=lr)


         # Convert data to PyTorch tensors
        input_data = torch.tensor(X)
        target_data =  torch.tensor(y)
        weights_arr=torch.tensor(posterior)

        # Load the initial tensor if provided and not restarting
        if (init_tensor is not None) and not self.restart:
            model.load_state_dict(init_tensor)

        # Create a custom dataset and data loader for data and weights
        dataset = MyDataset(input_data,torch.unsqueeze( target_data,1),␣
↪torch.unsqueeze(weights_arr,1))
        batch_size=input_data.size()[0]//self.nbatch
        dataloader = DataLoader(dataset, batch_size=batch_size,␣
↪shuffle=True)

        # Training loop
        model.train()
        for epoch in range(self.epochs):
            for inputs, targets, weights in dataloader:
          # Zero the gradients
                  optimizer.zero_grad()

                  # Forward pass
                  predictions = model(inputs)

                  # Compute the quantile regression loss + alpha *␣
↪Lasso loss weighted by the sum of the weights
                  loss = loss_fn(predictions, targets,weights)  +torch.
↪sum(weights)*lasso_loss(model)

                  # Backpropagation
                  loss.backward()

                  # Update the model's parameters using Adam optimizer
                  optimizer.step()

        return model
```

4.2.2 Nelder-Mead

The Nelder-Mead simplex algorithm is a widely used direct search method for solving the unconstrained optimization problem

$$\min f(x) \tag{4.2.2}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function and n the dimension. A simplex is a geometric figure in n dimensions forming the the convex hull of $n+1$ vertices. We denote a simplex with vertices $\mathbf{x}_1, \mathbf{x}_1, \ldots, \mathbf{x}_{n+1}$ by Δ.

The Nelder-Mead method iteratively generates a sequence of simplices to approximate an optimal point of $f(x)$. At each iteration, the vertices $\{\mathbf{x}_j\}_{j=1}^{n+1}$ of the simplex are ordered according to the objective function values

$$f(\mathbf{x}_1) \leq f(\mathbf{x}_2) \leq \cdots \leq f(\mathbf{x}_{n+1})$$

We refer to $\mathbf{x}_1$ as the best vertex, and to $\mathbf{x}_{n+1}$ as the worst vertex. If several vertices have the same objective values, consistent tie-breaking rules are required for the method to be well-defined.

The algorithm employs four primary operations: reflection, expansion, contraction, and shrinkage, each linked to a scalar parameter: α (reflection), β (expansion), γ (contraction), and δ (shrink). The values of these parameters satisfy $\alpha > 0$, $\beta > 1, 0 < \gamma < 1$, and $0 < \delta < 1$. In a common implementation of the Nelder-Mead method the parameters are chosen to be

$$\{\alpha, \beta, \gamma, \delta\} = \{1, 2, 1/2, 1/2\}$$

Let $\overline{\mathbf{x}}$ be the centroid of the n vertices with smallest f. Then

$$\overline{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$$

We now outline the Nelder-Mead method:

Algorithm 4 Nelder mead

- **1** Sort. Evaluate f at the $n+1$ vertices of Δ and sort the vertices so that (1.2) holds.

- **2** Reflection. Compute the reflection point $\mathbf{x}_r$ from

$$\mathbf{x}_r = \overline{\mathbf{x}} + \alpha\left(\overline{\mathbf{x}} - \mathbf{x}_{n+1}\right)$$

  Evaluate $f_r = f\left(\mathbf{x}_r\right)$. If $f_1 \leq f_r < f_n$, replace $\mathbf{x}_{n+1}$ with $\mathbf{x}_r$.

- **3** Expansion. If $f_r < f_1$ then compute the expansion point $\mathbf{x}_e$ from

$$\mathbf{x}_e = \overline{\mathbf{x}} + \beta\left(\mathbf{x}_r - \overline{\mathbf{x}}\right)$$

  and evaluate $f_e = f\left(\mathbf{x}_e\right)$. If $f_e < f_r$, replace $\mathbf{x}_{n+1}$ with $\mathbf{x}_e$; otherwise replace $\mathbf{x}_{n+1}$ with $\mathbf{x}_r$.

- **4** Outside Contraction. If $f_n \leq f_r < f_{n+1}$, compute the outside contraction point

$$\mathbf{x}_{oc} = \overline{\mathbf{x}} + \gamma\left(\mathbf{x}_r - \overline{\mathbf{x}}\right)$$

  and evaluate $f_{oc} = f\left(\mathbf{x}_{oc}\right)$. If $f_{oc} \leq f_r$, replace $\mathbf{x}_{n+1}$ with $\mathbf{x}_{oc}$; otherwise go to step 6.

- **5** Inside Contraction. If $f_r \geq f_{n+1}$, compute the inside contraction point x_{ic} from

$$\mathbf{x}_{ic} = \overline{\mathbf{x}} - \gamma\left(\mathbf{x}_r - \overline{\mathbf{x}}\right)$$

  and evaluate $f_{ic} = f\left(\mathbf{x}_{ic}\right)$. If $f_{ic} < f_{n+1}$, replace $\mathbf{x}_{n+1}$ with $\mathbf{x}_{ic}$; otherwise, go to step 6.

- **6** Shrink. For $2 \leq i \leq n+1$, define

$$\mathbf{x}_i = \mathbf{x}_1 + \delta\left(\mathbf{x}_i - \mathbf{x}_1\right)$$

While the Nelder-Mead method might not always reach a critical point of F, it consistently demonstrates strong performance and remains widely favored as one of the main direct search methods.

In our code Nelder-Mead algorithm was already implemented in the library scipy.optimize.minimize, and since it deals with numpy functions we had to build the computation of the prediction through Almon polynomials, now working with numpy.arrays.

Example of code

```
[ ]:
def almon_coeff(theta,data, n, power, pow_mat):
        """
    Calculate single coefficients using exp Almon polynomial␣
 ↪parametrization multiplied with the data and summed.
```

```
    Args:
        theta (np.array): Parameters of the Almon polynomial.
        data (np.array): Feature matrix.
        n (int): Number of consecutive columns representing lags in the␣
↪input variables.
        power (int): degree of the almon plynonmial.
        pow_mat (np.array): Auxiliary matrix with powers of integers␣
↪needed for the calculation of Almon polynomials.

    Returns:
        np.array: Almon coefficients for each lag of the variable␣
↪multiplied by the data and summed.
    """
        power_matrix=pow_mat[0:n,0:power ]
        alm_pol = np.matmul(power_matrix, theta)
        alm_exp = softmax(alm_pol)
        return np.matmul( data, alm_exp)
```

Example of code

```
[ ]: def almon_QR(coeff, data, y, lags, alm_bool, power, pow_mat, tau,␣
↪weights, alpha=0):
        ''' given the parameters,  returns the weighted quantile loss with␣
↪lasso of the linear regression with exp almon parametrisation of␣
↪coefficients
        Args:
            coeff (np.array): parameters of the predictor made of almon␣
↪polynomials and pure lienar coefficients
            data(np.array) = feature matrix
            y(np.array) : target variables
            lags (list of int) : number of consecutive columns that␣
↪represent lags of the inputs variables present in data
            alm_bool (list of bool) : for each lagged feature True if the␣
↪variable is trained with exp almon coefficients (if False they will be␣
↪linear)
            power (lis of int) : number of parameters for each varaiable␣
↪that is trained with almon (needs a placeholder for the purely linear␣
↪ones)
            pow_mat(np.array): auxiliary matrix with powers of integers␣
↪needed for calculation of almon polynomials
            tau (float): quantile parameter for the QR
            weights(np.array): an array of weights for each element of y in␣
↪the quantile loss
            alpha(float): parameter of the lasso regularization

        returns:
            float: quantile loss of the calculated regression
```

```
        '''
    theta_len=sum(power)
    #divide the model parameters and data in chunks according to the␣
↪data variables and powers
    theta_chunks = np.split(coeff[:theta_len], np.cumsum(power)[:-1],␣
↪axis=0)
    data_chunks = np.split(data, np.cumsum(lags)[:-1], axis=1)
    output_chunks = []
    lasso_lin=[]

    #compute the outputs of each variable according to their␣
↪parametrisation
    for (d,b,k,t_chunk, d_chunk) in zip(lags, alm_bool, power,␣
↪theta_chunks, data_chunks):
        if b==True:

          output_chunks.append(almon_coeff(t_chunk, d_chunk, d, k,␣
↪pow_mat))

        else:
            output_chunks.append(np.matmul(d_chunk, t_chunk))
            lasso_lin.append(abs(t_chunk).sum())


    #multiply by a linear coefficient to the almon speciefied variables
    j=0
    for c in coeff[theta_len:-1]:
        if alm_bool[j]==True:
          output_chunks[j]=c*output_chunks[j]
        j=j+1
    output=np.column_stack(output_chunks)

    #calculate the regularised loss
    loss= rho(y-(np.sum(output, axis=1)+coeff[-1]), tau)
    return np.dot(loss,weights)+weights.
↪sum()*alpha*(sum(coeff[theta_len:-1])+sum(lasso_lin))
```

Example of code

```
[ ]: def compute_betas(self, y, X, weights):
        """
    Compute quantile regression coefficients (betas) for each regime␣
↪according to self.type_of_reg
    and saves the respective quantile regression model for each regime

    Parameters:
    - y (np.array): The target variable for regression.
    - X (np.array): The matrix of features.
    - weights (array-like): Posterior probabilities for each regime.
```

```
    Returns:
    - betas (np.array): Regression coefficients for each regime.
        """
        # Initialize an array to store regression coefficients for each␣
↪regime and quantile
        betas = np.zeros([self.n_components, self.n_features, len(self.
↪quantile)])

        # Iterate through each regime
        for j in range(self.n_components):
            for q in range(len(self.quantile)):

                ⋮

                elif(self.type_of_reg=='almon'):
                    # Use Nelder-Mead optimization for Almon quantile␣
↪regression
                    result=minimize(almon_QR, self.initial_guess[j],␣
↪args=( X, y, self.lags, self.alm_bool, self.alm_power,
                    self.pow_mat, self.quantile[q], weights[:,j], self.
↪alpha), method='Nelder-Mead', tol=1e-16)
                    if not self.restart:
                        self.initial_guess[j]=result.x

                    #convert our values into linear parameters for the␣
↪data
                    quant_reg_result=almon_to_linear(result.x,␣
↪alm_bool=self.alm_bool, lags=self.lags, power=self.alm_power,␣
↪pow_mat=self.pow_mat)
                    betas[j,:,q]=quant_reg_result
                    self.qrmodel[j]= result

                    ⋮

        return betas
```

4.3 Handling multifrequency data

In order to fit a multifrequency quantile regression model, a dataset is needed. However data are often given in a timeseries format, where each datapoint for each variable is placed in a row that represent the time when that specific variable was sampled. For this reason a tool to convert this dataset to a dataset useful for regression tasks is needed. The function that was developed is quite straightforward: given a time series dataframe, an output variable and a list of features, possibly sampled at different timesteps, the number of their past lags to be put in a row and the time difference between the output variable and the most recent lag of the covariates, return a matrix that contains a column with the outputs and other columns with all the covariates and their respective lags. The code is quite simple too, however it is able to handle many different data frequencies such as daily weekly, monthly and quarterly.

The followings are auxiliary variables, necessary to adjust the date that we are working with, since weekly or monthly shifts in data may occur in days such as weekend or holiday, or shifting the date by one month may not end up in the last day of the month.

Example of code

[ ]:
```
def is_weekend(date):
    ''' Check if the day of the week is Saturday (5) or Sunday (6)
    Args:
        date (datetime)
    Returns:
        bool: True if the day is either a Saturday or a Sunday, False␣
↪otherwise'''
    day_of_week = date.weekday()

    return day_of_week >=5
```

Example of code

[ ]:
```
def begin_date(date, n):
    ''' given a date go forward exactly n weekdays and return the␣
↪correspoonding date in a weekday
    Args:
        date (datetime) : starting date
        n (int) : minimum number of days to advance
    returns:
      datetime: The resulting first weekday date after moving forward at␣
↪least n days

    '''
    weeks=n//5
    new_date=date+relativedelta(days=weeks*7)
    new_date=new_date+relativedelta(days=n%5)
    if (is_weekend(new_date)):
        new_date=new_date+relativedelta(days=2) #either saturday or␣
↪sunday means that the day that was meant was two days later
    return new_date
```

Example of code

[ ]:
```
def end_date(date, n): #given a date go backward exactly n weekdays and␣
↪return the corresponding date in a weekday
    ''' given a date go backward exactly n weekdays and return the␣
↪correspoonding date in a weekday
    Args:
        date (datetime) : starting date
        n (int) : minimum number of days to go backward in time
    returns:
```

```
        datetime: The resulting first weekday date after moving backward at␣
↪least n days

    '''
    weeks=n//5
    new_date=date-relativedelta(days=weeks*7)
    new_date=new_date-relativedelta(days=n%5)
    if (is_weekend(new_date)):
        new_date=new_date-relativedelta(days=2) #either saturday or␣
↪sunday means that the day that was meant was two days before
    return new_date
```

Example of code

```
[ ]: def is_friday(date):
         ''' Check if the day of the week is Friday (4)
         Args:
             date (datetime)
         returns:
             bool True if the date is a friday False otherwise '''

         day_of_week = date.weekday()
         return day_of_week ==4
```

Example of code

```
[ ]: def date_shift_plus(date, shift):

         ''' Add the date with a relativedelta (shift) and adjustes the date␣
     ↪according to the shift:
         when it is a weekend if shifted by one day or
         when it is not friday if the shift is 7 days or
         when it is not the last day of the month if it is shifted by one or␣
     ↪more months


         Args:
             date (datetime): starting date
             shift (relativedelta) : time period to shift date into the future
         returns:
             datetime: the shifted date adjusted acoording to the kind of␣
     ↪shift'''

         new_date=date+shift
         if(new_date+shift<new_date+relativedelta(days=7)):
             while(is_weekend(new_date)): #avoid weekends
                 new_date=new_date+relativedelta(days=1)

         if new_date+shift==new_date+relativedelta(days=7):
             while not is_friday(new_date)   :
```

```
            #if the Y are sampled weekly start regression at the first␣
↪following friday
            new_date= new_date+relativedelta(days=1)

    if(new_date+shift>=new_date+relativedelta(months=1)):
        #added because it may happen that when getting one month into the␣
↪future it may not to be the last day of the month
        new_date=next_last(new_date)

    return new_date
```

Example of code

```
[ ]: def date_shift_minus(date, shift):
    ''' subtract the date with a relativedelta (shift) and adjustes the␣
↪date according to the shift:
    when it is a weekend if shifted by one day or
    when it is not friday if the shift is 7 days or
    when it is not the last day of the month if it is shifted by one or␣
↪more months



    Args:
        date (datetime): starting date
        shift (relativedelta) : time period to shift date into the past
    returns:
        datetime: the shifted date adjusted acoording to the kind of␣
↪shift'''

    new_date=date-shift
    if(new_date+shift<new_date+relativedelta(days=7)):
        while(is_weekend(new_date)): #avoid weekends
            new_date=new_date-relativedelta(days=1)

    if new_date+shift==new_date+relativedelta(days=7):
        while not is_friday(new_date)  :
            #if the Y are sampled weekly start regression at the first␣
↪following friday
            new_date= new_date-relativedelta(days=1)

    if(new_date+shift>=new_date+relativedelta(months=1)):
        #added because it may happen that when getting one month into the␣
↪past, it may not to be the last day of the month
        new_date=next_last(new_date)

    return new_date
```

Example of code

```
[ ]: def next_last(date):
        ''' given a date find the first next date which is at the end of the␣
     ↪month,
            if the date is already the last keeps the original one
        Args:
            date (datetime): starting date
        returns:
            datetime: the last date of the month of date'''
        new_date=date+relativedelta(months=1)
        year = new_date.year
        month = new_date.month
        new_date = datetime.datetime(year, month, 1)
        return new_date-relativedelta(days=1)
```

Then the following is the main function that allows us to build a dataset for regression. The function first computes the first date for the return variable, such that we have enough past data on the other variables to perform regression according to the lags that we considered. Then for each feature, their oldest lag that is going to be inserted in the dataset is retrieved. Then the function proceeds to go forward in time collecting all the needed lags of such variables. This procedure is repeated for each feature and after running all the features, the function simply shifts the return variable by one lag into the future, and the features are again computed starting from that date. At the end of the code we also perform a simple quantile regression on the newly built dataset.

Example of code

```
[ ]: def easy_quant_midas(data, Y_label, Y_freq, X_labels, X_freq, past_m,␣
     ↪quantile=0.5, aplha=0, time_offset=1):
        ''' convert a multivariate time series dataset, into a dataset to be␣
     ↪used for regression, and performs quantile regression on the dataset.
        Args:
        data(pd.dataframe): timeseries dataframe
        Y_label(string) : name of the output variable
        Y_freq (relativedelta): frequency at which the output data is sampled␣
     ↪(either 7 days or 1 month)
        X_label(list of string) :the names of the imput variables
        X_freq (list of relativedelta): frequency at which each of the imput␣
     ↪data is sampled (euqally or more frequently than Y, the output␣
     ↪variable)
        past_m (list of int): lags of each of the inputs to be inserted in␣
     ↪each row the dataset
        quantile (float): quantile parameter for the quantile regression
        alpha (float): coefficient of the L1 regularization of the quantile␣
     ↪regression
        time_offset(int): time_offset*Y_freq is the time difference between␣
     ↪the Y we are estimating and the most recent lag of the features X

        returns:
```

```
        quant_reg_result(class QuantileRegressor)= fitted lieanr quantile␣
↪regressor model
        X(np.array)= matrix of the features
        Y(np.array) = array of the output data
        index(list of datetime)= dates corresponding to the Y in the␣
↪return dataset'''


    X = []
    Y = []
    index=[]
    shift_time=time_offset*Y_freq

     # Find the first Y date in which there are enough past variables to␣
↪do the regression for every X_label
    start_time=[]
    for (time, m) in zip(X_freq, past_m):
        if (time==relativedelta(days=1)):
            start_time.append(begin_date(data.index[0]+shift_time, m))


        else:
            start_time.append(data.index[0]+shift_time+m*time)

    run_date=max(start_time)

     # Start regression at the first following non-weekend day if Y is␣
↪sampled daily
    while((run_date+Y_freq<run_date+relativedelta(days=7)) and␣
↪(is_weekend(run_date))): #the comparison is computed in this way␣
↪beacuse there is no < or > for relativedata class objects
         run_date= run_date+relativedelta(days=1)

    # Start regression at the first following Friday if Y is sampled␣
↪weekly
    while((run_date+Y_freq==run_date+relativedelta(days=7))  and (not␣
↪is_friday(run_date))):
         run_date= run_date+relativedelta(days=1)

     # Find the first feasible month if Y is sampled quarterly
    if (run_date+Y_freq==run_date+relativedelta(months=3)):
            while((run_date.month%3) != 1):
                run_date= run_date+relativedelta(months=1)

     # Start regression at the first following 1 of the month if Y is␣
↪sampled more than monthly
    if (run_date+Y_freq>=run_date+relativedelta(months=1)):
        run_date=next_last(run_date)
```

```
    end_time=data.index[-1]

    while(run_date <= end_time):

            if pd.isnull(data.loc[run_date][Y_label]):
                raise RuntimeError(Y_label+" , the response variable is␣
↪null in date " + str(run_date))

            row = np.array([])
            feat_date= run_date-shift_time
            for regr in zip(X_labels, X_freq, past_m): #! the dates are␣
↪ordered from the farthest to the closest to the Y date for every label␣
↪

                        # Collect data in the past from the time offset␣
↪selected

                        label=regr[0]
                        freq=regr[1]
                        m=regr[2]
                        aux_date=date_shift_minus(feat_date+freq, freq)␣
↪#trick to go back to the first feasable date

                        for j in range(m):

                            new_aux_date=aux_date

                            while pd.isnull(data.loc[new_aux_date]␣
↪[label] ):

                                    # Replace any data that is null by␣
↪looking further in the past
                                    ␣
↪new_aux_date=date_shift_minus(new_aux_date,freq)

                            aux_date=date_shift_minus(aux_date, freq)

                            row= np.append(  row, data.loc [new_aux_date]␣
↪[label])
```

```
            if not np.any(np.isnan(row)):
                X.append(row)
                Y.append(data.loc[run_date][Y_label])
                index.append(run_date)

            run_date=date_shift_plus(run_date,Y_freq)



    X = np.array(X)
    Y = np.array(Y)

    # Create a linear quantile regressor model
    quant_reg_result=[]
    qr = QuantileRegressor(quantile=quantile, alpha=0,solver='highs')
    quant_reg_result = qr.fit(X, Y)
    y_pred = quant_reg_result.predict(X)


    return [quant_reg_result, X,Y, index]
```

Chapter 5

Testing quantile models

Assessing the validity of quantile regression estimates given by a model is a non trivial problem as quantiles are not observable. Therefore the analysis has to rely upon the study of the behaviour of the violations in order to test its validity, that is the study of the instances where the observed value exceeds the predicted quantile. A model is hence valid if the violation process satisfies some theoretical hypothesis. On this basis we present three tests that aim to address this problem. When performing estimation of quantiles, there are three universal points that arise :

- The power of backtesting tests, crucial for identifying model validity, tends to be low, especially in small samples.
- Backtesting methodologies should be model-free to ensure applicability across different models.
- Estimation risk must be accounted for, as the risk of estimation error present in the estimates of the parameters pollutes quantile forecasts.

While the second condition is satisfied by our tests we will discuss the other issues in the last section of this chapter.

5.1 Framework

Consider the following general statistical model:

$$y_t = f\left(y_{t-1}, \mathbf{x}_{t-1}, \ldots, y_1, \mathbf{x}_1; \beta_0\right) + \epsilon_t\theta \equiv f_t(\beta_0) + \epsilon_{t,\tau}, \quad t = 1, \ldots, T, \tag{5.1.1}$$

where $f_1(\beta)$ is some given initial condition, $\mathbf{x}_t$ is a vector of exogenous or predetermined variables, $\mathcal{F}_t = [y_{t-1}, \mathbf{x}_{t-1}, \ldots, y_1, \mathbf{x}_1, f_1(\beta)]$ is the information set available at time t, and for every $t = 1, \ldots, T$, $\epsilon_{t,\tau}$ is a random variable for which we assume $Q_{\epsilon_{t,\tau}}(\tau|\mathcal{F}_t) = 0$.

Then we have that

$$\Pr\left[y_t < f_t\left(\beta_0\right)|\mathcal{F}_t\right] = \tau \quad \forall t = 1, \ldots, T \tag{5.1.2}$$

This is equivalent to requiring that $\{I\left(y_t < f_t\left(\beta_0\right)\right)\}_{t=1}^{T}$, the violation process satisfies the property:

$$\mathbb{E}[I_t|\mathcal{F}_{t-1}] = \tau \tag{5.1.3}$$

It can be verified that this condition implies that the sequence of indicator functions is a sequence of Bernoulli iid random variabales, with parameter τ . Hence a property that any quantile estimate should satisfy is that of providing a filter to transform a (possibly) serially correlated and heteroskedastic time series into a serially independent sequence of indicator functions.

Indeed let's remark separately that the process $\{I(y_t < f_t(\beta_0))\}_{t=1}^T \equiv \{I_t\}_{t=1}^T$ satisfies the following two hypotheses:

- The Unconditional coverage (UC thereafter) hypothesis: the probability of the observed y_t exceeding the quantile forecast must be equal to:

$$\Pr[I_t = 1] = \mathbb{E}[I_t] = \tau. \tag{5.1.4}$$

- The independence hypothesis: violations observed at two different dates must be distributed independently. In other words, past violations should not be informative about current and future violations.

The UC hypothesis is a straightforward one. Indeed, if the frequency of violations observed over T periods is significantly lower (respectively higher) than the quantile (also called coverage rate) τ then the model overestimates (respectively underestimates) the true quantile. However, the UC hypothesis shades no light on the possible dependence of violations.

Therefore, the independence property of violations is an essential one, because it is related to the ability of a quantile model to accurately model the higher-order dynamics of returns. In fact, a model which does not satisfy the independence property can lead to clustering of violations (for a given period) even if it has the correct average number of violations. Consequently, there must be no dependence in the violations variable.

Thus a first natural way to test the validity of the forecast model with parameter β, is to check whether the sequence $\{I(y_t < f_t(\beta))\}_{t=1}^T \equiv \{I_t\}_{t=1}^T$ is iid.

However while these kind of tests can detect the presence of serial correlation in the sequence of indicator functions $\{I_t\}_{t=1}^T$, this is still only a necessary but not sufficient condition to assess the performance of a quantile model. Indeed, it is not difficult to generate a sequence of independent $\{I_t\}_{t=1}^T$ from a given sequence of $\{y_t\}_{t=1}^T$: it suffices to define a sequence of independent random variables $\{z_t\}_{t=1}^T$, such that

$$z_t = \begin{cases} 1 & \text{with probability } \tau \\ -1 & \text{with probability } (1-\tau) \end{cases} \tag{5.1.5}$$

Then setting $f_t(\beta) = Kz_t$, for K large, will be a sequence of random variables, that can deceive our tests.

Notice, however, that once z_t is observed, the probability of exceeding the quantile is known to be almost 0 or 1 . Thus the unconditional probabilities are correct and serially uncorrelated, but the conditional probabilities given the quantile are not. This example is an extreme case of quantile measurement error. Any noise introduced into the quantile estimate will change the conditional probability of overestimating the present quantile given the estimate itself.

Therefore, just testing for the iid condition has no power against this form of misspecification. Now, with the goal of building a test that can deal with this situation we define:

$$\text{Hit}_t \equiv \text{Hit}_t\left(\beta^0\right) \equiv I\left(y_t < f_t\left(\beta^0\right)\right) - \tau$$

The Hit_t function assumes value $(1-\tau)$ every time y_t is less than the quantile and $-\tau$ otherwise. Clearly, we have:

$$\mathbb{E}\left[\text{Hit}_t\right] = 0.$$

Furthermore, from the definition of the quantile function, the conditional expectation of Hit_t given any information known at $t-1$ must also be 0 .

In particular, Hit_t must be uncorrelated with its own lagged values and with $f_t(\beta)$, and must have expected value equal to 0 . If Hit_t satisfies these moment conditions, then there will be no autocorrelation in the hits, no measurement error as in 5.1.5, and the correct fraction of exceptions.

5.2 Preliminary definitions

Before defining the first two tests let's briefly introduce the Wald test. Further reference can be found for example in [11]

Let's consider a statistical model with parameter $\delta_0 \in \mathbb{R}^d$, and let's call $\hat{\delta}_n \in \mathbb{R}^d$, $n \in \mathbb{N}$ a sequence of estimators satisfying the following central limit theorem:

$$n^{1/2}(\hat{\delta}_n - \delta_0) \to^d N(0, I^{-1}(\delta_0)) \tag{5.2.1}$$

where $I(\delta_0)$ is the Fisher information matrix of δ_0.

We consider testing hypotheses about δ of the form

$$H_0 : \delta_0 = \delta$$

versus

$$H_1 : \delta_0 \neq \delta.$$

We note that if δ_0 is a vector of regression coefficients and $\delta = 0$, this is a test about the significance of corresponding covariates.

Then a commonly used test statistics for testing our hypothesis is the Wald statistic:

$$n(\hat{\delta_n} - \delta)' I(\delta)(\hat{\delta_n} - \delta)$$

which measures the weighted distance between the unrestricted estimate $\hat{\delta}_n$ of δ_0 and its hypothetical value δ under H_0. Alternatively, $I(\delta_0)$ may be replaced by any consistent estimator $V(\hat{\delta}_n)$ of the variance of δ_n. The test rejects the hypothesis at significance level α when

$$n(\hat{\delta}_n - \delta)' I(\delta_0)(\hat{\delta}_n - \delta) > \chi^2_{1-\alpha,d} \tag{5.2.2}$$

where $\chi^2_{1-\alpha,d}$ is the quantile at level α of the chi-squared distribution with d degrees of freedom.

We now define the linear regression model. Given a dataset $(y_t, x_t) \in \mathbb{R}\times\mathbb{R}^d$ for $t = 1, \ldots, n$ where y_i will be the return variables and x_t the features or covariates, we consider the model

$$y_t = \delta_0^T x_t + \epsilon_t \tag{5.2.3}$$

where $\delta_0 \in \mathbb{R}^d$ and $\epsilon_{t_{t=1}^n}$ is a sequence random variables that represent the error terms that are assumed to have all mean zero and limited variance. We define the OLS estimator $\hat{\delta}_n$ as:

$$\hat{\delta}_n = \operatorname*{argmin}_{\delta\in\mathbb{R}^d} \sum_{t=1}^{n} |y_t - \delta^T x_i|^2 \tag{5.2.4}$$

or more compactly:

$$\hat{\delta}_n = (X^\top X)^{-1} X^\top y \tag{5.2.5}$$

where X is the matrix whose t-th column is x_i, and y is the vector whose t-th component is y_t. One can thus express the variance of $\hat{\delta}_n$ as:

$$V(\hat{\delta}_n) = V\left((\mathbf{X}^\top X)^{-1} X^\top y\right) = (X^\top X)^{-1} X^\top \mathbf{\Sigma} X (X^\top X)^{-1} \tag{5.2.6}$$

where where $\mathbf{\Sigma}$ represents the covariance matrix of the error terms.

5.3 Our tests

5.3.1 Unconditional Coverage and joint dynamic quantile test

Given a sequence of $\text{Hit}(\hat{\beta})_{t=0}^n$, for $\hat{\beta}$ an estimator of β_0, the first tests we introduce are based on the following linear regression models:

$$\text{Hit}_t(\hat{\beta}) = \delta^{uc} + \epsilon_t^{uc} \tag{5.3.1}$$

$$\text{Hit}_t(\hat{\beta}) = \delta^{jdq} + \mu^{jdq}\,\text{Hit}_{t-1}(\hat{\beta}) + \epsilon_t^{jdq} \tag{5.3.2}$$

We define $\hat{\delta}_n^{uc}$, and $(\hat{\delta}_n^{jdq}, \hat{\mu}_n^{jdq})$ as the OLS estimator of 5.3.1 and 5.3.2 respectively.

We now call the Unconditional Coverage (UC)test as the Wald test with null hypothesis $\delta^{uc} = 0$ applied in model 5.3.1 to the δ_n^{uc} estimator. The name is self-explanatory, with this test we are trying to verify the unconditional coverage property. That means that we are aiming not to reject the null hypothesis

Similarly, we refer to the Joint Dynamic Quantile (JDQ) test as the Wald test with null hypothesis $(\delta^{jdq}, \mu^{jdq}) = (0, 0)$ applied in model 5.3.2 to the $(\delta_n^{jdq}, \mu_n^{jdq})$ estimator. With this test instead we are aiming to verify the independence of the Hit_t variable with respect to its past lag, by showing lack of correlation. Again we are aiming not to reject the null hypothesis.

In order to being able to compute the statistics we now need only to define our estimator for the covariance matrices of our parameters. A common assumption, especially when dealing with economic variables, is heteroskedasticity of the errors, that is the variance of the error terms may change through time. Another common assumption is to assume the error variables to be autocorrelated. For this reason both $V(\hat{\delta}_n^{uc})$ and $V((\hat{\delta}_n^{jdq}, \hat{\mu}_n^{jdq}))$ were estimated using the heteroskedasticity and autocorrelation robust standard estimator proposed in [12], also known as the Newey–West estimator:

$$X^{\mathrm{T}}\Sigma X = \frac{1}{T}\sum_{t=1}^{T} e_t^2 x_t x_t^{\mathrm{T}} + \frac{1}{T}\sum_{\ell=1}^{L}\sum_{t=\ell+1}^{T} w_\ell e_t e_{t-\ell}(x_t x_{t-\ell}^{\mathrm{T}} + x_{t-\ell}x_t^{\mathrm{T}}) \tag{5.3.3}$$

$$\tag{5.3.4}$$

where $w_\ell = 1 - \frac{\ell}{L+1}$, L is the number of past lags for which we assume autocorrelation, X represent the covariates of the linear regression, and e_t are the residuals, the difference between the return variable and its prediction at time t. In order to perform our tests we assume only one lag of autocorrelation.

Example of code

```
[ ]:
def uc_and_jdq_test(Y,quant_pred_Y, q, alpha=0.05, time_offset=0):
  """
    Conducts unconditional coverage and joint dynamic quantile tests.

    Parameters:
    - Y (np.ndarray): Observations
    - quant_pred_Y (np.ndarray): Prediction of quantile
    - q (float): Quantile
    - alpha (float): Threshold for rejecting the null hypothesis (default␣
↪is 0.05)
    - time_offset (int): if Y is calculated as time increments of another␣
↪quantity, reprents
        the number of Y lags of Y that separates the values used to␣
↪compute the increments

    Returns:
    - p_value_1 (float): P-value of the unconditional coverage test
    - p_value_2 (float or None): P-value of the joint dynamic quantile␣
↪test (or None if an error occurs)
    """

#Compute the Hits
  H=(Y<quant_pred_Y).astype(np.float64)-q

  # Create a constant term and fit an OLS model
  X = sm.add_constant(np.ones_like(H))
  model_1 = sm.OLS(H, X).fit(cov_type='HAC', cov_kwds={'maxlags':1,␣
↪'use_correction':True})

  # Extract the delta parameter and perform hypothesis test
  delta=model_1.params[0]
  print(delta)
```

```
    hypothesis_test = model_1.t_test("const = 0")
    p_value_1 = hypothesis_test.pvalue
    print(f"P-value: {p_value_1} ",  end="")

# Check if the null hypothesis is rejected based on the p-value
    if p_value_1 < alpha:
        print("unconditional coverage test: Reject the null hypothesis(bad␣
↪fit)")
    else:
        print("unconditional coverage test: Fail to reject the null␣
↪hypothesis(possibly good fit)")


    # Create lagged data for the joint dynamic quantile test
    lagged_data = np.roll(H, shift=-time_offset)
    lagged_data= lagged_data[:-time_offset]
    H_reg=H[:-time_offset]
    print(H)
    print(time_offset)
    if not np.all(H_reg==H_reg[0]):

         # Fit an OLS model for the joint dynamic quantile test
        model_2 = sm.OLS(lagged_data, sm.add_constant(H_reg)).
↪fit(cov_type='HAC',  cov_kwds={'maxlags':time_offset, 'use_correction':
↪True})

         # Specify null hypothesis for the joint dynamic quantile test
        hypothesis = '(const= 0, x1 = 0)'  # Null hypothesis: intercept and␣
↪slope are both zero

        # Print the p-value for the joint dynamic quantile test
        p_value_2= model_2.wald_test(hypothesis,scalar=True).pvalue
        print(f"P-value: {p_value_2} ",  end="")

        # Check if the null hypothesis is rejected based on the p-value
        if p_value_2 < alpha:
            print("joint dynamic quantile test: Reject the null␣
↪hypothesis(bad fit)")
        else:
            print("joint dynamic quantile test: Fail to reject the null␣
↪hypothesis(possibly good fit)")
        return p_value_1, p_value_2
    else:
        print("Error: joint dynamic quantile test: Reject the null␣
↪hypothesis(bad fit) (all H are the same)")
        return p_value_1, 0
```

5.3.2 DQ test

With the preceding tests we only performed inference on the properties of unconditional coverage and independence. Thus we are not yet able to detect the misspecification of 5.1.5. Using the properties that we deduced from the $\text{Hit}(\beta_0))_t$-variables, a natural way to set up a test as done in [13] is the following. Given again a sequence of $\{\text{Hit}_t(\hat{\beta})\}_{t=1}^{T}$ is to check whether the test statistic:

$$T^{-1/2}\mathbf{X}^{\top}(\hat{\beta})\mathbf{Hit}(\hat{\beta}) \tag{5.3.5}$$

is significantly different from 0 , where $\mathbf{X}_t(\hat{\beta}), t \in \{1, \ldots, T\}$, the typical row of $\mathbf{X}(\hat{\beta})$ (possibly depending on $\hat{\beta}$) is a q-vector measurable $\mathcal{F}_t$ and $\text{Hit}(\hat{\beta}) \equiv \left[\text{Hit}_1(\hat{\beta}), \ldots, \text{Hit}_T(\hat{\beta})\right]'$.

Essentially what we are doing is to check multiple correlations of the Hit variables with the past information.

To derive the out-of-sample DQ test, let T_R denote the number of training observations and let N_R denote the number of test observations. We will make explicit the dependence of the relevant variables on the number of observations, using appropriate subscripts. Let's define the q-vector measurable $\mathcal{F}_n$, $\mathbf{X}_n\left(\hat{\boldsymbol{\beta}}_{T_R}\right)$, for $n = T_R + 1, \ldots, T_R + N_R$, as the typical row of $\mathbf{X}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)$, possibly depending on $\hat{\boldsymbol{\beta}}_{T_R}$, and $\boldsymbol{Hit}\left(\hat{\boldsymbol{\beta}}_{T_R}\right) \equiv \left[\text{Hit}_{T_R+1}\left(\hat{\boldsymbol{\beta}}_{T_R}\right), \ldots, \text{Hit}_{T_R+N_R}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\right]^{\top}$.

With the following theorem, under the hypothesis of consistency of the estimator $\hat{\beta}$ and some technical assumptions, we show that the statistics for the DQ-test indeed converges in distribution to a normal Gaussian variable. The validity of the result relies heavily on the assumption

$$\lim_{R\to\infty} N_R/T_R = 0$$

which connects the size of the training set with the size of the test set. Given that the result is asymptotic it is not entirely clear how one would need to replicate this condition with a finite sample. For this reason in the following chapters we will explore some simulations to better understand its behavior.

Theorem 5.3.1 (Out-of-sample dynamic quantile test). *Assume that:*

- *Conditional on all of the past information $\mathcal{F}_t$, the error terms $\varepsilon_{t\theta}$ form a stationary process, with continuous conditional density $h_t(\varepsilon \mid \mathcal{F}_t) \leq N < \infty \forall t$, for some constant N.*
- *$f_t(\boldsymbol{\beta})$ is differentiable and $\|\nabla f_t(\boldsymbol{\beta})\| \leq F_0 < \infty$, for some constant F_0, $\|\nabla f_t(\boldsymbol{\beta}) - \nabla f_t(\boldsymbol{\gamma})\| \leq M_0\|\boldsymbol{\beta} - \boldsymbol{\gamma}\| < \infty$ for some constants M_0.*
- *$T_R^{-1/2}(\hat{\beta}_{T_R} - \beta_0)$ obeys the central limit theorem.*
- *$\mathbf{X}_t(\boldsymbol{\beta})$ is measurable $\mathcal{F}_t, \|\mathbf{X}_t(\boldsymbol{\beta})\| \leq W_0 < \infty$, for some constant W_0 and there exist $\|\nabla \mathbf{X}_t(\boldsymbol{\beta})\| \leq Z_0$, for some constant Z_0*
- $\lim_{R\to\infty} T_R = \infty$, $\quad \lim_{R\to\infty} N_R = \infty$, $\quad$ *and* $\lim_{R\to\infty} N_R/T_R = 0$.
- *The sequence $\left\{N_R^{-1/2}\mathbf{X}'(\beta_0)\,\mathbf{Hit}(\beta_0)\right\}$ obeys the central limit theorem.*

Then:

$$
\begin{aligned}
DQ \equiv & \mathbf{Hit}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\mathbf{X}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\left[\mathbf{X}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\cdot\mathbf{X}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\right]^{-1} \\
& \times \mathbf{X}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\mathbf{Hit}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)/(\theta(1-\theta)) \overset{d}{\sim} \chi^2_q \quad as\ R \to \infty
\end{aligned}
$$

Proof. We first approximate the discontinuous function $\mathrm{Hit}_t(\hat{\boldsymbol{\beta}})$ with a continuously differentiable function, for any fixed $\hat{\beta}$. Define

$$
\begin{aligned}
Hit_t^{\oplus}(\hat{\boldsymbol{\beta}}) &\equiv \left[1+\exp\left\{c_T^{-1}\hat{\varepsilon}_t\right\}\right]^{-1} - \theta \\
&\equiv I^*\left(\hat{\varepsilon}_t\right) - \theta
\end{aligned}
$$

where $\hat{\varepsilon}_t \equiv y_t - f_t(\hat{\boldsymbol{\beta}})$ and c_T is a sequence such that $\lim_{T\to\infty} c_T = 0$. Then

$$
\begin{aligned}
\nabla_{\boldsymbol{\beta}} Hit_t^{\oplus}(\hat{\boldsymbol{\beta}}) &= c_T^{-1}\exp\left\{c_T^{-1}\hat{\varepsilon}_t\right\}\left[1+\exp\left\{c_T^{-1}\hat{\varepsilon}_t\right\}\right]^{-2}\nabla f_t(\hat{\boldsymbol{\beta}}) \\
&\equiv k_{c_T}\left(\hat{\varepsilon}_t\right)\cdot\nabla f_t(\hat{\boldsymbol{\beta}}).
\end{aligned}
$$

Note that $k_{c_T}\left(\hat{\varepsilon}_t\right)$ is the pdf of a logistic with mean 0 and parameter c_T. In matrix form, we write $\nabla_{\boldsymbol{\beta}}\mathbf{Hit}^{\oplus}(\hat{\boldsymbol{\beta}}) = \mathbf{K}\left(\hat{\varepsilon}_t\right)\nabla f(\hat{\boldsymbol{\beta}})$, where $\mathbf{K}\left(\hat{\varepsilon}_t\right)$ is a diagonal matrix with entries $k_{c_T}\left(\hat{\varepsilon}_t\right)$. We now prove:

$$
\begin{aligned}
& T^{-1/2}\mathbf{X}'(\hat{\boldsymbol{\beta}})\mathbf{Hit}^{\oplus}(\hat{\boldsymbol{\beta}}) \\
&= T^{-1/2}\sum_{t=1}^{T}\left[\mathbf{X}_t'(\hat{\boldsymbol{\beta}})Hit_t^{\oplus}(\hat{\boldsymbol{\beta}})\right] \\
&\quad + o_p(1)
\end{aligned}
$$

Since we assumed $||\mathbf{X}_t(\hat{\boldsymbol{\beta}})|| \leq W_0$ we only need to bound $\left|Hit_t^{\oplus}\left(\hat{\beta}\right) - Hit_t\left(\hat{\beta}\right)\right|$.

We observe that due to the consistency of $\hat{\beta}$ we have:

$$
\left|Hit_t^{\oplus}\left(\hat{\beta}\right) - Hit_t\left(\hat{\beta}\right)\right| = \left|Hit_t^{\oplus}\left(\beta_0\right) - Hit_t\left(\beta_0\right)\right| + o_p(1) \tag{5.3.6}
$$

Noting that $I^*\left(|\varepsilon_{t\theta}|\right) = 1 - I^*\left(-|\varepsilon_{t\theta}|\right)$, we have, for each t,

$$
\begin{aligned}
& \left|Hit_t^{\oplus}\left(\beta_0\right) - Hit_t\left(\beta_0\right)\right| \\
& \leq I^*\left(|\varepsilon_{t\theta}|\right)\left[I\left(|\varepsilon_{t\theta}| \geq T^{-d}\right) + I\left(|\varepsilon_{t\theta}| < T^{-d}\right)\right] \\
& \equiv C_t + D_t
\end{aligned}
$$

where d is a positive number greater than 1/2, such that $\lim_{T\to\infty} c_T T^d = 0$. Therefore

$$
\begin{aligned}
&T^{-1/2}\sum_{t=1}^{T}\left\|\mathbf{X}_t(\beta_0)\left[Hit_t^{\oplus}(\beta_0)-\mathrm{Hit}_t(\beta_0)\right]\right\| \\
&\quad\leq T^{-1/2}W_0\sum_{t=1}^{T}\cdot\left|Hit_t^{\oplus}(\beta_0)-Hit_t(\beta_0)\right| \\
&\quad\leq T^{-1/2}W_0\sum_{t=1}^{T}\cdot(C_t+D_t),
\end{aligned}
$$

where $C_t\equiv I^*(|\varepsilon_{t\theta}|)\cdot I\left(|\varepsilon_{t\theta}|\geq T^{-d}\right)$ and $D_t\equiv I^*(|\varepsilon_{t\theta}|).\ I\left(|\varepsilon_{t\theta}|<T^{-d}\right)$.

Noting that $I^*(|\varepsilon_{t\theta}|)$ is decreasing in $|\varepsilon_{t\theta}|$, we have $C_t\leq I^*\left(T^{-d}\right)$. Therefore,

$$
T^{-1/2}\sum_{t=1}^{T}C_t\leq T^{1/2}W_0\left[1+\exp\left(c_T^{-1}T^{-d}\right)\right]^{-1}\xrightarrow{T\to\infty}0.
$$

For D_t, note that $D_t\leq I\left(|\varepsilon_{t\theta}|<T^{-d}\right)$, because $I^*(|\varepsilon_{t\theta}|)$ is bounded between 0 and 1 . Therefore, for any $\xi>0$,

$$
\begin{aligned}
&T^{-1/2}W_0\sum_{t=1}^{T}\Pr(D_t>\xi) \\
&\quad\leq T^{-1/2}\xi^{-1}W_0\sum_{t=1}^{T}E\left[\int_{-T^{-d}}^{T^{-d}}s_t(\lambda\mid\Omega_t)\,d\lambda\right] \\
&\quad\leq T^{-1/2}\xi^{-1}W_0\sum_{t=1}^{T}\cdot 2T^{-d}N \\
&\quad=2\xi^{-1}W_0NT^{-d+1/2}\xrightarrow{T\to\infty}0.
\end{aligned}
$$

We now apply the mean value expansion to the continuous approximation,

$$
\begin{aligned}
&N_R^{-1/2}\mathbf{X}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\mathbf{Hit}^{\oplus}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)= \\
&\qquad N_R^{-1/2}\left\{\mathbf{X}'(\beta_0)\,\mathbf{Hit}^{\oplus}(\beta_0)\right. \\
&\qquad\left.+\left[\nabla\mathbf{X}(\boldsymbol{\beta}^*)\,\mathbf{Hit}^{\oplus}(\boldsymbol{\beta}^*)+\mathbf{X}(\boldsymbol{\beta}^*)\,\mathbf{K}(\varepsilon_t^*)\,\nabla f(\boldsymbol{\beta}^*)\right]\left(\hat{\boldsymbol{\beta}}_{T_R}-\beta_0\right)\right\},
\end{aligned}
$$

where $\boldsymbol{\beta}^*$ lies between $\hat{\boldsymbol{\beta}}$ and β.

We rewrite:

$$
\begin{aligned}
&\lim_{R\to\infty}N_R^{-1/2}\mathbf{X}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\mathbf{Hit}^{\oplus}\left(\hat{\boldsymbol{\beta}}_{T_R}\right) \\
&\quad=\lim_{R\to\infty}\left\{N_R^{-1/2}\mathbf{X}'(\beta)\,\mathbf{Hit}^{\oplus}(\beta)\right. \\
&\qquad\left.+\left(\frac{N_R}{T_R}\right)^{1/2}\times\frac{1}{N_R}\left[\nabla\mathbf{X}(\boldsymbol{\beta}^*)\,\mathbf{Hit}^{\oplus}(\boldsymbol{\beta}^*)\ +\mathbf{X}(\boldsymbol{\beta}^*)\,\mathbf{K}(\varepsilon_t^*)\,\nabla f(\boldsymbol{\beta}^*)\right]T_R^{1/2}\left(\hat{\boldsymbol{\beta}}_{T_R}-\beta\right)\right\}
\end{aligned}
$$

We now focus on proving that there exists a constant C such that for any R large enough:

$$\frac{1}{N_R}\left[\nabla \mathbf{X}\left(\boldsymbol{\beta}^*\right) \mathbf{Hit}^{\oplus}\left(\boldsymbol{\beta}^*\right)+\mathbf{X}\left(\boldsymbol{\beta}^*\right) \mathbf{K}\left(\varepsilon_t^*\right) \nabla f\left(\boldsymbol{\beta}^*\right)\right] \leq C \tag{5.3.7}$$

Again due to consistency this is equivalent to prove:

$$\frac{1}{N_R}\left[\nabla \mathbf{X}\left(\beta_0\right) \mathbf{Hit}^{\oplus}\left(\beta_0\right)+\mathbf{X}\left(\boldsymbol{\beta_0}\right) \mathbf{K}\left(\varepsilon_{t\theta}\right) \nabla f\left(\beta_0\right)\right] \leq C \tag{5.3.8}$$

The first term is bounded since $||\nabla X_t(\boldsymbol{\beta}^*)|| \leq Z_0$, for any t.

For the second term lets consider $\mathbf{H}$ the diagonal matrix with typical entry $h_t(0|\mathcal{F}_t)$. We observe that by hypothesis $N_R^{-1}\left[\mathbf{X}'\left(\boldsymbol{\beta}^0\right) \mathbf{H} \nabla f\left(\boldsymbol{\beta}^0\right)\right]$ is also bounded. So we need to prove that

$$N_R^{-1}\left[\mathbf{K}\left(\varepsilon_{t\theta}\right)-\mathbf{H}\right]=o_p(1)$$

We write

$$\begin{aligned}
& N_R^{-1}\left[\mathbf{K}\left(\varepsilon_{t\theta}\right)-\mathbf{H}\right] \\
& =N_R^{-1} \sum_{t=1}^{N_R}\left[k_{c_{N_R}}\left(\varepsilon_{t\theta}\right)-E\left(k_{c_{N_R}}\left(\varepsilon_{t\theta}\right) \mid \Omega_t\right)\right] \\
& \quad+N_R^{-1} \sum_{t=1}^{N_R}\left[E\left(k_{cN_R}\left(\varepsilon_{t\theta}\right) \mid \mathcal{F}_t\right)-h_t\left(0 \mid \mathcal{F}_t\right)\right]
\end{aligned}$$

First, we show that the expected value of $k_{C_{N_R}}\left(\varepsilon_{t\theta}\right)$, given $\mathcal{F}_t$, converges to $h_t\left(0 \mid \mathcal{F}_t\right)$. Let $k(u) \equiv e^u\left[1+e^u\right]^{-2}$. Then

$$\begin{aligned}
& E\Big[k_{c_{N_R}}\left(\varepsilon_{t\theta}\right) \mid \mathcal{F}_t\Big] \\
& =\int_{-\infty}^{\infty} k(u) h_t\left(u c_{N_R} \mid \mathcal{F}_t\right) d u \\
& =\int_{-\infty}^{\infty} k(u)\left[h_t\left(0 \mid \mathcal{F}_t\right)+s_t'\left(0 \mid \mathcal{F}_t\right) u c_{N_R}+o\left(c_{N_R}\right)\right] d u \\
& =h_t\left(0 \mid \mathcal{F}_t\right)+o\left(c_{N_R}\right)
\end{aligned}$$

where in the first equality we performed a change of variables, in the second we applied the Taylor expansion to $h_t\left(u c_{N_R} \mid \mathcal{F}_t\right)$ around 0 , and the last equality comes from the fact that $k(u)$ is a density function with first moment equal to 0 .

Next, we need to show that

$$N_R^{-1} \sum_{t=1}^{N_R}\left[k_{c_{N_R}}\left(\varepsilon_{t\theta}\right)-E\left(k_{c_{N_R}}\left(\varepsilon_{t\theta}\right) \mid \mathcal{F}_t\right)\right] \mathbf{X}_t'\left(\boldsymbol{\beta}^0\right) \nabla f_t\left(\boldsymbol{\beta}^0\right)=o_p(1). \tag{5.3.9}$$

We observe that it has 0 expectation. We prove that its variance converges to 0 , then the result follows from the application of the Chebyshev inequality.

$$
\begin{aligned}
&E\left[\left[k_{c_T}\left(\varepsilon_{t\theta}\right)-E\left(k_{c_T}\left(\varepsilon_{t\theta}\right)\mid\mathcal{F}_t\right)\right]^2\mid\mathcal{F}_t\right]\\
=&E\left[k_{c_T}\left(\varepsilon_{t\theta}\right)^2\mid\mathcal{F}_t\right]-E\left[k_{c_T}\left(\varepsilon_{t\theta}\right)\mid\mathcal{F}_t\right]^2\\
=&\int_{-\infty}^{\infty}k_{c_T}(\lambda)^2h\left(\lambda\mid\mathcal{F}_t\right)d\lambda-h\left(0\mid\mathcal{F}_t\right)^2+o\left(c_T\right)\\
=&c_T^{-1}\int_{-\infty}^{\infty}k(u)^2h\left(uc_T\mid\mathcal{F}_t\right)du-h\left(0\mid\mathcal{F}_t\right)^2+o\left(c_T\right)\\
\leq&1/4c_T^{-1}\int_{-\infty}^{\infty}k(u)\left[h\left(0\mid\mathcal{F}_t\right)+h'\left(0\mid\mathcal{F}_t\right)uc_T+o\left(c_T\right)\right]du\\
&-h\left(0\mid\mathcal{F}_t\right)^2+o\left(c_T\right)\\
\leq&1/4c_T^{-1}\left[h\left(0\mid\mathcal{F}_t\right)+o\left(c_T\right)\right]-h\left(0\mid\mathcal{F}_t\right)^2+o\left(c_T\right)\\
=&O\left(c_T^{-1}\right).
\end{aligned}
$$

Finally since by hypothesis $\frac{N_R}{T_R}\to_R 0$ then we conclude:

$$
\lim_{R\to\infty}N_R^{-1/2}\mathbf{X}'\left(\hat{\boldsymbol{\beta}}_{T_R}\right)\mathbf{Hit}^{\oplus}\left(\hat{\boldsymbol{\beta}}_{T_R}\right)=\lim_{R\to\infty}N_R^{-1/2}\mathbf{X}'\left(\beta_0\right)\mathbf{Hit}^{\oplus}\left(\beta_0\right)=\lim_{R\to\infty}N_R^{-1/2}\mathbf{X}'\left(\beta_0\right)\mathbf{Hit}'\left(\beta_0\right)
$$

where the last equality can be proven in the same way that we proved the first step of the proof. The result then follows from the assumption on $N_R^{-1/2}\mathbf{X}'\left(\beta_0\right)\mathbf{Hit}'\left(\beta_0\right)$.

$\square$

Example of code

```
[ ]: def outofsample_DQ_test(Y,quant_pred_Y, X, q, alpha=0.05):
       """
         Conducts an out-of-sample DQ test for quantile regression.

         Parameters:
         - Y (np.ndarray): Observations
         - quant_pred_Y (np.ndarray): Prediction of quantile
         - X (np.ndarray): Vector of q variables measurable at time t (past␣
      ↪return values for quantile regression)
         - q (float): Quantile
         - alpha (float): Threshold for rejecting the null hypothesis (default␣
      ↪is 0.05)

         Returns:
         - value (float): Chi-squared statistic
         - p_value (float): P-value of the test
         """
       T,p=X.shape

       # Calculate the Hit variable
       H=(Y<quant_pred_Y).astype(np.float64)-q

       aux_mat=X.T@H
         # Check if X.T @ X matrix is invertible
       if np.linalg.det(X.T@X) == 0:
           print('grad_f'+str(X.T@X))
           print('D is not invertible')
           return

         # Calculate the test statistic value
       value= aux_mat.T @ np.linalg.solve(X.T@X ,aux_mat )/(q*(1-q))
       print(value)
     # Calculate the p-value using the chi-squared distribution
       p_value = 1 - stats.chi2.cdf(value, p)
       print(f"P-value: {p_value} ",  end="")

     # Check if the null hypothesis is rejected based on the p-value
       if p_value < alpha:
           print("Outofsample test: Reject the null hypothesis(bad fit)")
       else:
           print("Outofsample test: Fail to reject the null␣
      ↪hypothesis(possibly good fit)")
       return p_value
```

One final remark about the implementation of this test concerns the choice of the columns of $\mathbf{X}$. One can effectively choose almost any data that is available, however results can be very different. Following the suggestion of [14] we always pick the past lags of y_t as the columns of our filtration matrix.

5.4 Final remarks

The assumption on the first test might look a little bit unrealistic since we assuming a linear model on discrete variables of only two possible outcomes. We explore a little more the theoretical properties.

Let's now consider a common simpler linear regression model:

$$y_i = x_i \delta^T + \epsilon_i$$

now ϵ_i are iid normal Gaussian variables centered in 0 and with variance σ^2..

In this framework the Wald statistic to use to test whether $\delta = 0$ through the OLS estimator assumes the form of:

$$(\hat{\delta}_n^\top - 0) I(0)^{-1} (\hat{\delta} - 0) = \frac{Y^\top X (X^\top X)^{-1} X^\top Y}{\sigma^2}$$

Where Y and X are the vectors formed by the observed x and y respectively. Being in an exponential model, the estimates are correct and we indeed converge to a $\chi^2(k)$ variable under the hypothesis. We also remark that it is not necessary to know σ as it can be estimated from the data as the mean squared error of the model under the null hypothesis.

If we apply this instance of Wald test to our framework with the $Hit_t(\hat{\beta})_{t=1}^T$ sequence, where we assess past lag correlation, we get:

$$\frac{HIT_t^\top HIT_{t-1}(HIT_{t-1}^\top HIT_{t-1})^{-1} HIT_{t-1}^\top HIT_t}{\sigma^2}$$

Where HIT_{t-i} is the vector of of $Hit_i(\hat{\beta})$-s starting at time $1-i$ and ending at time $T-1+i$. Since σ^2 is unknown needs to be estimated under the null hypothesis. However the null hypothesis is $\beta = 0$ σ^2 it will simply correspond to the variance of the Hit_t variable, $\tau(1-\tau)$.

We can now connect this simplified version of the JDQ test that was performed under some unrealistic hypothesis and the out of sample DQ test.

In fact we notice that both statistics are calculated using the HIT_t variables. Furthermore, since HIT_{t-1} is measurable according to past filtrations our Wald test satisfies the out of sample DQ test. The only real difference is in the fact that we use an estimated version of the variance of Y, since we know its distribution under the null hypothesis, (assuming that the quantile model was fitted with enough data). However the proof of the DQ test theorem easily extends also in this case. A similar reasoning can be applied to the UC test.

We furthermore notice that we can also perform JDQ test as:

$$\frac{HIT_t^\top HIT_{t-1} HIT_{t-1}^\top HIT_t}{n(\tau(1-\tau))^2}$$

Since $HIT_{t-1}^\top HIT_{t-1}$ is n times the estimated variance of $Hit(\beta)_{t-1}$ whose distribution is again known. One interesting observation, is that since we know the distribution of all the variables at use, we can also estimate the speed of convergence to the Normal distribution provided of the central limit theorem.

In fact we know that $\{Hit_i(\hat{\beta})\}_{i=0}^{\infty}$ under the Null hypothesis is a sequence of iid random variable where Hit_1 is $-\tau$ with probability $1-\tau$ and $1-\tau$ with probability τ. We call $Z_i = Hit_i Hit_{i-1}$ for $i = 1, \dots \infty$ and:

$$S_n = \frac{1}{n}\sum_{i=1}^{\infty} Z_i$$

We highlight that $\mathbb{E}[Z_i] = 0$. We want to prove some bounds for the following quantity:

$$\sup_{x\in\mathbb{R}} \left| \mathbb{P}\left(\frac{\sqrt{n}S_n}{(\tau(1-\tau))^2} \leq x \right) - \Phi(x) \right| \tag{5.4.1}$$

where $\Phi(x)$ is the cumulative distribution function of the standard Gaussian variable. We observe that $\{Z_i\}_{i=1}^{\infty}$ is a stationary discrete time Markov chain, with states

$$[(1-\tau)^2, -\tau(1-\tau), \tau^2]$$

, transition matrix:

$$\begin{bmatrix} \tau & 1-\tau & 0 \\ \tau/2 & 1/2 & (1-\tau)/2 \\ 0 & \tau & 1-\tau \end{bmatrix} \tag{5.4.2}$$

and starting vector of probabilities:

$$\begin{bmatrix} \tau^2 & 2\tau(1-\tau) & (1-\tau)^2 \end{bmatrix} \tag{5.4.3}$$

With this framework there some results that extends the ones of Berry-Esseen, for example ([15]), and state that the convergence is indeed of rate $O(\sqrt{n})$,

The argument is even simpler in the case when the regression is performed against a constant vector of all 1s. In that case the equivalence between Wald test and out-of-sample DQ test still holds and the speed of convergence is guaranteed by The classical Berry-Esseen theorem applied to the random variable $Z_i = Y_i$.

When performing multivariate regression against both multiple lags of the Hit variables and the constant variable, the equivalence between the tests is again satisfied, without further analysis, since, under the null hypothesis, the Fisher information matrix has non zero elements only in the diagonal.

With little struggle one can also write the transition matrix for the regression of each of the other lags of the Hit variables, and retrieve that indeed the convergence to the normal distribution is again $O(\sqrt{n})$.

Chapter 6

Experiments

In order to assess the performance of our model, multiple experiments have been conducted. The code was written in Python version 3.8.8, making extensive use of the library hmmlearn [6], that provided the baseline for the model to be written. Among others sklearn was the library that provided an already implemented method for base quantile regression, while we used torch [10] and in particular the torch.optim implementation of Adam algorithm and scipy.optimize for the implementation of Nelder Mead algorithm. Finally we made use of the scipi.stats library to help the implementation of the statistical tests. Datetime was the main library for the managing of the real data.

6.1 Preliminary experiment

The first experiment consisted in fitting the model with artificial data generated by a hmm model.

We consider a very simple parametrization for our artificial data and set the parameters to be:

$$A = \begin{bmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{bmatrix} \qquad \beta_0 = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}, \beta_1 = \begin{pmatrix} 0 \\ -3 \\ 0.5 \end{pmatrix} \tag{6.1.1}$$

where the first term of the β-s is the bias term and the vector of initial probabilities is set as $\pi = \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}$.

We then sampled the first state i using π and by sampling two normal random variables $x_1 \sim \mathcal{N}(0,1), x_2 \sim \mathcal{N}(3,0.5)$ proceeded to compute the y_1 return variable as

$$y_1 = \beta_i \cdot \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} + \epsilon_1 \quad i = 0,1 \tag{6.1.2}$$

where β_i are some state dependent coefficients, $\cdot$ represents the scalar product, and ϵ_1 is an error term with distribution $\mathcal{N}(0,0.1)$.

The next state is then sampled using the transition probabilities given from the i-th line of the matrix A and after sampling the x components, y_2 is generated. The process is thus repeated until our dataset of 1000 points is generated. We observe that the only thing that relates present and past data, is thus the Markov switching part: we do not assume autoregression of the return variable.

Given this set we fit our different models trying to predict the 0.5 quantile. We observe that in this framework the exact parameters that we chose should be retrieved after training our model.

We try to assess the performance of all 3 optimization algorithms that we presented, thus for this purpose, for each we perform linear quantile regression, and optimize using linear programming, Adam and Nelder-Mead respectively. In all the tests the initialization was performed through k-means algorithm due to its generally faster performance with respect to the random approach. For the Adam and Nelder-Mead we first tried performing restart between the iteration of EM, that is at each iteration we picked the starting point of the optimization algorithm so that it doesn't depend of the previous iteration. The parameters of Adam optimiser were fixed to having learning rate equal to 0.01, and number of batches equal to 10, granting thus a minibatch size of 100.

After fitting we retrieve the following parameters:

- for linear programming optimization we got:

$$A^{lin} = \begin{bmatrix} 0.950 & 0.050 \\ 0.150 & 0.850 \end{bmatrix} \qquad \beta_0^{lin} = \begin{pmatrix} -0.079 \\ 2.002 \\ 0.985 \end{pmatrix}, \beta_1^{lin} = \begin{pmatrix} -0.178 \\ -2.998 \\ 0.519 \end{pmatrix}$$

- for Adam, with restart, we got:

$$A^{AdamR} = \begin{bmatrix} 0.951 & 0.049 \\ 0.148 & 0.852 \end{bmatrix} \qquad \beta_0^{AdamR} = \begin{pmatrix} -0.005 \\ 1.998 \\ 1.003 \end{pmatrix}, \beta_1^{AdamR} = \begin{pmatrix} -0.005 \\ -2.995 \\ 0.504 \end{pmatrix}$$

- for Adam, without restart, we got:

$$A^{AdamN} = \begin{bmatrix} 0.951 & 0.049 \\ 0.148 & 0.852 \end{bmatrix} \qquad \beta_0^{AdamN} = \begin{pmatrix} -0.005 \\ 1.998 \\ 1.003 \end{pmatrix}, \beta_1^{AdamN} = \begin{pmatrix} -0.005 \\ -2.995 \\ 0.504 \end{pmatrix}$$

- for Nelder-Mead, with restart, we got:

$$A^{NMR} = \begin{bmatrix} 0.957 & 0.043 \\ 0.129 & 0.871 \end{bmatrix} \qquad \beta_0^{NMR} = \begin{pmatrix} -0.010 \\ 2.001 \\ 1.004 \end{pmatrix}, \beta_1^{NMR} = \begin{pmatrix} -3.374 \\ -2.787 \\ 1.598 \end{pmatrix}$$

- for Nelder-Mead, without restart, we got:

$$A^{NMN} = \begin{bmatrix} 0.951 & 0.049 \\ 0.148 & 0.852 \end{bmatrix} \qquad \beta_0^{NMN} = \begin{pmatrix} -0.004 \\ 2.002 \\ 1.002 \end{pmatrix}, \beta_1^{NMN} = \begin{pmatrix} -0.026 \\ -2.990 \\ 0.509 \end{pmatrix}$$

We observe that the only model that does not look suitable for this task is the one that used the Nelder-Mead optimization with restart. It seems that although all other quantities have been predicted relatively well, the algorithm failed to retrieve the parameters related to the second state, having β_1^{NMR} being completely unrelated to the real value. On the contrary, allowing the new starting point of the maximization step of EM to be the previous optimum point, seem to retrieve convergence to the right parameters. The opposite seems to happen with the Adam optimization, as it seems that both the restart and not-restart version, converged to the same optimum point.

It appears that all the optimizations apart from NMR, seem to have learned, with little errors the true parameters of the data, thus showing that the Markov Switching Quantile Regression model is able to capture information relative to the data, in spite of its unrealistic assumption on the emission distribution. It also seems that although the Adam, and Nelder-Mead algorithms might not end the maximization step in an actual maximum of the objective function, convergence to optimum of the whole algorithm may still be retrieved.

6.2 Real data experiments

In this section we explore the performance of the Markov Switching Quantile Regression model with real economical data. For this section we will consider only frequency homogeneous data points, and consequently we will look at the performance of only the linear quantile regression, with the linear programming algorithm.

This research leverages a dataset kindly provided by the European Central Bank (ECB) encompassing macroeconomic variables. The multifrquency economical variables of US used for the experiments with the exponential Almon lag specification were instead obtained from [16]. The data traces the economic and social conditions of the European Union and the United States, spanning various start years. he setting in all our experiments will be prediction of the median of yearly differences of log of GDP.

In both datasets the variables are sampled monthly, in spite of the fact GDP is usually calculated on a quarterly base, as preliminary tests showed that by considering quarterly variables, our dataset would end up being too small to fit our model. Thus an estimated version of GDP using external economical variables was used in order to fill and interpolate data during each year.

In order to perform our task and each variable was transformed to be used as yearly differences or yearly differences of log according to the nature of such variables.

The experiments, focused on training until 2016, and using the following datapoints until 2018 perform model selection on the lasso parameter. Testing was then performed on data from 2018 until 2020. The reason for this choice although not ideal, is caused by the lack of data necessary for training on the one hand, but also the unpredictable and disruptive effect that Covid 2019 had on economy on the other. If for the first reason we would want to use as much data as possible, the data sampled from 2020 until 2022 couldn't be correctly predicted using our variables, leading to very poor results of our model.

In order to initialize the expectation maximization we leveraged an external variable whose role was to represent period of normal economy or recession in the considered state.

With both datasets, we fit a 2-states quantile regression model and we apply lasso regularization with a parameter chosen through the validation set according to the likelihood

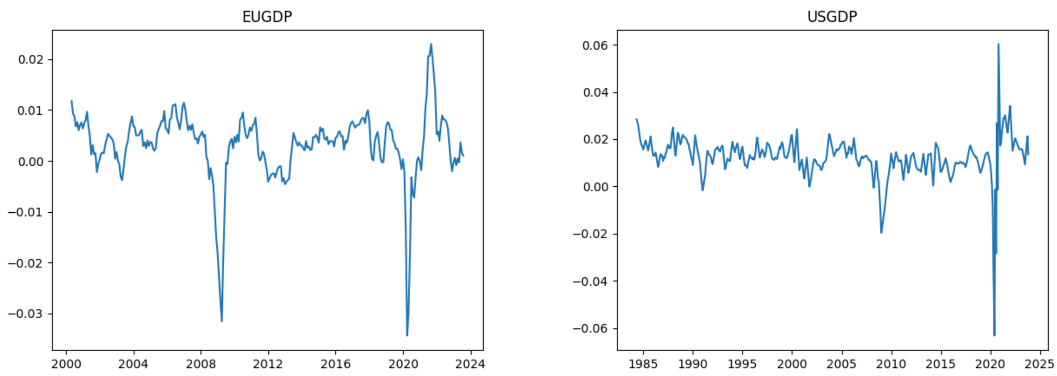


Figure 6.1: plot of yearly log differences of GDP in EU and US in our dataset

score, with the set of possible α parameter being $\{j10^{-i}\}$ for $i = 1, \ldots, 5$ and $j = 0, \ldots, 9$.

We now look at the results:

6.2.1 EU data

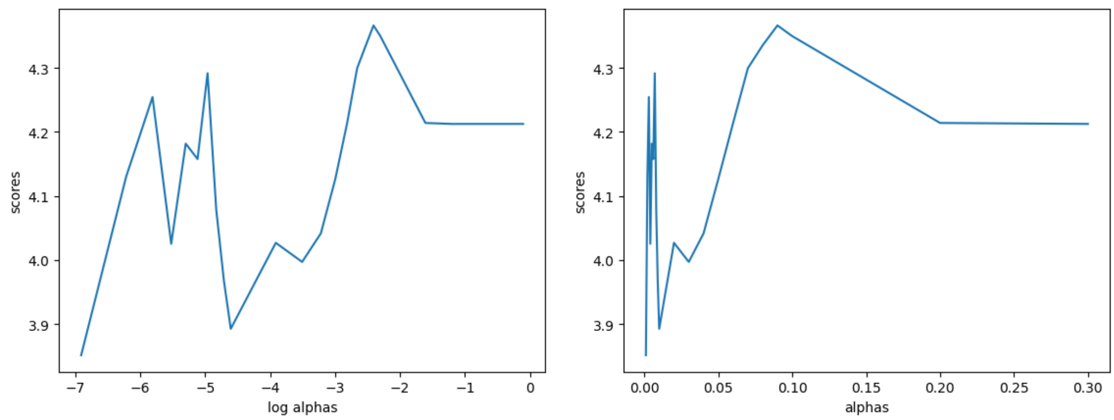


Figure 6.2: Likelihood score on the validation set for different α of lasso regularization

We notice that after some noisy scores, probably due to overfitting, the best value was provided for $\alpha = 0.09$, so we use this value to train the model on the time period until 2018. We observe how the regime identification matches with our economical intuition:

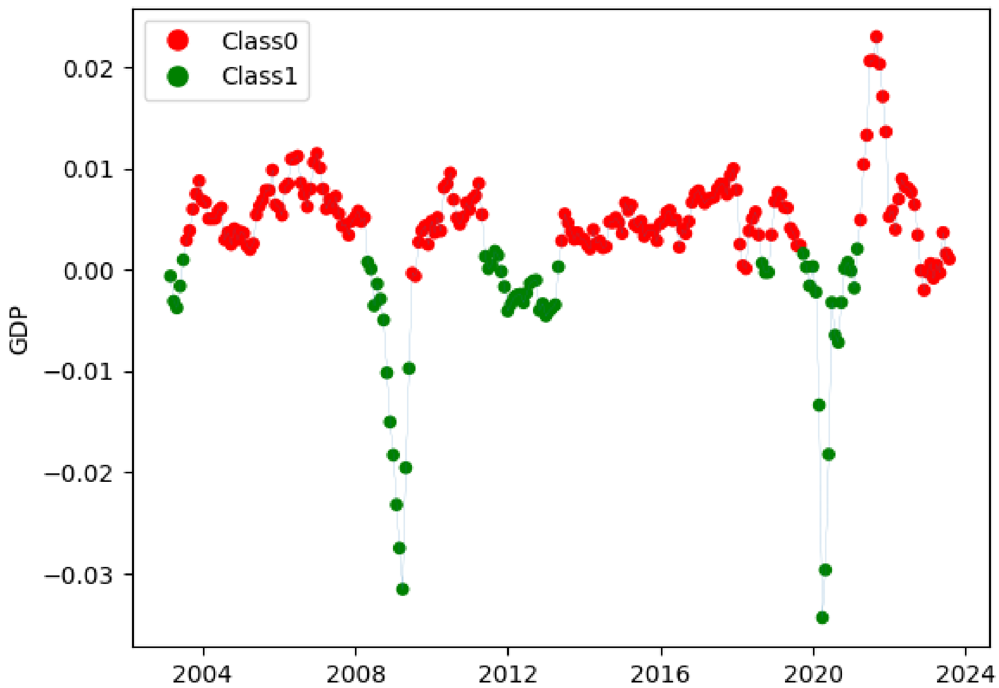


Figure 6.3: regime detected by the model given all data until 2023

score of train set	score of test set	uc test	jdq test	DQ test
4.435	4.389	0.74	0.11	0.61

Table 6.1: normalized likelihoods and p-values for the tests

The model seem to effectively distinguish between normal and abnormal financial and economic regimes, as evidenced by the figure. Notably, it captures major economic events like the 2007-2008 financial crisis, the European debt crisis's impact, and the COVID-19 pandemic.

We observe that the tests too provide encouraging results. We notice that the normalized likelihoods appear to be similar, suggesting that the $L1$ regularization managed to overcome overfitting. The tests all give acceptable p-values, thus we cannot conclude that our model does not capture the true data distribution.

Finally we can use the model for interpretation of the role played by each variable in the prediction of the median of future GDP:

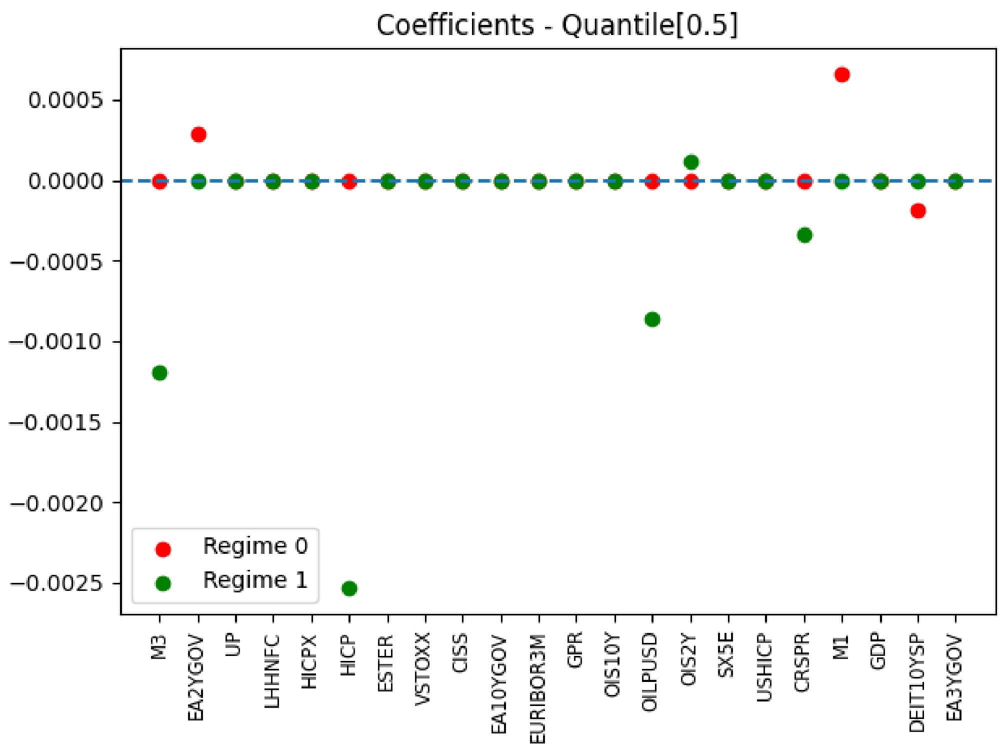


Figure 6.4: Coefficients of quantile regression according to regime

As we can see lasso left only a handful of variables to be relevant to our model, making it easier for interpretation. One curious phenomenon that we observe is that we expected to see some autoregressive behavior for lags of GDP, that in this model appear to be obscured by other variables. We also observe that generally it appears that inflation (HICP) affects GDP median during abnormal financial events, and they are negatively correlated. The same happens for the price of oil (OILPUSD) and financial systemic risk (CRSPR). These results are consistent with economic theory, as an in increase of any of these variables is often connected with price uncertainty, one of the main negative factors that affect economy. We observe that also the broad money variable (M3) seem to affect GDP negatively, during periods of economical stress. This is unsurprising as it is known that the amount of money circulating in an economy is connected with an increase in inflation. However it seems that a different measure of the money supply, M1, is positively correlated with GDP we can this interpret this as a phenomenon that is strictly dependent on the current economical situation.

6.2.2 US data

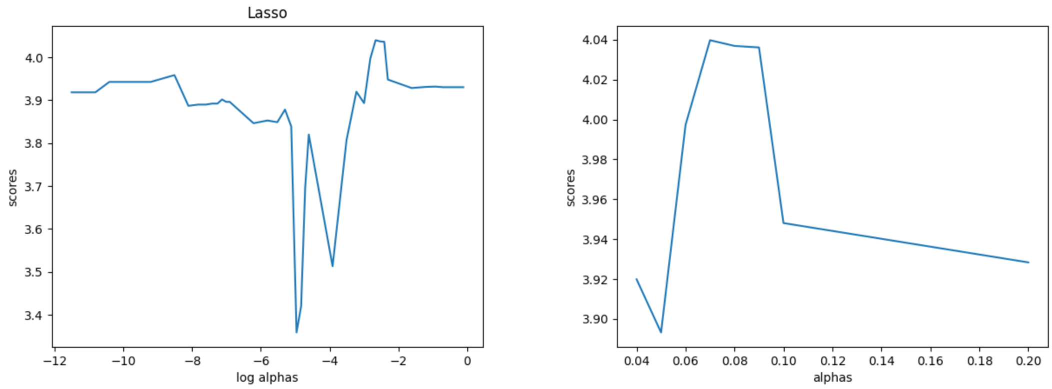


Figure 6.5: likelihood score on the validation set for different α of lasso regularization

We repeat the same experiment with the US data. Firstly we notice again that after some noisy scores, again probably due to overfitting, the best value was provided for $\alpha = 0.07$, and we use this value to train the model on the time period until 2018. Again regime identification matches with our economical intuition:

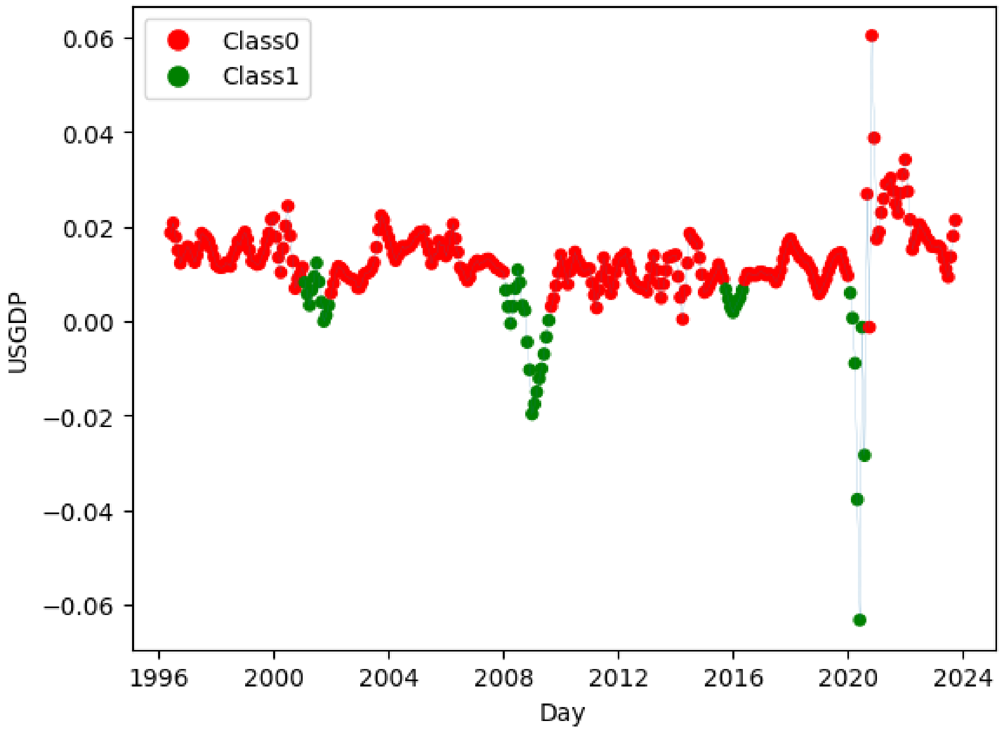


Figure 6.6: regime detected by the model given all data until 2023

Here we can appreciate also other major economical events that affected the US. The longer time scale of the data allows us to identify the early 2000 recession. We also visualise the 2008 worldwide crisis. Contrary to EU that experienced another crisis in 2012, in the US that time period is classified as non-abnormal. On the other hand 2016 is classified as abnormal probably due to the 2015–2016 stock market selloff. Once again we can identify the abnormal economic state during Covid.

score of train set	score of test set	uc test	jdq test	DQ test
4.068	4.374	1	0.23	0.91

Table 6.2: normalized likelihoods and p-values for the tests

For this dataset we notice that the normalized likelihoods appear again to be similar, though, probably due to an absence of exceptional events in the time period 2018-2020, the score appears to be somewhat higher than the one of the training set. The tests again all return acceptable p-values, thus we cannot conclude that our model does not capture the true data distribution.

Again, we can use the model for interpretation of the role played by each variable in the prediction of the median of future GDP:

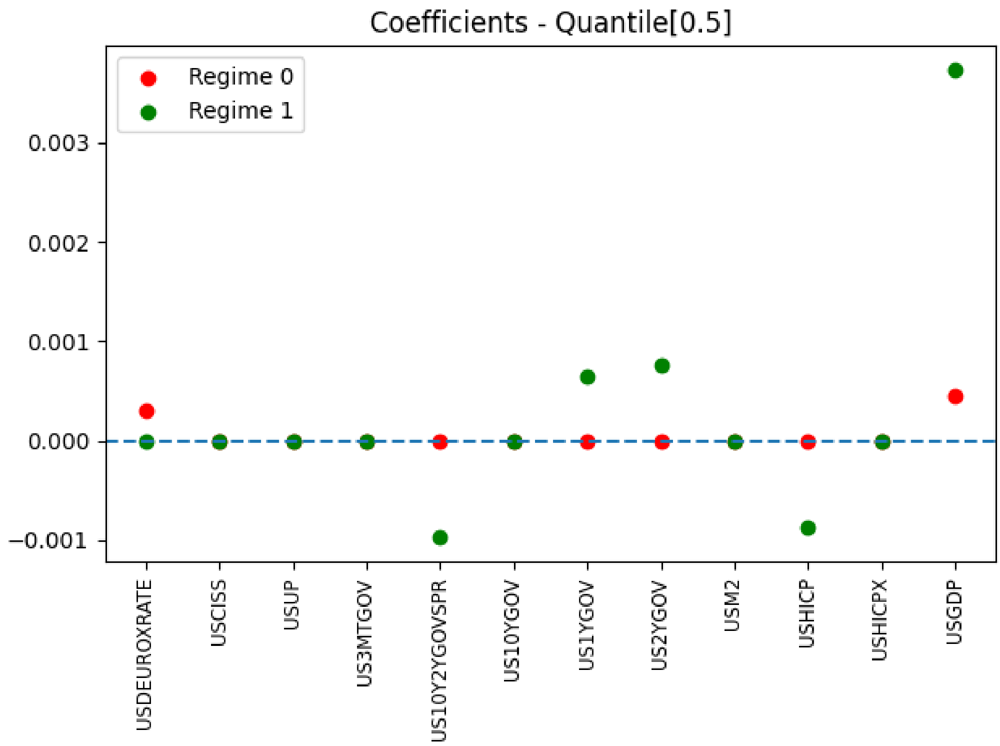


Figure 6.7: Coefficients of quantile regression according to regime

Again only few variables survived the $L1$ regularization. However we notice that with this set of data, the autoregressive behavior of GDP is now captured by the model in both regimes, being actually stronger during the periods of economical distress. We retrieve the same interpretation for inflation. We also get that an increase of interest rates for government bond (US10YGOV and US1YGOV) seems to be correlated with an increase of GDP during period of financial distress. We observe that instead the difference between 10 year treasury rate and the 2 year treasury rate seems to affect negatively GDP, confirming once again known economic theory.

6.3 Multifrequency

In this section we explore the model, with multifrequency data, that is with variables that can be sampled monthly, weekly or daily. As we explained in previous sections, we performed linear quantile regression, but due to parameter proliferation, we imposed a structure on coefficients sampled at different times of the same variable, given by the Almon polynomials. Due to the new nature of our problem we could no longer use linear programming and started using non linear optimization algorithms, one gradient based in the form of the Adam algorithm and the other euristic in the form of Nelder-Mead algorithm.

6.3.1 A simpler experiment

To assess the performance of these algorithms in the framework of expectation maximization with real data, we considered a simplified problem first.

We considered just two variables, chosen to have significant economic meaning: USHICPX, USCISS and together with past lags of USGDP were used to predict the median of USGDP. HICPX represents a measure of the inflation, while CISS represent a measure of the systemic risk,in other words how likely is for a single event to cause widespread failure throughout an entire system, like the financial crisis where a bank collapse could cripple the whole economy. The models were trained on US data until 2008. We don't expect great predictive capabilities from such a choice of variables, but what we do expect is the models to pick up information on the nature of the variables. We describe the economic variables in the appendix.

First of all we trained our model using linear programming and obtained

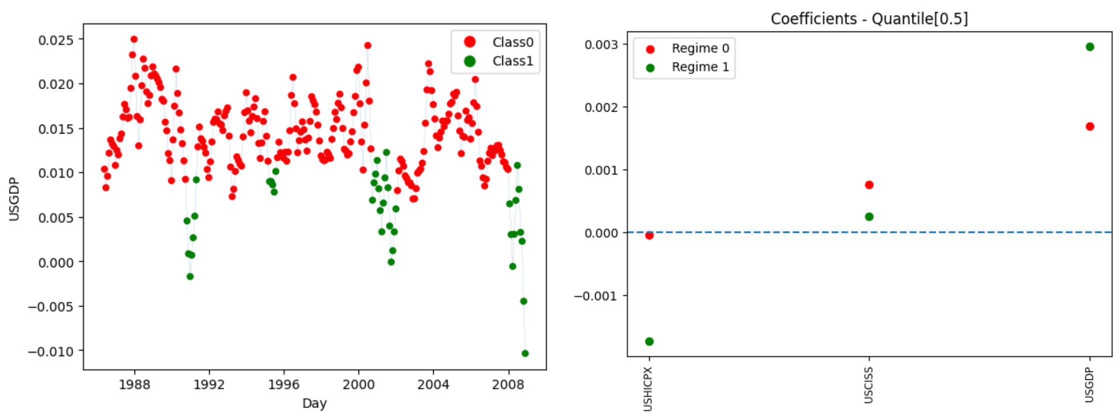


Figure 6.8: Results for the simplified framework with linear programming optimization

As we can see the trained model effectively picks up information on the state of the economy during the different time periods. Given that our time window if further in the past, we now were able to classify the early 1990's recession. Moreover the coefficient associated with the predicting variables appear to be consistent with some aspects of the economical interpretations of the variables: we get GDP positively self-correlated and inflation to be negatively correlated. As far as systemic risk is concerned we can interpret our results as the fact that in normal economical conditions, high risk isn't likely to actually cause a crisis.

Seeing the performance of the linear programming algorithm we then go on and try to fit the same model using Adam algorithm with and without restart. We also tried cross validating the hyperparameters of the algorithm leading to a choice of:

- epochs in $\{10, 20, 50, 100, 1000\}$, where with epochs we mean the number of iterations of Adam at each maximisation step of EM;

- learning rate in $\{0.01, 0.001, 0.0001\}$

- number of batches in $\{1, 2, 5, 10, 20, 50\}$ plus the size of the whole dataset. With these choice of parameters, 1 is thus the Gradient Descent algorithm, while the whole dataset creates minibatches of 1 datapoint.

However a very first inspection showed failure for almost all setting of parameters in both cases.

We observed that in very few cases the model actually recognized the presence of two regimes, seemingly without an easily derivable criterion of choice for our hyperparameters. For this reason we decided to discard the usage of Adam as an optimization algorithm, as there seem not to be a safe universal choice for its parameters.

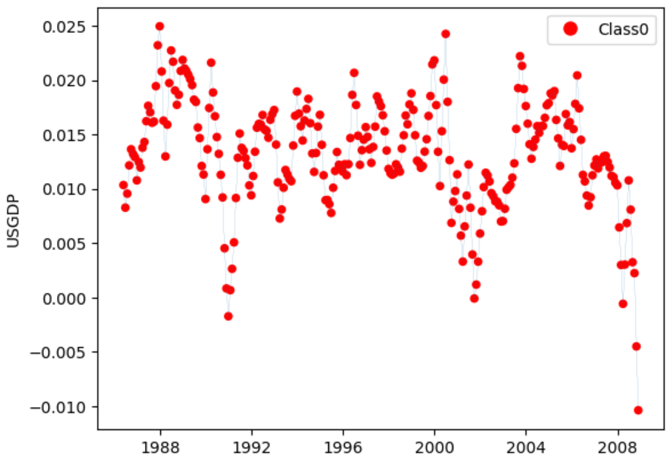


Figure 6.9: Most likely regimes for model trained with Adam with epochs=1000, learning rate=0.0001 and number of batches = 50

We take this as a proof of the instability of using Adam for the maximisation step with noisy data, and decide to perform no further experiment with richer datasets.

We then moved on to the Nelder Mead optimization algorithm. Given the previous results we only performed experiments with the non-starting algorithm. We fit again the model in the time period until end of 2008, and now we get a much more interesting classification for the regime of the training point.

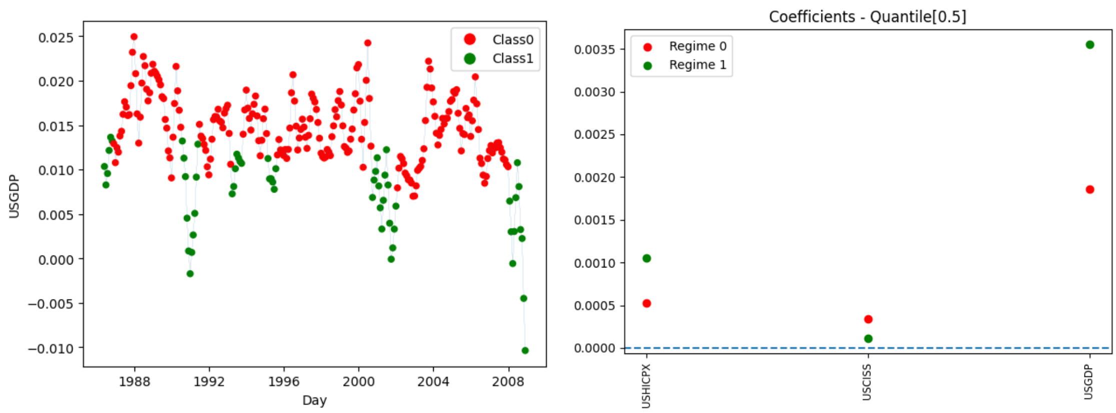


Figure 6.10: Results for the simplified framework with NM optimization, no restart

As observed in the linear programming optimization, here too we observe that all the financial crisis time periods are classified differently from other points, suggesting a possibly meaningful, although noisy segmentation of the points.

However when looking at the coefficients we are again dissatisfied. This plot not only is in contrast with what we got in the linear regression case, but also returns a result that is in contrast with the economical sense of the variables, suggesting positive influence of inflation to economic growth.

We then performed one final experiment allowing the linear coefficient to be parametrized through the Almon polynomials.

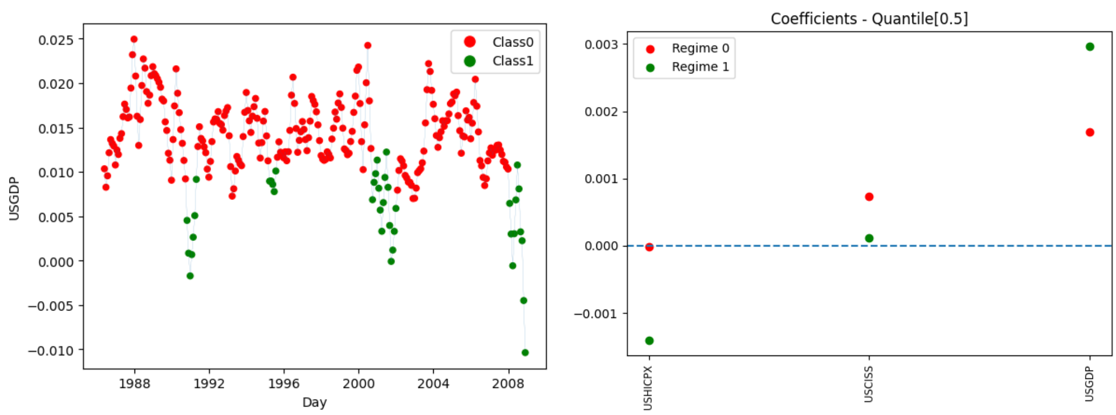


Figure 6.11: Results for the simplified framework with NM optimization, no restart

As we can see in this framework we achieve the very same result of linear regression, however we are not sure on the reason why parametrizing in such a way may have affected improved the performance of our model. We tried to perform the experiments with the Almon polynomial framework with the Adam optimizer too, however in this framework the results were again uninteresting.

6.3.2 Full time period experiment

Given the positive results presented above, we fit a model with the same time split as we did in the linear case for the full US data, but allowing for some variables to be sampled more frequently than monthly, using Nelder-Mead algorithm without restart and parametrization through the Almon polynomials.

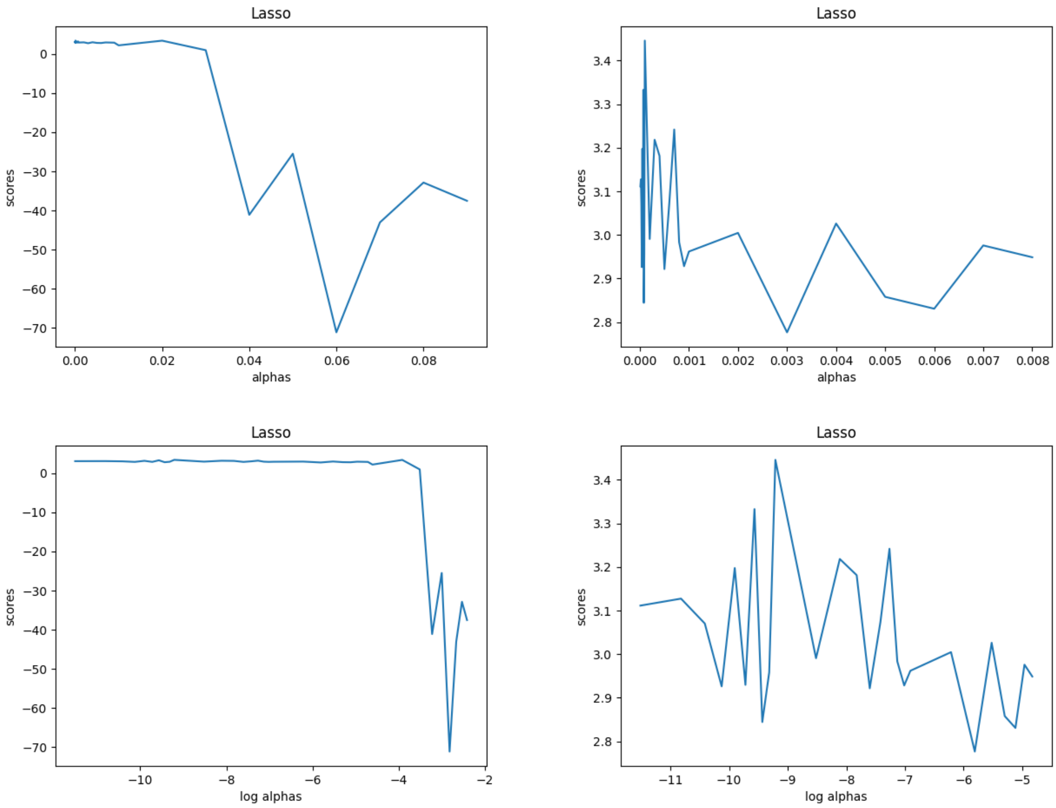


Figure 6.12: normalized likelihood score on the validation set for different α of lasso regularization

We notice the results were again very noisy at the beginning, probably due to overfitting, the best value was provided for $\alpha = 0.0001$, so we use this value to train the model on the time period until 2018. We observe again how the regime identification matches with our economical intuition:

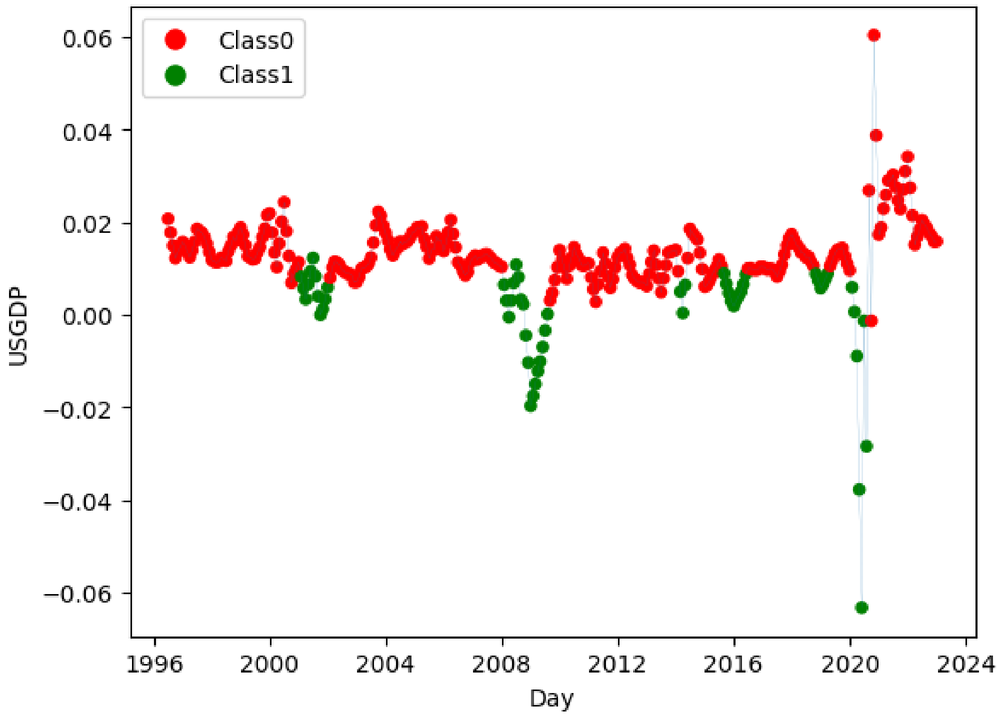


Figure 6.13: regime detected by the model given all data until 2023

score of train set	score of test set	uc test	jdq test	DQ test
4.279	3.894	0.03	0.25	0.07

Table 6.3: normalized likelihoods and p-values for the tests

We observe that contrary to the linear case, the use of multifrequency data provided with less ideal results for the scores, as the normalized likelihood on the test set appears to be considerably lower than the one in the training set. Also the statistical tests provided less conclusive results when compared to our previous experiments.

Again, we can use the model for interpretation of the role played by each variable in the prediction of the median of future GDP:

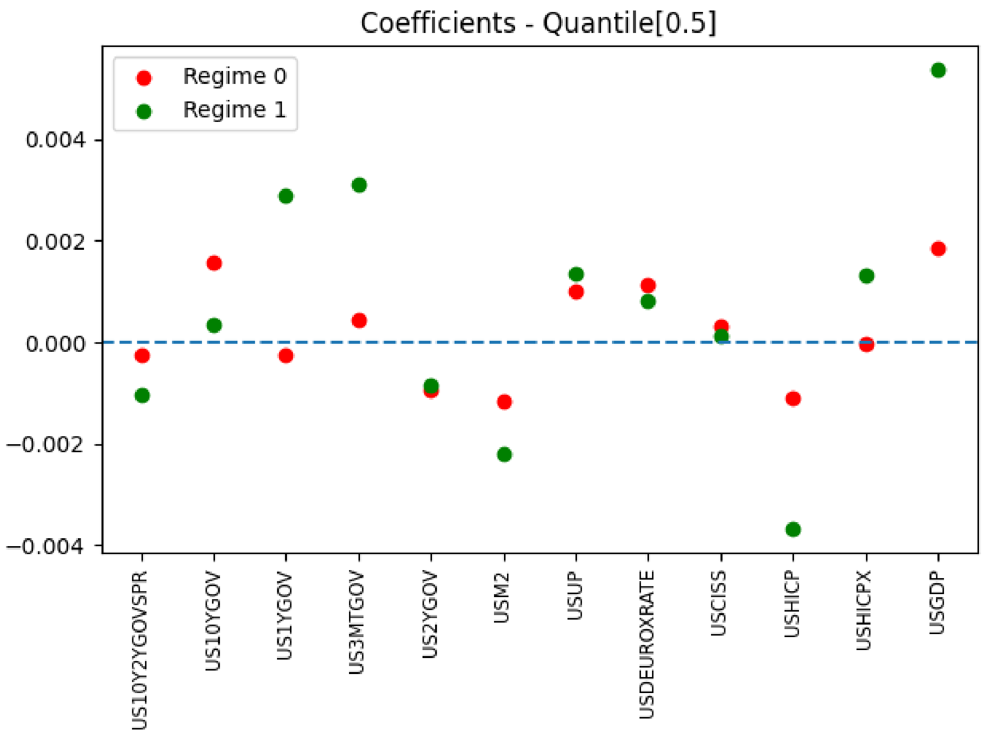


Figure 6.14: Coefficients of quantile regression according to regime

Due to our choice of the parameter of lasso more variables seem to affect the output. We observe that generally all the variable either provide the same correlation with respect to the output in both regime, or are just relevant in a single regime. We also notice how it appears that in different regimes the multifrequency data seems to put more weight in present or past information according to the regime. Due to the number of variables that are kept in the model, we find it harder to interpret the coefficients, but, once again, we observe the same general relationships that bind GDP growth with its past lags and inflation (USHICP), as well as money supply (given by M2) and the different treasury rates.

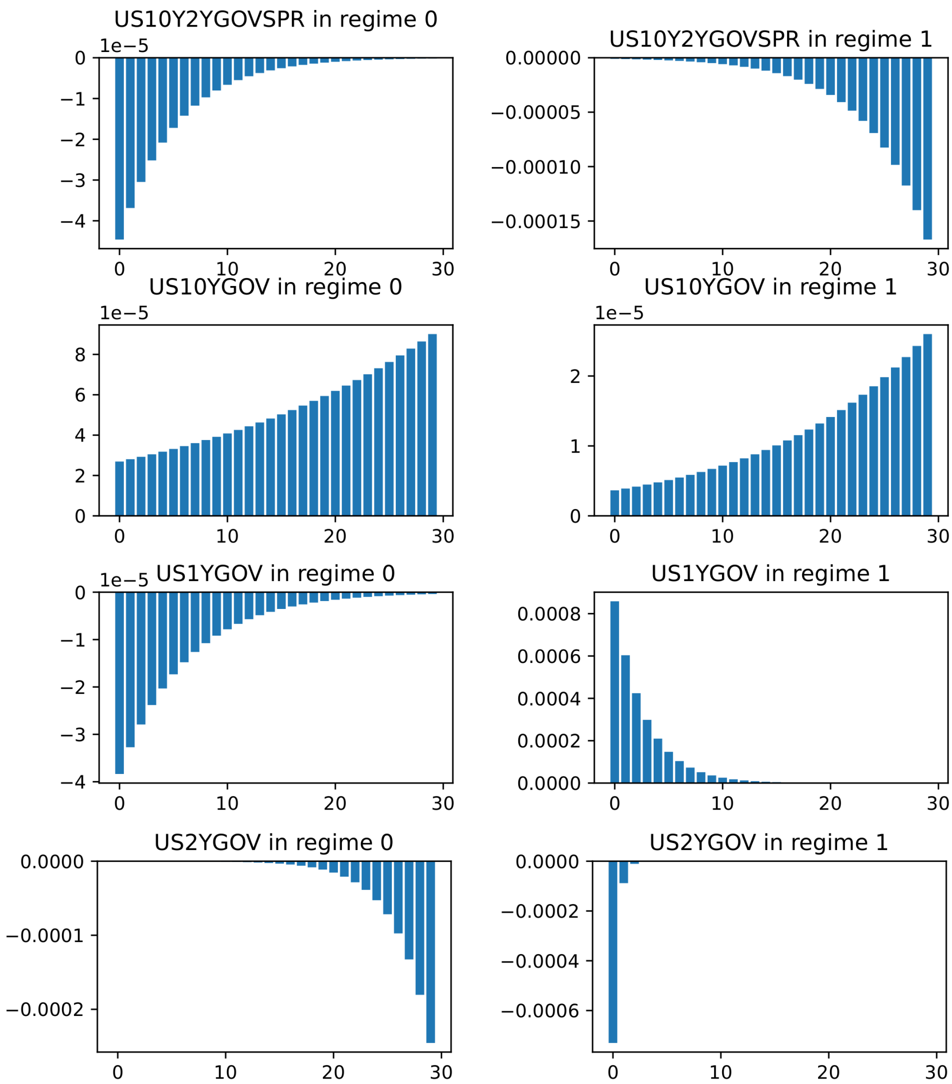


Figure 6.15: Coefficients of quantile regression according to regime

6.4 Experiments on tests performance

Given the two issues that we mentioned in the introduction to chapter 5, we performed some simulation to visualize both the influence of the quality of the parameters estimated and the power of our tests.

6.4.1 Test results on artificial data

We first fix some notation: from now on $\{\epsilon_t\}_{t=0}^T$ will be a sequence of iid Normal variables with mean 0 and variance 1 with the property of being also independent from any future variable that we will introduce. We also define $x_{t}{}_{t=1}^T$ to be a sequence of random variables with no particular property. During the tests we will generate these variables as iid normal random variables. In order to conduct our tests, we considered 4 kind of artificial datasets:

- the first one is defined by the following equation:

$$y_t = x_t \beta^\intercal + \epsilon_t \tag{6.4.1}$$

we observe that after generating our data we get that the ys are again a sequence of independent variables

- the second one is similar to the first one but is autoregressive:

$$y_t = x_t \beta_x^\intercal + y_{t-1} \beta_y^\intercal + \epsilon_t \tag{6.4.2}$$

- for the third model we added a regime switching component to the first model. We consider the process $\{S_t\}_{t=0}^T$ defined as a discrete Markov chain with two states, 0 and 1, and transition matrix:

$$M = \begin{bmatrix} 0.95 & 0.05 \\ 0.15 & 0.85 \end{bmatrix} \tag{6.4.3}$$

we consider two arrays of real coefficients β_0 and β_1, and the dynamic of y_t, dependent on S_t will be defined in the following manner:

$$y_t = x_t \beta_{S_t}^\intercal + \epsilon_t \tag{6.4.4}$$

- the final model will add the Markov regime component to the autoregressive model. With S_t defined as above the equation defining the dynamic will be:

$$y_t = x_t \beta_{x,S_t}^\intercal + y_{t-1} \beta_{y,S_t}^\intercal + \epsilon_t \tag{6.4.5}$$

From each of this statistical models we generated a dataset and performed estimation of parameters in a fitting manner performing quantile regression. For the first two models we performed simple linear quantile regression. For the last two we applied the Markov switching linear quantile regression framework. For prediction in the Markov switching linear quantile regression framework, at each time step we assumed knowledge of all the past and predicted the probability of being in each regime at the final past time. Then we applied the transition matrix to the probability vector and picked the most probable state as a result. We then performed prediciton using the quantile linear regression framework using the coefficients of the estimated regime. We remark that this means that for future time steps we assume knowledge of all the history until the predicted time.

The results were confronted with a model whose parameters were the actual parameters generating the sequence, and one whose parameters were chosen to be significantly different from the true ones. Here is a table showing our choices:

	true model	mispecified model	fitted model
purely independent data	const=1, x=2	const=1, x=0.5	const=0.9657 x=2.0461
purely autoregressive data	x=2, y=0.5	x=3, y=0.9	const=0.0063 x=2.0545 y=0.5100
Markov switching independent data	x1=[2,1], x2=[-3,0.5]	x1=[1,2], x2=[-1,0.1]	const1=-0.557, x1=[2.077,1.174], const2=0.081, x2=[-2.733,0.369]
Markov switching autoregressive data	[x1=2,y1=0.9], [x2=-3,y2=0.5]	[x1=1,y1=2], [x2=-1,y2=0.1]	[const1=0.016,x1=-2.848,y1=0.437], [const2=-0.029, x2=1.963,y2=0.867]

Here we have taken just one example of fitted parameters, due to the fact that they didn't show significant changes with different seeds.

We then performed the unconditional coverage and joint dynamic quantile, and the out-of-sample dynamic quantile tests and verified the tests performance. The tests were performed both with fitted models and with true parameters, in order to better understand the nature of the success or failure of the tests. The filtration that we chose for the DQ test is composed by the last k lags of the true return variable, where k is the number of covariates used for the regression, thus either 2 or 3. We also tried using data starting from the present time to invalidate the hypothesis of the outDQ test on the filtration and verify that it fails. Furthermore we performed our tests adding a column of constants to the filtration matrix in order to complement the information that it provides.

The "nan" entries represent a combination of model/test that wasn't performed due to lack of interesting information retrievable from its results.

The datasets were generated to have a sample size of 300 datapoints, similar to the real world dataset that we have access to, while the tests were conducted on three different test sets, the first one being of size 20000, the second one being of size 40, and the last one being of size 200. With such test sets we expect to capture the nature of the quantile tests that we conducted, since we recall that theorem 5.3.1 depend on the following condition on the size of the train and test set :

$$\frac{N_{test}}{N_{train}} \to 0$$

when

$$N_{test} \to \infty \qquad \text{and} \qquad N_{train} \to \infty$$

At the same time for the smaller datasets we expect our test to exhibit low power.

During unreported tests we observed that there was little to no difference when performing predictions using true true or estimated matrices or any other variables, (the predicted states were essentially the same for the true and the estimated model) so we didn't perform further analysis on those parameters.

The first results that we present are the ones of the 20000 datapoints test set:

	uc 20000	jdq 20000	outDQ correct 20000	outDQ correct+const 20000	outDQ present 20000	outDQ present+const 20000
purely independent data, true model	0.138	0.327	0.453	0.272	0.000	0.000
purely independent data, fit model	0.002	0.004	0.000	nan	nan	nan
purely independent data, mispecified model	0.000	0.000	nan	0.000	nan	nan
purely autoregressive data, true model	0.157	0.162	0.091	0.078	0.000	0.000
purely autoregressive data, fit model	0.033	0.102	0.041	nan	nan	nan
purely autoregressive data, mispecified model	0.054	0.000	nan	0.000	nan	nan
Markov switching independent data, true model	0.406	0.463	0.030	0.065	0.000	0.000
Markov switching independent data, fit model	0.000	0.000	0.000	nan	nan	nan
Markov switching independent data, mispecified model	0.000	0.000	nan	0.000	nan	nan
Markov switching autoregressive data, true model	0.406	0.424	0.250	0.325	0.000	0.000
Markov switching autoregressive data, fit model	0.000	0.000	0.000	nan	nan	nan
Markov switching autoregressive data, mispecified model	0.505	0.000	nan	0.000	nan	nan

Table 6.4: Results for tests , sample size=20000, seed=60

The first thing that we observe is that with these many points, the very misspecified model that we built, generally fails the tests. However even during other simulations it seems that with this setup the UC test, might fail to reject even by the model with purposely wrong parameters. We also notice, as expected from the theory, that for dimensions of the test sample that far exceeds the one of the train sample, these tests clearly reject the fitted model in most cases. Finally we note that as expected the model with the true parameters generally performs well and better than the other instances. However, we observe that the p-values seems to possibly get quite low, making intepretation of the results of the tests more uncertain. Another observation is that, although the results for the DQ tests with and without an extra column of constants differ, there seems to be no clear preference or way to distinguish the performance of the two spefications. On the other hand the violation of the hypothesis of the test, in the form of inclusion in the filtration of present data, seems to consistently make the test fail. On a side note, this can also be interpreted as the use of past quantile estimates for future data produces undesirable results. No difference is detected if a column of costants is added to the filtration matrix of such test.

We then show the results obtained from a test set of 40 datapoints:

	uc 40	jdq 40	outDQ correct 40	outDQ present 40
purely independent data, true model	1.000	0.048	0.967	0.007
purely independent data, fit model	0.773	0.445	0.765	0.000
purely independent data, mispecified model	0.001	0.000	0.004	nan
purely autoregressive data, true model	0.664	0.386	0.581	0.162
purely autoregressive data, fit model	0.885	0.422	0.479	0.048
purely autoregressive data, mispecified model	0.039	0.000	0.000	nan
Markov switching independent data, true model	0.777	0.287	0.687	0.005
Markov switching independent data, fit model	0.569	0.320	0.271	0.001
Markov switching independent data, mispecified model	0.235	0.309	0.031	nan
Markov switching autoregressive data, true model	1.000	0.973	0.999	0.179
Markov switching autoregressive data, fit model	0.757	0.887	0.940	0.216
Markov switching autoregressive data, mispecified model	0.768	0.586	0.419	nan

Table 6.5: Results for the uc and jdq tests , sample size=40, seed=50

As we can see with a smaller sample our results become highly more uncertain. Not only we notice that in Markov switching models we seem not to be able to distinguish a reasonable model from an unrelated one, but we may also seem to have our tests work when we violate the hypothesis of the test. We observe however that as expected, the performance of the fitted model in such regime is always recognised, even at cases when the test really manage to distinguish the true model vs a fake one.

Adding more data seem to however to improve the results.

	uc 200	jdq 200	outDQ correct 200	outDQ present 200
purely independent data, true model	0.593	0.219	0.016	0.000
purely independent data, fit model	0.356	0.060	0.000	nan
purely independent data, mispecified model	0.000	0.000	0.000	nan
purely autoregressive data, true model	0.942	0.705	0.895	0.000
purely autoregressive data, fit model	0.113	0.105	0.055	0.000
purely autoregressive data, mispecified model	0.314	0.157	0.000	nan
Markov switching independent data, true model	0.645	0.054	0.016	0.000
Markov switching independent data, fit model	0.548	0.206	0.008	0.000
Markov switching independent data, mispecified model	0.000	0.000	0.000	nan
Markov switching autoregressive data, true model	0.176	0.167	0.277	0.000
Markov switching autoregressive data, fit model	0.891	0.680	0.505	0.000
Markov switching autoregressive data, mispecified model	0.122	0.001	0.000	nan

Table 6.6: Results for the uc and jdq tests , sample size=200, seed=50

6.4.2 Markov chain test

In order to better understand the poor performance of our tests against some very invalid models, we perform a final test, using the jdq framework. These tests consist in generating a stream of data, where the variables have the same theoretical distributions of the Hit_t variables, that is they are iid bernoulli, to which we subtracted their mean. Then the statistics of the jdq test is computed using the theoretical variance, and then confronted against the normal distribution.

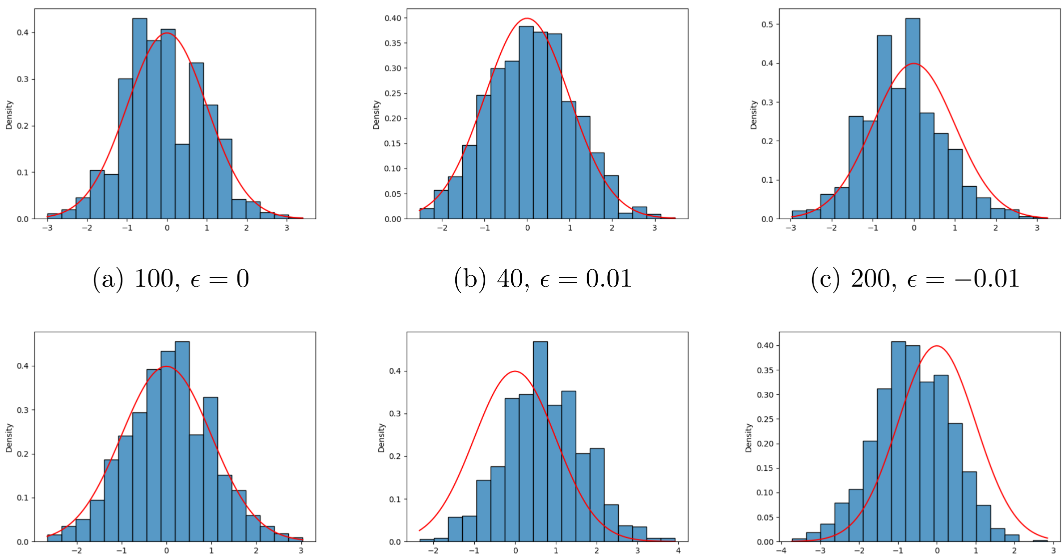
Then we repeated the same test, but instead of generating data according to the distribution of the Hits, we perturbed their probability, by building a Markov chain, that has different probabilities of landing in one of the values of Hit_T, that depends on the current state in the form of:

$$\begin{bmatrix} \tau + \epsilon & 1 - \tau - \epsilon \\ \tau - \epsilon & 1 - \tau + e\epsilon \end{bmatrix} \tag{6.4.6}$$

where ϵ can be positive or negative. Then the same computation as before is performed, using again the theoretical variance according to the previous probability. Finally we performed these with multiple samples of different size, $40, 100, 200, 1000$, each time generating 1000 statistics, and their results confronted. In all our tests τ was 0.5.

Our tests found, as expected a situation of greater uncertainty in the case of samples generated by less data.

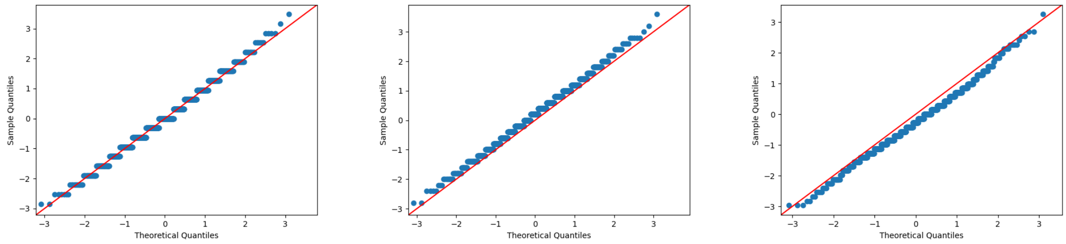
As we can observe for 40 points, the distribution of the statistics can be easily confused, and this issue seems to be mitigated while the statistics are computed with enough data.



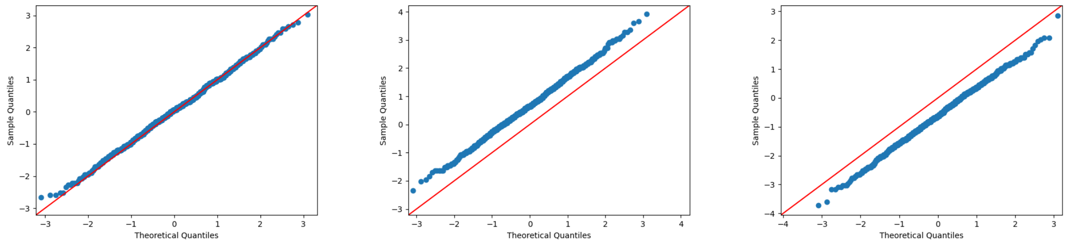
(a) 100, $\epsilon = 0$ (b) 40, $\epsilon = 0.01$ (c) 200, $\epsilon = -0.01$

(d) 1000, $\epsilon = 0$ (e) 1000, $\epsilon = 0.01$ (f) 1000, $\epsilon = -0.01$

Figure 6.16: Barplots of statistics computed from simulations confronted with the standard Gaussian distribution



(a) 40, $\epsilon = 0$ (b) 100, $\epsilon = 0.01$ (c) 200, $\epsilon = -0.01$



(d) 40, $\epsilon = 0$ (e) 1000, $\epsilon = 0.01$ (f) 1000, $\epsilon = -0.01$

Figure 6.17: QQ plots of statistics computed from simulations confronted with the standard Gaussian distribution

Finally our conclusion is that although we got some encouraging results from our models, lack of data prevents us to really assess the performance of our model.

Chapter 7

Extended theoretical model

In this section we present an extended theoretical model that allows both much more freedom for the choice of the distribution of the data, and considers autoregressive behaviour, other than a regime switching dynamic.

7.1 Quantile autoregression

We first introduce quantile autoregression. Let $\{U_t\}$ be a sequence of iid standard uniform random variables, and consider the autoregressive process:

$$y_t = \psi_0\left(U_t\right) + \psi_1\left(U_t\right) y_{t-1} \tag{7.1.1}$$

where the ψ_j 's are unknown functions $[0,1] \rightarrow \mathbb{R}$ that we will want to estimate.

Provided that the right hand side of 7.1.1 is monotone increasing in U_t, it follows that the τ-th conditional quantile function of y_t can be written as,

$$Q_{y_t}\left(\tau \mid y_{t-1}\right) = \psi_0(\tau) + \psi_1(\tau) y_{t-1}$$

or more compactly as,

$$Q_{y_t}\left(\tau \mid \mathcal{F}_{t-1}\right) = x_t^\top \psi(\tau)$$

where $x_t = \left(1, y_{t-1}\right)^\top$, and $\mathcal{F}_t$ is the σ-field generated by $\{y_s, s \leq t\}$. In the above model, the autoregressive coefficients may be τ-dependent and thus can vary over the quantiles. We will refer to this model as the QAR(1) model.

This model extends some already known models. For example let $\Phi^{-1}(\tau)$ be the Gaussian cumulative distribution function and let's consider our model with $\theta_0(\tau) = \sigma\Phi^{-1}(\tau)$, and $\theta_1(\tau) = \theta_1$ is a constant in τ. This model is now the standard AR(1) model.

Monotonicity of the conditional quantile functions imposes some discipline on the forms taken by the θ functions. This discipline essentially requires that the function $Q_{y_t}\left(\tau \mid y_{t-1}, \ldots, y_{t-p}\right)$ is monotone in τ in some relevant region Υ of $\left(y_{t-1}, \ldots, y_{t-p}\right)$-space. The correspondence between the random coefficient formulation of the QAR model 7.1.1 and the conditional quantile function formulation presupposes the monotonicity of the latter in τ. We note that while model 7.1.1 can, even in the absence of this monotonicity, be taken as a valid data generating mechanism, in that case the link to the strictly linear conditional quantile model is no longer valid.

7.2 Markov Switching autoregressive models

Following the work of [17], we specialize their definition in simpler framework. Let $(Y_t, S_t)_{t=0}^{\infty}$ be a discrete-time stochastic process such that, for each $t \in \mathbb{N}$, $S_t \in \mathbb{S} \equiv \{s_1, \ldots, s_{|\mathbb{S}|}\} \subset \mathbb{R}$ is the unobservable state and $Y_t \in \mathbb{Y} \subseteq \mathbb{R}^h$, for some $h \in \mathbb{N}$, is the observable state. Moreover, for each $t \in \mathbb{N}$, the conditional distribution of Y_t, given Y_0^{t-1} and S_0^t, depends only on S_t, and the conditional distribution of S_t, given Y_0^{t-1} and S_0^{t-1}, depends only on Y_{t-1} and S_{t-1}, so that

$$\begin{aligned} Y_t \mid \left(Y_0^{t-1}, S_0^t\right) &\sim P_*\left(Y_{t-1}, S_t, \cdot\right) \\ S_t \mid \left(Y_0^{t-1}, S_0^{t-1}\right) &\sim Q_*\left(Y_{t-1} \cdot\right) \end{aligned}$$

with $(y, s) \mapsto P_*(y, s, \cdot) \in \mathcal{P}(\mathbb{Y})$ and $(s) \mapsto Q_*(s, \cdot) \in \mathcal{P}(\mathbb{S})$ denoting the true transition probabilities. It is further assumed that, for each $(y, s) \in \mathbb{Y} \times \mathbb{S}$, $P_*(y, s, \cdot)$ admits a density $p_*(y, s, \cdot)$ with respect to some σ-finite measure on $\mathbb{Y}$.

The researcher's model is given by a family of transition probabilities $(y, s) \mapsto P_\theta(y, s, \cdot) \in \mathcal{P}(\mathbb{Y})$ and $(y, s) \mapsto Q_\theta(s, \cdot) \in \mathcal{P}(\mathbb{S})$ indexed by an (unknown) parameter $\theta \in \Theta \subseteq \mathbb{R}^q$, for some $q \in \mathbb{N}$, such that, for each $\theta \in \Theta$,

$$\begin{aligned} Y_t \mid \left(Y_0^{t-1}, S_0^t\right) &\sim P_\theta\left(Y_{t-1}, S_t, \cdot\right) \\ S_t \mid \left(S_0^{t-1}\right) &\sim Q_\theta\left(S_{t-1}, \cdot\right) \end{aligned}$$

and, for each $(y, s) \in \mathbb{Y} \times \mathbb{S}$, $P_\theta(y, s, \cdot)$ admits a density $p_\theta(y, s, \cdot)$ with respect to the same measure used to define $p_*(y, s, \cdot)$.

This framework is now similar to the one of hidden Markov models, but the addition of the dependence of the future lag of the return variable on the present lag variable, makes the model indeed autoregressive.

Let $\bar{P}_*^\nu$ denote the true distribution over $(Y_t)_{t=0}^{\infty}$ when the distribution of (Y_0, S_0) is ν. Let's consider for any $T \in \mathbb{N}$, the function $\ell_T^\nu : \mathbb{Y}^{T+1} \times \Theta \to \mathbb{R}$ defined as:

$$\ell_T^\nu\left(Y_0^T, \theta\right) = T^{-1} \sum_{t=1}^{T} \log p_t^\nu\left(Y_t \mid Y_0^{t-1}, \theta\right)$$

where $p_t^\nu\left(Y_t \mid Y_0^{t-1}, \theta\right)$ denotes the conditional density of Y_t given Y_0^{t-1} for any $\theta \in \Theta$; the latter is then defined as follows: for any $t \geq 1$,

$$p_t^\nu\left(Y_t \mid Y_0^{t-1}, \theta\right) = \sum_{s' \in \mathbb{S}} \sum_{s \in \mathbb{S}} p_\theta\left(Y_{t-1}, s', Y_t\right) Q_\theta\left(s, s'\right) \delta_t^{\theta,\nu}(s)$$

and $s \mapsto \delta_t^{\theta,\nu}(s) \equiv \bar{P}_\theta^\nu\left(S_{t-1} = s \mid Y_0^{t-1}\right)$. For each $t \geq 2$ and any $s \in \mathbb{S}$, $s \mapsto \delta_t^{\theta,\nu}(s)$ satisfies the recursion

$$\delta_t^{\theta,\nu}(s)=\sum_{\tilde{s}\in\mathbb{S}}\frac{Q_\theta\left(\tilde{s},s\right)p_\theta\left(Y_{t-2},\tilde{s},Y_{t-1}\right)\delta_{t-1}^{\theta,\nu}(\tilde{s})}{\sum_{s'\in\mathbb{S}}p_\theta\left(Y_{t-2},s',Y_{t-1}\right)\delta_{t-1}^{\theta,\nu}\left(s'\right)}$$

with $s\mapsto\delta_1^{\theta,\nu}(s)=\sum_{\tilde{s}\in\mathbb{S}}Q_\theta\left(\tilde{s},s\right)\nu\left(\tilde{s}\mid Y_0\right)$, where $\nu(\cdot\mid\cdot)$ is the conditional density corresponding to ν.

For a given initial distribution $\nu\in\mathcal{P}(\mathbb{Y}\times\mathbb{S})$ over (Y_0,S_0), we define our estimator as $\hat{\theta}_{\nu,T}$, where

$$\ell_T^\nu\left(Y_0^T,\hat{\theta}_{\nu,T}\right)\geq\sup_{\theta\in\Theta}\ell_T^\nu\left(Y_0^T,\theta\right)-\eta_T$$

for some $\eta_T\geq0$ and $\eta_T=o(1)$.

We now consider the situation where ν is a Borel probability measure on $\mathbb{Y}\times\mathbb{S}$, for which Y_0^∞ is stationary and ergodic. Under this hypothesis then the process Y_0^∞, can be extended to a two-sided sequence $Y_{-\infty}^\infty$. This hypothesis will be a consequence of our list of assumptions.

Let $H^*:\Theta\to\mathbb{R}_+\cup\{\infty\}$ be the Kullback-Leibler information criterion $\theta\mapsto H^*(\theta)$, which is given by

$$H^*(\theta)=E_{\bar{P}_*^\nu}\left[\log\frac{p_*^\nu\left(Y_0\mid Y_{-\infty}^{-1}\right)}{p^\nu\left(Y_0\mid Y_{-\infty}^{-1},\theta\right)}\right]$$

where, for any $\theta\in\Theta, p^\nu\left(Y_t\mid Y_{-\infty}^{t-1},\theta\right)$ is defined as $\liminf_{M\to\infty}p_\theta^\nu\left(Y_t\mid Y_{-M}^{t-1}\right)$; $p_*^\nu\left(Y_t\mid Y_{-\infty}^{t-1}\right)$ is defined analogously. Under the assumptions stated in the theorems $p^\nu\left(Y_0\mid Y_{-\infty}^{-1},\theta\right)$ will also correspond to the conditional density of Y_0 given $Y_{-\infty}^{-1}$ induced by (P_θ,Q_θ,ν), and $p_*^\nu\left(Y_0\mid Y_{-\infty}^{-1}\right)$ will be its counterpart induced by the true transition kernels (P_*,Q_*,ν).

We allow for misspecified models, and thus $p_*^\nu\notin\{p^\nu(\cdot\mid\cdot,\theta):\theta\in\Theta\}$ and the relevant limiting set for our estimator is

$$\Theta_*=\underset{\theta\in\Theta}{\arg\min}H^*(\theta),$$

which is the pseudo-true parameter (set) that minimizes the Kullback-Leibler information criterion.

We now present the results of consistency as presented in [17] as Theorem 1.

Theorem 7.2.1. . *Suppose the following assumptions:*

1) There exists a constant $q>0$ such that, for all $Q\in\{Q_\theta:\theta\in\Theta\}\cup Q_*,Q\left(s,s'\right)\geq q$ for all $(s',s)\in\mathbb{S}^2$.

2) There exist constants $\lambda'\in(0,1),\gamma\in(0,1),b'>0$ and $R>2b'/(1-\gamma)$, a lower semi-continuous function $\mathcal{U}:\mathbb{Y}\to[1,\infty)$, and a measure $\varpi\in\mathcal{P}(\mathbb{Y})$ such that, for all $s\in\mathbb{S}$: (i) $\int_{\mathbb{Y}}\mathcal{U}\left(y'\right)P_*\left(y,s,dy'\right)\leq\gamma\mathcal{U}(y)+b'1\{y\in A\}$, with $A\equiv\{y\in\mathbb{Y}:\mathcal{U}(y)\leq R\}$; (ii) A is bounded and $\varpi(A)>0$; (iii) $\inf_{y\in A}P_*(y,s,C)\geq\lambda'\varpi(C)$ for any Borel set $C\subseteq\mathbb{Y}$.

3) (i) Θ is compact; (ii) H^* exists and is lower semi-continuous.

4) For any $\delta > 0$ and any $\dot{\theta} \in \Theta$, let $B(\delta, \dot{\theta}) \equiv \{\theta \in \Theta : \|\dot{\theta} - \theta\| < \delta\}$.

(i) For any $\epsilon > 0$, there exists some $\delta > 0$ such that

$$\max_{\dot{\theta}\in\Theta} E_{\bar{P}^{\nu}_{*}} \left[\sup_{\theta \in B(\delta,\dot{\theta})} \frac{p^{\nu}\left(Y_0 \mid Y^{-1}_{-\infty}, \theta\right)}{p^{\nu}\left(Y_0 \mid Y^{-1}_{-\infty}, \dot{\theta}\right)} \right] \leq 1 + \epsilon$$

(ii) there exists a function $(y, y') \mapsto C(y, y') \in \mathbb{R}_{+}$ such that $\sup_{\theta\in\Theta} \frac{\max_{s\in\mathbb{S}} p_{\theta}(Y,s,Y')}{\min_{s\in\mathbb{S}} p_{\theta}(Y,s,Y')} \leq C(Y, Y')$ and $\frac{\max_{s\in\mathbb{S}} p_{*}(Y,s,Y')}{\min_{s\in\mathbb{S}} p_{*}(Y,s,Y')} \leq C(Y, Y')$ a.s. $-\bar{P}^{\nu}_{*}$.

Then,

$$d_{\Theta}\left(\hat{\theta}_{\nu,T}, \Theta_{*}\right) = o_{\bar{P}^{\nu}_{*}}(1)$$

where, for any set $A \subseteq \Theta, d_{\Theta}(\theta, A) \equiv \inf_{\dot{\theta}\in A} \|\theta - \dot{\theta}\|$

.

The theorem can be reformulated with a set of slightly different conditions, by following the results already present in [17], in particular Lemma 12.

Theorem 7.2.2. *Suppose that assumptions 1) and 4)(ii) hold. Assume further:*

2') there exists a $\nu \in \mathcal{P}(\mathbb{Y} \times \mathbb{S})$ such that, under $\bar{P}^{\nu}_{*}, (Y_t)^{\infty}_{t=0}$ is stationary and ergodic.

5) $T^{-1} \sum_{t=1}^{T} \max\{1, C(Y_{t-1}, Y_t)\} \left(1 - \underline{q}\right)^{t} = o_{\bar{P}^{*}_{\nu}}(1)$.

Suppose finally that Θ is compact and that for each $n \in \mathbb{N}_0, \theta \mapsto p^{\nu}_{\theta}\left(Y_1 \mid Y^{0}_{-n}\right)$ is uniformly continuous a.s. $-\bar{P}^{\nu}_{*}$. Suppose also that there exists functions $(y_1, y_0) \mapsto \left(\bar{p}(y_0, y_1), \underline{p}(y_0, y_1)\right)$ such that for any $p \in \{p_{\theta} : \theta \in \Theta\}$,

$$\underline{p}(y_0, y_1) \leq p(y_0, s, y_1) \leq \bar{p}(y_0, y_1) \;\; \textit{for all } s \in \mathbb{S}$$

, and

$$E_{\bar{P}^{\nu}_{*}}\left[\bar{p}(Y_0, Y_1) / \underline{p}(Y_0, Y_1)\right] < \infty \qquad E_{\bar{P}^{\nu}_{*}}\left[p_{*}(Y_0, Y_1) / \underline{p}(Y_0, Y_1)\right] < \infty$$

.

Then,

$$d_{\Theta}\left(\hat{\theta}_{\nu,T}, \Theta_{*}\right) = o_{\bar{P}^{\nu}_{*}}(1)$$

We note that we asked for slightly weaker conditions then the ones proposed in Lemma 12. However the proof itself doesn't use those conditions in their full form and our assumptions can be used in the same fashion to prove the lemma.

7.3 Markov switching QAR

We consider a model where $\mathbb{S} = \{0, 1\}$, and

$$y_t = \psi_0^{S_t}(U_t) + \psi_1^{S_t}(U_t)\, y_{t-1} \tag{7.3.1}$$

Now the transition from Y_{t-1} to Y_t, knowing that $S_t = s$ have now density with respect to the Lebesgue measure given by

$$p_(y', s, y) = \left(\frac{\partial}{\partial y}\Psi^{-1}_{s,y'}(y)\right) = \frac{1}{\Psi'_{s,y'}(\Psi^{-1}_{s,y'}(y))} \tag{7.3.2}$$

where $\Psi_{s,y'}(u) = \psi^s_0(u) + \psi^s_1(u)y'$, where we further assumed that $\Psi_{s,y'}(u)$ is strictly increasing, or $\Psi'_{s,y'}(u) > 0$.

That is we are assuming a dynamic where the transitions dependent from the hidden state may assume two different forms of quantile autoregressive process.

Let $\Theta \subset (0,1)^2 \times \mathbb{R}^q \times \mathbb{R}^q$, for $q \in \mathbb{N}$ and let $\theta = (p_0, p_1, \theta_0, \theta_1) \in \Theta$. We consider p_1, p_2 as the parameters of the two state hidden Markov chain, with transition matrix:

$$\begin{bmatrix} p_0 & 1-p_0 \\ 1-p_1 & p_1 \end{bmatrix} \tag{7.3.3}$$

while θ_s are the parameters for the quantile regression coefficients assuming we are at regime s. For any $T \in \mathbb{N}$, let $\mathcal{L}^\nu_T : \mathbb{Y}^{T+1} \times \Theta \to \mathbb{R}$ be the sample criterion function given by

$$\mathcal{L}^{\tau,\nu}_T \left(Y^T_0, \theta\right) = T^{-1}\sum_{t=1}^{T} log\left(L^{\tau,\nu}_t \left(Y_t \mid Y^{t-1}_0, \theta\right)\right)$$

where $L^{\tau,\nu}_t$ is a function defined as:

$$L^{\tau,\nu}_t \left(Y_t \mid Y^{t-1}_0, \theta\right) = \sum_{s\in\mathbb{S}} exp(-\rho_\tau \left(Y_t - \theta_{1,s}Y_{t-1} - \theta_{0,s}\right) \delta^{\theta,\nu}_t(s)$$

for $(\theta_{1,s}, \theta_{0,s}) = \theta_s, s = \{0,1\}$, ρ_τ the quantile loss function and $\delta^{\theta,\nu}_t(s)$ defined as in the previous section, with $p_\theta(Y_{m-1}, s, Y_m) = exp(-\rho_\tau\left(Y_m - \beta^1_s(\theta)Y_{m-1} - \beta^0_s(\theta)\right)$ for any $m \in \mathbb{N}$,

Again, for a given initial distribution $\kappa \in \mathcal{P}(\mathbb{Y} \times \mathbb{S})$ over (Y_0, S_0), we define our estimator as $\hat{\theta}_{\kappa,T}$, where

$$\mathcal{L}^{\tau,\kappa}_T \left(Y^T_0, \hat{\theta}^\tau_{\kappa,T}\right) \geq \sup_{\theta\in\Theta} \mathcal{L}^{\tau,\kappa}_T \left(Y^T_0, \theta\right) - \eta_T$$

for some $\eta_T \geq 0$ and $\eta_T = o(1)$.

We now wish to provide some convergence result about $\hat{\theta}^\tau_{\kappa,T}$. In the following, we will prove that indeed $\hat{\theta}^\tau_{\kappa,T}$ converges to a point of the set Θ_*, defined as:

$$\Theta_* = \arg\max_{\theta\in\Theta} E_{\bar{P}^\nu_*}\left[\log L^{\tau,\nu}_t \left(Y_t \mid Y^{t-1}_{-\infty}, \theta\right)\right] \tag{7.3.4}$$

The key observation is that we can interpret the exponential of the quantile loss as a kernel, given by some scaled Laplace distribution. This interpretation is made possible by the fact that we can consider each term as a scaled probability density function, since we showed in chapter 2 that the scaling parameter depends only on τ. Thus we can think of our problem, as a misspecified problem of maximum likelihood for autoregressive hidden

Markov models, and under some reasonable assumptions we can rely on the work of [17] and prove that 7.2.2 holds in our framework. From now on we will use $p_\theta(Y_{t-1}, s, Y_t)$ and $exp\left(-\rho_\tau\left(Y_t - \theta_{1,s}Y_{t-1} - \theta_{0,s}\right)\right)$ interchangeably.

Our assumptions are the followings:

- Θ is compact
- Assumption *2')* holds with ν such that $\bar{P}_*^\nu[|Y_0| \geq M] \lesssim e^{-M^\gamma}$, for $\gamma \geq 0$, for every $M \geq \bar{M} > 0$.
- $\Psi'_{s,y'}(u) > k$, for some $k > 0$ and for every $s \in \{0, 1\}$ and $y' \in \mathbb{Y}$.
- $\Psi'_{s,y'}(\Psi^{-1}_{\theta;s,y'}(y)) \gtrsim e^{M\gamma'}$, for $\gamma' \geq 0$, for every y such that $|y| \geq \bar{M}(y') > 0$ and for every $s \in \{0, 1\}$ and $y' \in \mathbb{Y}$

We note that these hypothesis are verified, for example if we have a standard AR(1) model in each state.

Thus we only need to check that 1), 4)(ii) and *5)* hold, $\theta \mapsto p_\theta^\nu\left(Y_1 \mid Y_{-n}^0\right)$ is uniformly continuous a.s. $-\bar{P}_*^\nu$. and that there exists functions $(y_1, y_0) \mapsto \left(\bar{p}\left(y_0, y_1\right), \underline{p}\left(y_0, y_1\right)\right)$ such that for any $p \in \{p_\theta : \theta \in \Theta\} \cup p_*$

$$\underline{p}\left(y_0, y_1\right) \leq p\left(y_0, s, y_1\right) \leq \bar{p}\left(y_0, y_1\right)$$

for all $s \in \mathbb{S}$, and

$$E_{\bar{P}_*^\nu}\left[\bar{p}\left(Y_0, Y_1\right) / \underline{p}\left(Y_0, Y_1\right)\right] < \infty \qquad E_{\bar{P}_*^\nu}\left[p_*\left(Y_0, Y_1\right) / \underline{p}\left(Y_0, Y_1\right)\right] < \infty$$

.

Uniformly continuity of $p_\theta^\nu\left(Y_1 \mid Y_{-n}^0\right)$ is verified as consequence of $exp(-\rho_\tau\left(Y_t - \beta_s^1(\theta)Y_{t-1} - \beta_s^0(\theta)\right)$ being Lipshitz as a function of θ. 1) is direct consequence of the compactness of Θ.

We now define the functions $\bar{p}$ and $\underline{p}$. $\bar{p}$ is straightforward since p is limited by a constant that we will call C.

Let $diam(\Theta) = \sup\{d(\theta_1, \theta_2)|\theta_1, \theta_2 \in \Theta\}$. Then we can define, for some constant C':

$$\bar{p}(y_1, y_2) = e^{C'|diam(\Theta)|(|y_1|+|y_2|)}$$

Since we also have $\Psi'_{\theta;s,y'}(u) > k$, we only need to show:

$$\int_{\mathbb{R}}\int_{\mathbb{R}} e^{C'|diam(\Theta)|(|y_1|+|y_2|)} \frac{1}{\Psi'_{s,y_1}(\Psi^{-1}_{s,y_1}(y_2))} dy_2 \nu(dy_1) \leq \infty \tag{7.3.5}$$

which is then true due to our assumptions on $\Psi^{-1}_{s,y_1}(y_2))$ and ν.

4) (ii) is then satisfied by $Ce^{C'|diam(\Theta)|(|y_1|+|y_2|)}$.

Finally we prove *5)*.

$$\mathbb{P}\left(T^{-1}\sum_{t=1}^{T}\max\left\{1, C\left(Y_{t-1}, Y_t\right)\right\}(1-q)^t > \epsilon\right) =$$

$$\mathbb{P}\left(T^{-1}\sum_{t=1}^{T}\max\left\{1, exp\left(diam(\Theta)C\left(|Y_{t-1}| + |Y_t|\right)\right)\right\}(1-q)^t > \epsilon\right) =$$

We disintegrate with respect to the set $\{Y_i \geq M, i = 1, \ldots, T\}$, and exploit the stationarity of the process to derive the union bound

$$\mathbb{P}(\{Y_i \geq M, i = 1, \ldots, T\}) \leq T\mathbb{P}(Y_0 \geq M)$$

, thus getting the inequality:

$$\mathbb{P}\left(T^{-1}\sum_{t=1}^{T}\max\{1, C(Y_{t-1}, Y_t)\}(1-q)^t > \epsilon\right) \leq$$
$$\mathbb{P}\left(T^{-1}\sum_{t=1}^{T} exp\left(diam(\Theta)C(\tau)2M\right)(1-q)^t > \epsilon\right) + T\mathbb{P}(Y_0 \geq M) \leq$$
$$\mathbb{P}\left(T^{-1}C'' exp\left(M\right) > \epsilon\right) + T\mathbb{P}(Y_0 \geq M)$$

for some constant C''.

Then by choosing $M = log(T)^{\delta}$, for $1/\gamma < \delta < 1$, we get:

$$\lim_{T\to\infty} C''\mathbb{P}\left(T^{-1} exp\left(M\right)\frac{1}{1-q} > \epsilon\right) + T\mathbb{P}(Y_0 \geq M) = 0 \tag{7.3.6}$$

And the proof is completed by the arbitrariness of ϵ.

Conclusions

In this thesis we explored the Markov switching quantile regression model, from its theoretical formulation to an extensive analysis of computational issues and method of assessment. While the results seem promising when the variables are sampled all at the same frequency, we obtained a less clear picture in the case of multifrequency data. The assessment of our results also happened to be harder due to the proved unrealiability of the statistical tests caused by the lack of datapoints. We leave to future work the exploration of further and different methods for parametrizing the MIDAS polynomials. In this thesis we also concentrated on prediction of the median, but the method can be applied with any quantile and it would be interesting to verify the performance of the model in such framework. Finally we exploited the results of [17] to prove a convergence result for the estimator of the markov switching quantile autoregression model. The immediate next step would be to prove that indeed the points of convergence are the quantiles expressed by the coefficients of the QAR model.

Bibliography

[1] R. Koenker, *Quantile Regression* (Econometric Society Monographs). Cambridge University Press, 2005. DOI: 10.1017/CBO9780511754098 (cit. on p. 7).

[2] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2 (cit. on p. 7).

[3] O. Cappé, E. Moulines, and T. Rydén, "Inference in hidden markov models," in *Proceedings of EUSFLAT conference*, 2009, pp. 14–16 (cit. on p. 13).

[4] D. Barber, *Bayesian reasoning and machine learning*. Cambridge University Press, 2012 (cit. on p. 13).

[5] A. Maruotti, L. Petrella, and L. Sposito, "Hidden semi-markov-switching quantile regression for time series," *Computational Statistics & Data Analysis*, vol. 159, p. 107 208, 2021 (cit. on pp. 23, 28).

[6] R. Weiss, S. Du, J. Grobler, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Mueller, B. Thirion, D. Nouri, G. Louppe, J. Vanderplas, J. Benediktsson, L. Buitinck, M. Korobov, R. McGibbon, S. Lattarini, V. Niculae, csytracy, A. Gramfort, S. Lebedev, D. Huppenkothen, C. Farrow, A. Yanenko, A. Lee, M. Danielson, and A. Rockhill, *Hmmlearn*, version 0.3.0, Apr. 18, 2023 (cit. on pp. 24, 73).

[7] E. Ghysels, A. Sinko, and R. Valkanov, "Midas regressions: Further results and new directions," *Econometric reviews*, vol. 26, no. 1, pp. 53–90, 2007 (cit. on p. 36).

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014 (cit. on p. 38).

[9] F. Gao and L. Han, "Implementing the nelder-mead simplex algorithm with adaptive parameters," *Computational Optimization and Applications*, vol. 51, pp. 259–277, May 2012. DOI: 10.1007/s10589-010-9329-3 (cit. on p. 38).

[10] P. Team, *Pytorch*, version 1.11.0, Oct. 21, 2021 (cit. on pp. 40, 73).

[11] E. Lehmann and J. Romano, *Testing Statistical Hypotheses* (Springer Texts in Statistics). Springer International Publishing, 2022 (cit. on p. 61).

[12] W. K. Newey and K. D. West, "A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix," *Econometrica*, vol. 55, no. 3, pp. 703–708, 1987 (cit. on p. 63).

[13] R. F. Engle and S. Manganelli, "Caviar: Conditional autoregressive value at risk by regression quantiles," *Journal of business & economic statistics*, vol. 22, no. 4, pp. 367–381, 2004 (cit. on p. 65).

[14] W. Gaglianone, L. Lima, O. Linton, and D. Smith, "Evaluating value-at-risk models via quantile regressions," *Journal of Business and Economic Statistics*, vol. 29, May 2009. DOI: 10.1198/jbes.2010.07318 (cit. on p. 70).

[15] E. Bolthausen, "The berry-esseén theorem for strongly mixing harris recurrent markov chains," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 60, pp. 283–289, 1982 (cit. on p. 72).

[16] "Fred, federal reserve economic data." (), [Online]. Available: https://fred.stlouisfed.org/series (cit. on p. 75).

[17] D. Pouzo, Z. Psaradakis, and M. Solá, "Maximum likelihood estimation in markov regime-switching models with covariate-dependent transition probabilities," *Econometrica*, 2016 (cit. on pp. 96–98, 100, 103).

Appendix A

Dataset variables description

Code	**Variable Name**	**Economic Meaning**
BUBILL3M	Short-term German government bond yield	Interest rate on 3-month Bundesbank bills
CISS	composite indicator of systemic stress (EU area)	Measure of health and stability of the financial system
CRSPR	Corporate bond spread	difference between yields on corporate and government bonds (IG-3M Euribor)
EA10YGOV	Euro area bond yield (10 years)	Interest rate on 10-year benchmark bond
EA2YGOV	Euro area bond yield (2 years)	Interest rate on 2-year benchmark bond
EA3YGOV	Euro area bond yield (3 years)	Interest rate on 3-year benchmark bond
ESTER	Euro short-term rate	Volume-weighted trimmed mean rate
EURIBOR3M	Euro Area Euribor 3-month	Interest rate of 3-month Euribor
EUROMPB	Euro Area macroprudential buffer	value of Euro-denominated securities held by the Eurosystem for monetary policy purposes
HICP	Harmonized Index of Consumer Prices (EU area)	Measure of inflation for the Eurozone
HICPX	Harmonized Index of Consumer Prices excluding energy and unprocessed food (EU area)	Measure of inflation for the Eurozone
LHHNFC	loan volumes to Non-financial corporates (NFC) and household (HH).	The total amount of money borrowed by businesses and households from banks.
M1	Money supply (M1 definition)	Narrow measure of money supply, including physical currency and checking accounts
M3	Money supply (M3 definition)	Broad measure of money supply, including M1 and other liquid assets

OILPUSD	Oil price (USD denominated)	Price of crude oil in US dollars
OIS10Y	Overnight indexed swap rate (10-year)	Interest rate derived from swaps referencing future short-term rates over a 10-year period
OIS1Y	Overnight indexed swap rate (1-year)	Interest rate derived from swaps referencing future short-term rates over a 1-year period
OIS2Y	Overnight indexed swap rate (2-year)	Interest rate derived from swaps referencing future short-term rates over a 2-year period
SX5E	Euro Stoxx 50 Index	Stock market index for the 50 largest companies in the Eurozone
UP	Unemployement rate (EU area)	Labour market condition
US10YGOV	US government bond yield (10-year)	Interest rate on 10-year US government bonds
USDEUROXRATE	US dollar to Euro exchange rate	Exchange rate between the US dollar and the Euro
USGDP	US Gross Domestic Product	Total value of goods and services produced in the US
USHICP	US Harmonized Index of Consumer Prices	Measure of inflation
USHICPX	Harmonized Index of Consumer Prices excluding energy and unprocessed food (US area)	Measure of inflation for the Eurozone
USUP	Unemployement rate (US area)	Labour market condition
VSTOXX	VSTOXX Volatility Index	Measure of stock market volatility in Europe
DEIT10YSP	German-Italian spread (10-year)	Difference between interests rate on 10-year German and Italian government bonds
GPR	Geopolitical Risk Index	measure of adverse geopolitical events and the associated risks
GDP	Gross Domestic Product (EU area)	Total value of goods and services produced in EU
USCISS	composite indicator of systemic stress (EU area)	Measure of health and stability of the financial system
US1YGOV	US government bond yield (1-year)	Interest rate on 1-year US government bonds
US2YGOV	US government bond yield (2-year)	Interest rate on 2-year US government bonds
US3MTGOV	US government bond yield (3 months)	Interest rate on 3-months US government bonds
US10Y2YGOVSPR	US area spread between 10-year and 2-year government bond yield	Difference between the 10 year treasury rate and the 2 year treasury rate.
USM2	US area Money supply (M2 definition)	Middle measure of money supply, including M1, and strictly included in M3