Forecasting quantiles of cryptocurrency returns using MCMC algorithms

Mémoire

Carlos Alberto Chaparro Sepulveda

Sous la direction de:

Richard Luger, directeur de recherche

Résumé

> Une version interactive du présent document est disponible à l'adresse `https://cacsfre.gitlab.io/msc`.

Ce travail résume les étapes et les technologies nécessaires pour construire une application web dynamique permettant de faire l'analyse de données financières en temps réel à l'aide des langages de programmation R et C++. R est utilisé pour la collecte et traitement des données entrantes ainsi que pour générer tout output. C++ est utilisé pour accélérer les simulations Monte-Carlo. L'output de ce travail consiste en l'application web elle-même et les fonctions permettant d'estimer les paramètres des modèles de régression quantile de la famille `CAViaR`. Le code pour reproduire ce travail est organisé de la façon suivante :

- Un paquetage R pour l'application `shiny`, disponible à l'adresse `https://gitlab.com/cacsfre/simulr`.
- Un paquetage R pour estimer les paramètres des modèles de la famille `CAViaR`, disponible à l'adresse `https://gitlab.com/cacsfre/caviarma`.
- Le code R pour générer le présent document avec `bookdown`, disponible à l'adresse `https://gitlab.com/cacsfre/msc`.

La famille de modèles CAViaR a été utilisée pour obtenir une estimation du quantile q_α au niveau α. Ces modèles s'adressent directement au quantile d'intérêt au lieu de le calculer indirectement comme dans d'autres cas, par exemple les modèles de type GARCH où l'on s'intéresse plutôt à la volatilité σ^2. Les résultats obtenus ici sont comparables à ceux se trouvant dans la littérature tel qu'illustré dans les chapitres 3 et 4.

Abstract

An interactive version of this document is available at `https://cacsfre.gitlab.io/msc`.

This work summarizes the steps and technologies required to build a dynamic web application for the analysis of real time financial data using the R and C++ programming languages. R is used to collect and process the input data as well as to generate all output. C++ is used to speed up the Monte Carlo simulations. The output from this work consists of the web application itself and the functions used to estimate the parameters of the `CAViaR` family of quantile regression models. The code to reproduce this work is organized as follows :

- An R package with the `shiny` web application, available at `https://gitlab.com/cacsfre/simulr`.
- An R package to estimate the parameters of the `CAViaR` family of models, available at `https://gitlab.com/cacsfre/caviarma`.
- The R code used to generate this document with `bookdown`, available at `https://gitlab.com/cacsfre/msc`.

The `CAViaR` family of models has been used to find an estimate of quantile q_α at level α. These models target the quantiles of interest directly instead of using indirect calculations as it is the case with other popular models, such as GARCH-like models where the variable of interest is the volatility σ^2 instead of a specific quantile. The results obtained here are close to those in the literature as shown in chapters 3 and 4.

Contents

List of Tables

List of Figures

Introduction

Cryptocurrencies are digital currencies traded through privately owned exchanges. Retail and institutional investors are increasingly interested in this new asset class given its price growth in recent years as well as a means of portfolio diversification and risk management. Bitcoin (Nakamoto, 2008) plays a leading role in cryptocurrency exchanges and its price movements have an impact across the whole market despite the existence of multiple competing cryptocurrencies (Kyriazis, 2019). When adding a cryptocurrency such as Bitcoin to a portfolio, we need to be aware of its risk level. An important market risk metric of an investment Y is its Value at Risk at level α or VaR_α^Y as defined in equation (2.1). This metric represents what portion of a given investment is at risk given a probability level $\alpha \in (0,1)$. VaR forecasting can be done using non-parametric, parametric or semi-parametric statistical models.

Ardia et al. (2019b) found an improvement over the benchmark GARCH(1,1) model for one day ahead VaR forecasting using a Markov–switching GARCH specification, implemented in the R `MSGARCH` package (Ardia et al., 2020) while Wang et al. (2019) used a multivariate extension of the original CAViaR model (Engle and Manganelli, 2004) by White et al. (2015) to study the spillover effect from external markets and found that Bitcoin exhibits safe-haven like characteristics making it a good candidate for portfolio diversification. However, Bitcoins' high volatility makes it a high-risk investment which has important implications from a risk management perspective for any financial institution interested in being exposed to this asset class. For instance, capital requirements as per the Basel accords are higher for cryptocurrencies (Stavroyiannis, 2018).

The first chapter will introduce the mechanics of the blockchain technology upon which Bitcoin's value is built both from a technological and a from historical perspective. Next, we present the statistical models and methods used to obtain a forecast for VaR_{t+1} based on the information set $\mathcal{I}_t$. The last two chapters include a summary of the results obtained using different Markov Chain Monte Carlo (MCMC) estimation methods and applying those methods to a simulated dataset as well as real time data. These results have been summarized in the Simulation Study and Application chapters.

Different R packages are used for the Simulation study and Applications sections. These packages are text files containing source code written by their respective authors using mainly the

R (R Core Team, 2020) and C++ (Stroustrup, 2013) programming languages. All source code used to produce this document is freely available online at `https://cran.r-project.org/`, the source code of the `caviarma` package (Chaparro Sepulveda, 2019) and `simulr` package (Chaparro Sepulveda, 2021) is available at `https://gitlab.com/cacsfre`. An interactive web version of this document can be found at `https://cacsfre.gitlab.io/msc` while its source code can be downloaded from `https://gitlab.com/cacsfre/msc`.

Chapter 1

Cryptocurrencies

Societies dating back to the ancient Greeks archaic period (750 – 500 BC) have used metals such as silver and gold (Migeotte and Lloyd, 2009) to build coins, which are then used either to store or to exchange value. [1] Gold and silver are not money by nature, but money's nature requires something like gold and silver (Marx, 1890) because of their intrinsic value as rare metals (their *Substatzwert*) as well as for their ability to be mint and other potential industrial uses (their *Gebrauchswert*). [2] Gold is considered a safe-haven asset in modern financial markets (Baur and McDermott, 2010) since it is expected to keep or increase its value in volatile markets. Figure 1.1 shows the normalized prices [3] of the following assets since 2020-01-02:

- Gold `GC=F` (USD).
- Silver `SI=F` (USD).
- CBOE Volatility Index `^VIX` (USD).
- USD `CAD=X` (CAD).
- JPY `JPYCAD=X` (CAD).
- Bitcoin `BTC-CAD` (CAD).

In this chapter we introduce the main building piece behind Bitcoin's success: the *blockchain* and its proof-of-work consensus mechanism which represents the computer *labour* required to produce a *Bitcoin*, its digital *Substanzwert*. The usefulness of a Bitcoin from a trading perspective, its *Gebrauchswert*, depends exclusively on its acceptance as a payment method in exchange for digital and physical goods and services. This acceptance has been growing since Bitcoin's inception as evidenced by:

[1] There are also several industrial applications for these metals.
[2] Storing value in bullions vs. coins mint for day-to-day transactions.
[3] Each price series starts from 1.

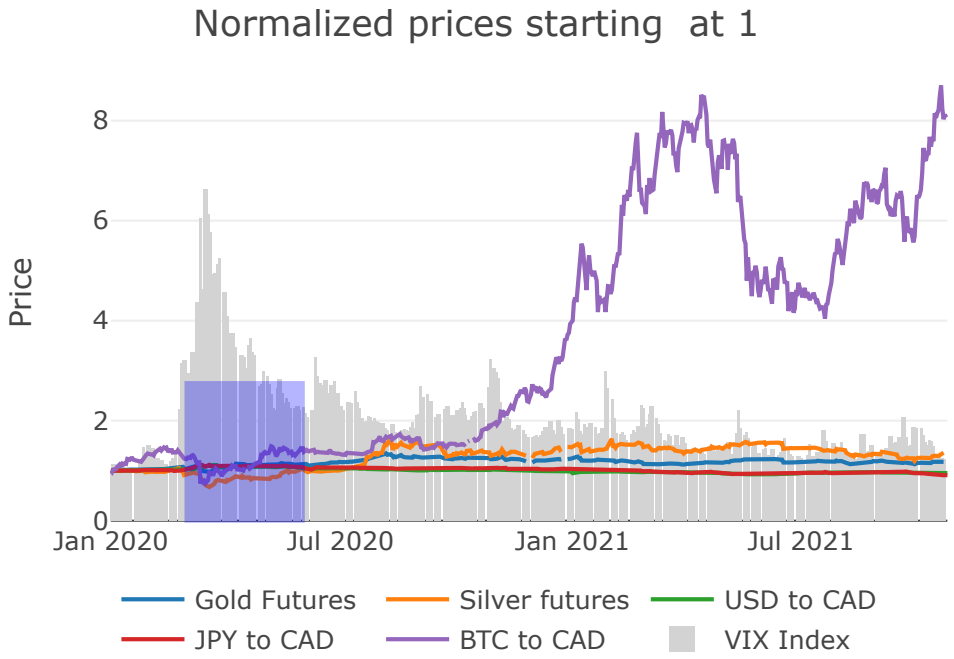

Figure 1.1: Gold, Silver (futures) and Bitcoin (spot) prices vs. VIX.

- its growing popularity in countries facing rapid inflation and currency restrictions (Bloomberg (2017) and Bloomberg (2021a)).
- its growing acceptance as a means of payment in electronic transactions (Bloomberg (2021c), PayPal (2021) and Bloomberg (2020-10-21)).

Cryptocurrencies constitute a largely unregulated asset class aiming to compete with traditional currencies and metals such as gold and silver as both a digital reserve currency and a safe-haven asset. Because of this, the most challenging risk faced by cryptocurrencies such as Bitcoin come from regulation, which could potentially render illegal holding or trading such crypto-assets. There exists evidence of such behaviour from world powers dating back to the Greek archaic period (Migeotte and Lloyd, 2009) whose political and economic power is linked to the use of their currency to settle transactions and to collect taxes. On the other hand, growing investors' concerns about the environmental impact of cryptocurrency mining hardware are opening the door for new and more sustainable blockchain-based cryptocurrencies.

1.1 Bitcoin

The Bitcoin cryptocurrency was introduced by Nakamoto (2008) as a mechanism to make electronic payments between two parties without the need of a trusted third party. A transaction between two parties using Bitcoin takes place in the same way as a typical physical transaction involving cash, i.e., a good or service is exchanged for a predetermined amount of coins and the transaction is non-reversible since the parties identities are never required: the

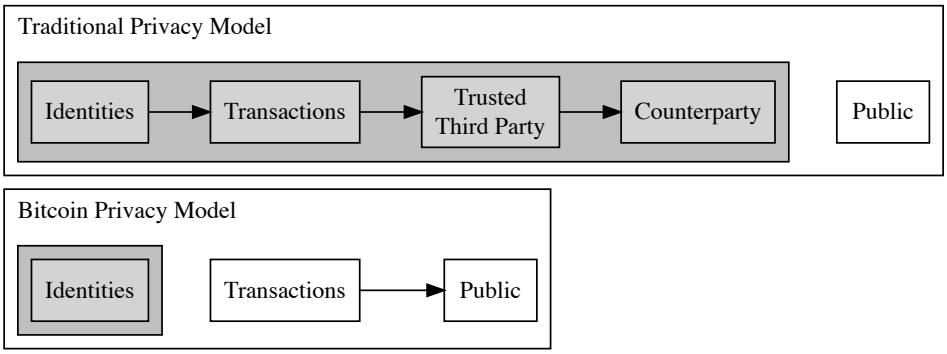

Figure 1.2: Traditional vs. new privacy model.

value is stored in the coins. Since no intermediary is involved in a cryptocurrency transaction, there is no central authority capable of modifying past records.

The need to reverse transactions arise from the inherent uncertainty of doing business over an electronic communications channel since one of the parties can easily default on its obligations, e.g., the delivery of a good or service. However, transaction costs increase with the need of a dispute mediation mechanism and non-reversible electronic transactions are not possible since a third party can potentially mediate disputes. Physical transactions using cash require the buyer and seller to be present for the transaction to take place and no third party is involved: cash transactions are also non-reversible. [4]

The goal of the Bitcoin technology is to serve as cash for digital transactions, hence avoiding the need of a third party while accepting the same condition one accepts when using cash, i.e., that the exchange is non-reversible. When using a traditional coin or a Bitcoin, the coin itself serves two purposes: *storing* and *exchanging* value. To attain this goal, a new privacy model is proposed by Nakamoto (2008) as shown in Figure 1.2. This new model requires all Bitcoin transactions to be public, which means that anybody can known its time and amount. However, the transactions are not linked to any identities. This level of privacy is comparable to a level II quote from a stock exchange, where volume and prices in the book are made public, but not the identities of the investors.

1.2 Blockchain

A crucial piece of technology behind Bitcoin is its public distributed ledger: the *Blockchain*. Bitcoin transactions data is first timestamped and digitally signed, then written into a *block*. This blocks of transaction records are linked to each other through a unique *Hash* obtained using SHA-256 and creating a chronologically ordered chain of transaction records or blockchain. This public ledger includes the history of all transactions since the beginning of (Bitcoin)

[4] A cash transaction does not require the identities to be disclosed.

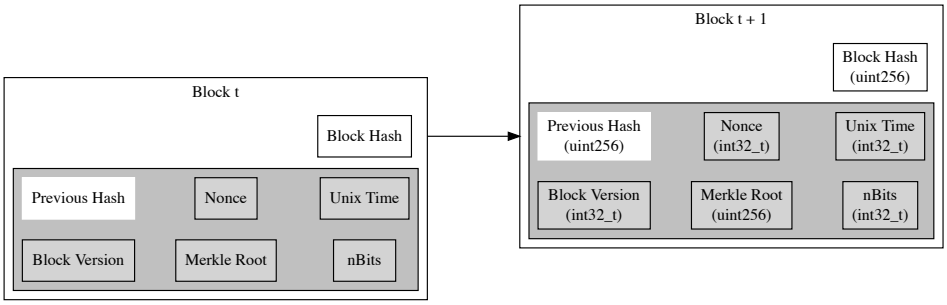

Figure 1.3: A block and a chain of blocks using Bitcoin's data types.

time and is equivalent to a bank's private ledger, where a bank would keep track of currency ownership. Now, imagine that the banks ledger itself, let's say an Excel spreadsheet, is the currency itself. In that hypothetical case, the Excel file would be equivalent to the *blockchain*: a database keeping track of money ownership and movements. Also, if one were to create such an Excel-based blockchain, the file sheets or tabs representing the individual blocks should be chronologically ordered, with several transactions per sheet as well as the corresponding cryptographic nonce (*number once*), previous hash and current hash as shown in Figure 1.3 as per the current Bitcoin specification. The source code is available at `https://github.com/bitcoin/bitcoin/blob/master/src/primitives/block.h`.

Each transaction is timestamped and digitally signed by the two parties and once a block has been created a process called *Mining* begins during which the hash must be computed taking as input the data from the block. The input data is hashed using a Merkle Tree (Merkle, 1980) and only the root of the tree is used when computing the block hash to save disk space.

The blockchain serves hence two purposes: it is the technology where transactions take place and it is its own database. Creating a new block, a process called *Mining*, requires finding a hash of all the data contained in the block while satisfying additional constraints. This mining step can take minutes to hours depending on the size of the block and the central processing unit (CPU) power available to find the hash. This computing *work* requires electricity and CPU to find a hash, which is then fed into the next block. Modifying the original data in any previous block requires redoing the work to find a new hash for that block and for any future blocks. Network nodes will by design always consider the longest chain to be the *true* chain and will keep working to find future hashes, which means that the Bitcoin system is designed to always follow the chain requiring the greatest computing power to be produced.

1.3 Proof of work

From a security perspective, the system is vulnerable by design to an attacker capable of controlling more than half of the available CPU in the network since the attacker could build a parallel longer chain that would eventually be followed by the honest nodes as shown in

Figure 1.5. The underlying logic to tolerate this risk is explained by Nakamoto (2008) as follows: someone capable of controlling more than half of the network's computing resources must decide whether to use their resources to maintain or to destroy their own wealth. It is important to mention that other cryptocurrencies and blockchain technologies use alternatives to the proof-of-work consensus algorithm such as Ethereum's proof-of-stake system (Buterin, 2013), where the weight of a node is proportional to its currency holdings instead of its computing power.

The proof-of-work system works by restricting the hash of the block to a fixed number of leading zeros which requires some time to be found since it must be randomly guessed. The time that it takes to find a valid block hash depends on the block's difficulty, which is controlled through the `nBits` target number. The SHA-256 hash of an empty string `""` obtained with the `openssl` package (Ooms, 2021) is:

```
openssl::sha256(paste0(data="", block="", nonce=""))
```

```
## [1] "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855"
```

which would not be a valid Bitcoin block hash since its leading characters are not zeros. However, the hash corresponding to `"1134816"` is

```
openssl::sha256(paste0(data="", block="1", nonce="134816"))
```

```
## [1] "000001f8479faf79c1a58152ffc6b027a93f6ae6b27dc19ef986b2c9e7cad3b3"
```

Which has 5 leading zeros. A current Bitcoin's hash needs 19 leading zeros to be considered valid and accepted by the network. The average median confirmation time from `2020-04-06` to `2021-04-04` was `12.01` minutes.

Since it takes *time* or computer *work* to find a valid block hash, the longer chain is the one which needed the most CPU power to be produced, i.e., to find valid block hashes. The R loop below shows a simple example of the type of work necessary to find a block hash:

```
difficulty <- 5
max_nonce <- 1e6
for(i in 1:max_nonce) {
  hash_i <- openssl::sha256(paste0(block_data="", block_number=1, nonce=i))
  if(substr(hash_i, 1, difficulty) == strrep(0, difficulty)) {
    break
```

```
  }
}
hash_i
```

Since every block includes the hash of the previous block as shown in Figure 1.3, modifying any data in a past block of the chain will render the remaining blocks invalid and a new mining process will be required to fix all the hashes that have been invalidated by the data tampering. This means that unless an attacker is capable of producing a longer chain faster than all honest nodes in the system by controlling more than half of the system's computing power, the network is capable of ensuring its own data integrity.

To link two blocks, we need to add an additional `prev_block_hash` data field to our previous example:

```
a_block <- paste0(
  prev_block_hash = character(),
  block_data = character(),  # Merkle root hash
  block_number = integer(),
  nonce = integer()
)
```

And then mine the block to find its hash. Once a block hash has been found, it becomes the input to the next block. The example below creates a blockchain with three blocks containing nothing but an empty string as `block_data` field. The first block's hash is deliberately left empty in this example.

```
n_blocks <- 3
max_nonce <- 1e6
difficulty <- 2
block_chain <- list("")
for(block_i in 2:(n_blocks)) {
  this_block <- paste0(
    prev_block_hash = block_chain[[block_i - 1]],
    block_data = "",
    block_number = block_i,
    nonce = 0
  )
  for(i in 1:max_nonce) {
    hash_i <- openssl::sha256(
```

Table 1.1: A sample blockchain with 3 blocks

	Hash
Block 1	
Block 2	00328ce57bbc14b33bd6695bc8eb32cdf2fb5f3a7d89ec14a42825e15d39df60
Block 3	00d7b5192f725a525822b57f5563514d0c4d4ff9bc4fb0ce84347b31f6fa577a

Table 1.2: Time (in milliseconds) required to mine increasingly difficult chains

expr	min	median	max
One Block, Level 1	496	535	2222
One Block, Level 2	3864	4001	5465
One Block, Level 3	51475	54744	77432
Two Blocks, Level 1	714	823	1018
Two Blocks, Level 2	5702	6978	9866
Two Blocks, Level 3	548195	584731	633797
Four Blocks, Level 1	2284	2410	2952
Four Blocks, Level 2	12716	15566	19171
Four Blocks, Level 3	1176702	1382572	1524348

```
      paste0(
        prev_block_hash = block_chain[[block_i - 1]],
        block_data = "",
        block_number = block_i,
        nonce = i
      )
    )
    if(substr(hash_i, 1, difficulty) == strrep(0, difficulty)) {
      break
    }
  }
  block_chain[[block_i]] <- hash_i
}
names(block_chain) <- paste0("block_", 1:n_blocks)
```

Table 1.2 shows benchmark times obtained using the `microbenchmkark` package (Mersmann, 2019) while mining blockchains of increasing difficulty (Level) and length (Blocks). We can see in Figure 1.4 that the time needed to mine 4 blocks increases exponentially with the difficulty, which is controlled with the number of leading zeros required by the hash to be considered valid.

We can also calculate the probability of successfully compromising the network as a function

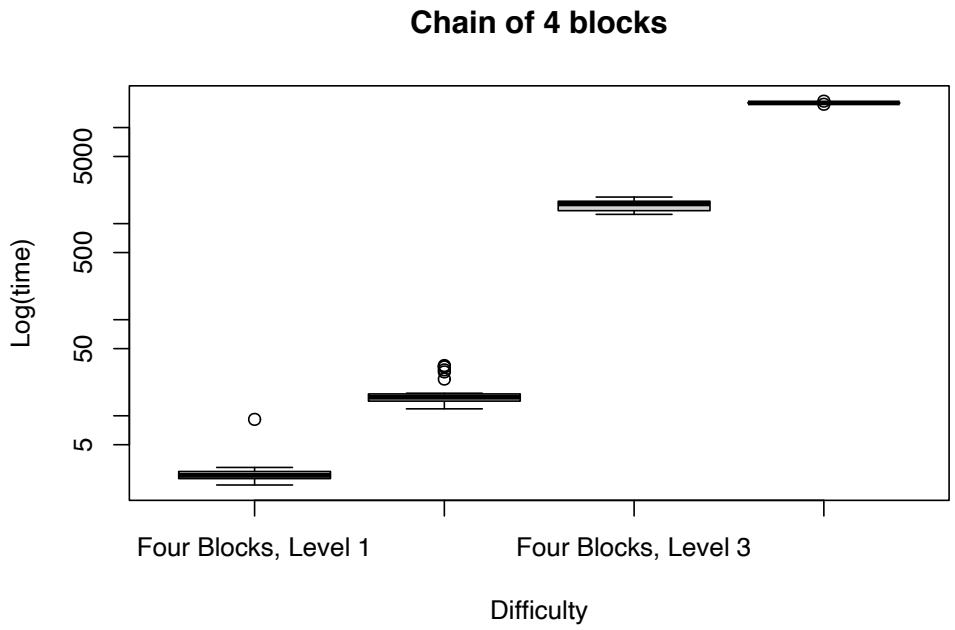

Figure 1.4: Time vs. difficulty, 4 blocks.

of an attacker's probability of finding the next block in the chain vs. an honest node finding it. We can compute the probability q_z that an attacker will catch up from z blocks behind following Nakamoto (2008) as follows:

$$q_z = 1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left\{ 1 - \left(\frac{q}{p} \right)^{(z-k)} \right\} \tag{1.1}$$

Where,

p = probability an honest node finds the next block.

q = probability the attacker finds the next block.

q_z = probability the attacker will ever catch up from z blocks behind.

Which can be translated into the following R code: [5]

```
attacker_success_probability <- function(q_prob, z_blocks) {
  p_prob <- 1 - q_prob
  lambda <- z_blocks * (q_prob / p_prob)
  sum_prob <- 1
  for(k in 0:z_blocks) {
```

[5] This is nothing but an R version of the original C code in the paper by Nakamoto (2008).

Table 1.3: The probability of success drops exponentially with z.

	q = 0.05	q = 0.1	q = 0.2	q = 0.35
0	1.00e+00	1.00e+00	1.00e+00	1.000000
5	2.87e-05	9.14e-04	2.74e-02	0.336365
10	1.25e-09	1.24e-06	1.07e-03	0.142805
15	5.60e-14	1.74e-09	4.29e-05	0.062454
20	-5.16e-17	2.46e-12	1.74e-06	0.027659
25	9.26e-17	3.30e-15	7.13e-08	0.012334
30	9.12e-17	-8.69e-17	2.93e-09	0.005524
35	-1.02e-17	2.24e-17	1.21e-10	0.002481
40	3.85e-17	-4.84e-17	4.97e-12	0.001117
45	-7.36e-17	-6.46e-17	2.05e-13	0.000504
50	2.66e-17	-4.30e-17	8.60e-15	0.000227

```
    poisson_prob <- exp(-lambda)
    i <- 1
    while(i <= k) {
      poisson_prob <- poisson_prob * lambda / i
      i <- i + 1
    }
    sum_prob <- sum_prob - poisson_prob * (1 - (q_prob / p_prob)^(z_blocks - k))
  }
  return(sum_prob)
}
```

The setup used in Figure 1.4 corresponds to `z_blocks = 4`, and the attacker's success probability can be found through Equation (1.1) as summarized in Table 1.3 .

1.4 Market value

In this section we have introduced the mechanics behind the Blockchain technology without mentioning prices nor quantiles so far. Several applications based on the Blockchain technology are currently in development/production (Bloomberg, 2021d) as evidence of Blockchain's *technological* or *intrinsic* value. However they do not necessarily have a relationship with Bitcoin nor with its price. Therefore, we might wonder whether there is *any* value in a cryptocurrency at all.

Linking computer resources to a *token* (the cryptocurrency) through a proof-of-work algorithm consumes electricity, which is mainly produced from coal and gas with only 28% of global consumption coming from renewable sources (Agency, 2020). Growing investors' concerns

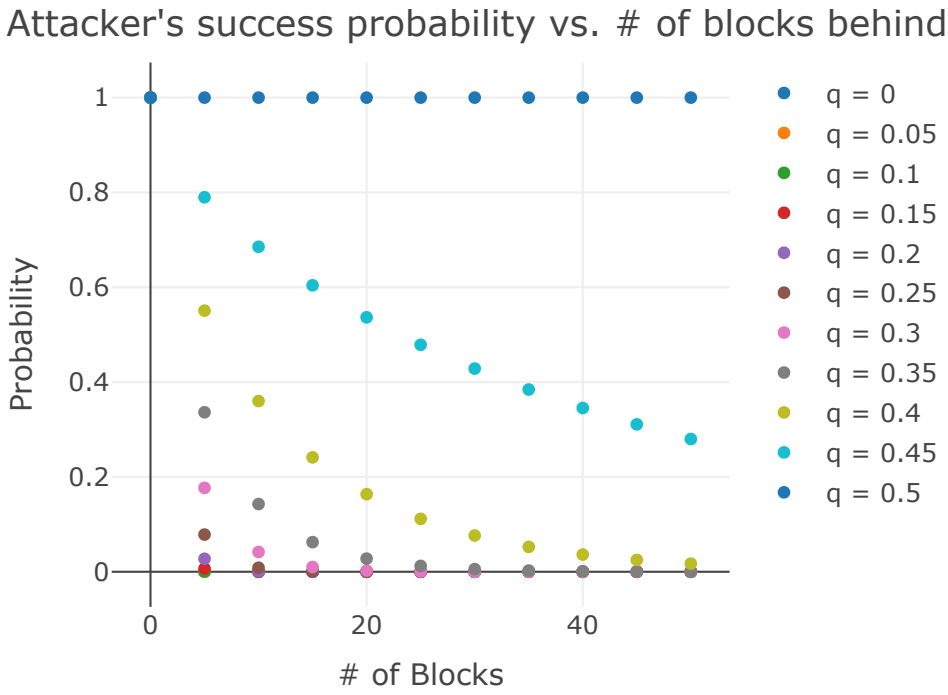

Figure 1.5: Attacker's success drops exponentially with the number of blocks

about the environmental impact of Bitcoin mining is one of the reasons behind alternative mining algorithms since a financial transaction's environmental impact is directly linked to the computing power required to process such transactions through the network. As a consequence of such innovations and with over 6000 cryptocurrencies available worldwide (Statista, 2021), investors' capital allocation to this new asset class is increasingly diversifying out of the original Blockchain implementation, i.e., Bitcoin.

We commonly use an asset's closing price as a proxy of the asset's (market) value, and Bitcoin's market price remains far from zero since its inception despite the legal and technological challenges (Bloomberg, 2021b). The goal of the following section is to test different MCMC methods and VaR models to forecast quantiles of the distribution of returns of a cryptocurrency such as Bitcoin. For this, we will apply standard econometric methods to a vector of cryptocurrency returns as we aim to get a forecast of VaR, which represents a quantile of the distribution of financial returns.

Chapter 2

Forecasting Methods

2.1 Quantile Models

When forecasting quantiles of a random variable Y, we must pay close attention to the assumptions behind the models being used. For instance, a standard *Value at Risk* (VaR) model used to estimate a quantile q_α at level α relies on the assumption of stationary and normally distributed asset returns, which tends to underestimate the probability of large losses. This can be especially problematic for volatile cryptocurrency returns which exhibit both *volatility clustering* and *fat tails* as evidenced in Figures 2.1 and 2.2. [1] In this chapter we present some statistical models and estimation methods that can be used to estimate VaR_α^Y, which we will often denote as VaR_α for simplicity. In the following section, we will compare the results obtained using common VaR forecasting models such as *GARCH* models to those obtained using quantile regression models of the CAViaR family proposed by Engle and Manganelli (2004). Here $\boldsymbol{y} = (y_1, y_2, \ldots, y_t)$ denotes a `numeric` vector of log-returns and we follow these definitions following Ruppert and Matteson (2011):

$$\text{VaR}(\alpha) = \inf\{x : \Pr(\mathcal{L} > x) \leq \alpha\} \tag{2.1}$$

and,

$$\text{ES}(\alpha) = \frac{\int_0^\alpha \text{VaR}(u)du}{\alpha} \tag{2.2}$$

where $\mathcal{L}$ in Equation (2.1) is the *loss* over a period T and $\text{VaR}(\alpha)$ denotes the *Value at Risk* quantile at level α or VaR_α. This value corresponds to a quantile of the return vector $\boldsymbol{y}$. Equation (2.2) defines the *expected shortfall* or average loss in case y_t exceeds VaR_α. We now discuss different methods used for forecasting q_α.

[1] We could also use histograms and quantile-quantile plots to better visualize this discrepancy.

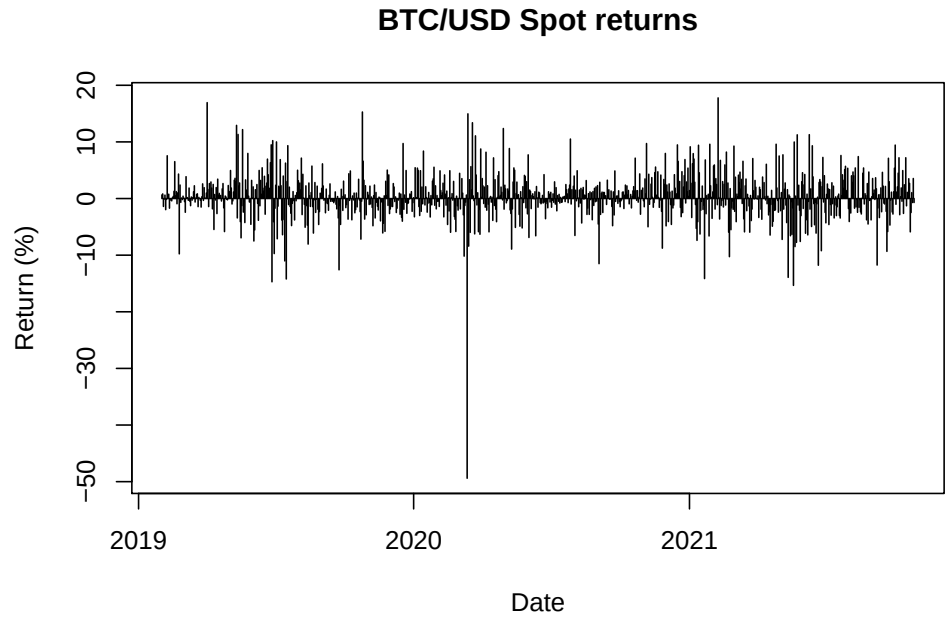

Figure 2.1: Historical returns (BTCUSD), prices from Yahoo! finance.

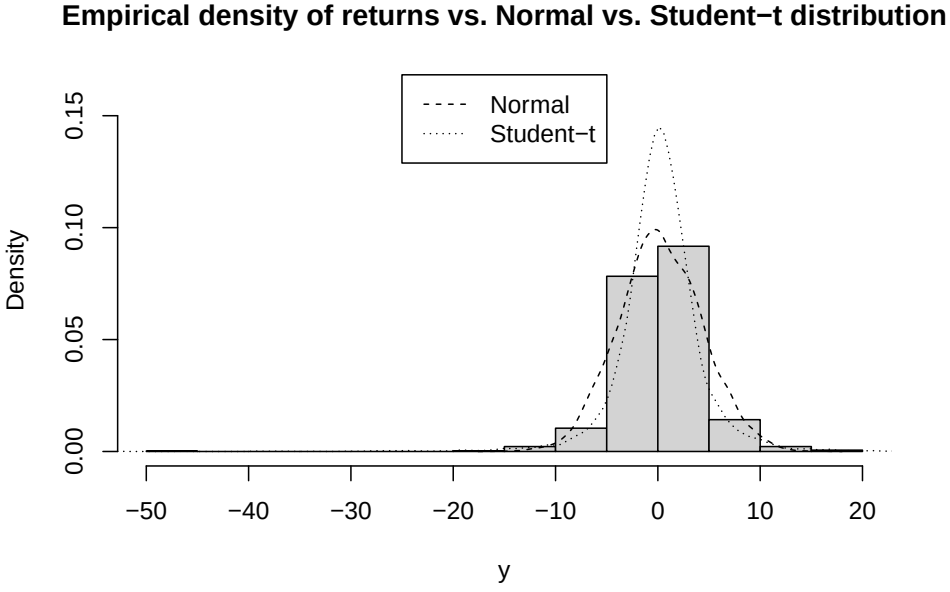

Figure 2.2: Fat tailed returns (BTCUSD), prices from Yahoo! Finance.

Using Equation (2.1) and values for $\alpha \in (0, 1)$, we can compare the different values obtained for $\hat{q}_\alpha$ using a normal and a Student-t distribution vs. the empirical quantiles as explained next.

Nonparametric

The first way we will try to get an estimate of VaR_α is by using the empirical quantile function $\hat{q}(\alpha)$, which can be easily obtained with the R `quantile` function. Non parametric methods do not require assumptions about the distribution of Y. The following two expressions are used to compute the Value at Risk and expected shortfall in this case:

$$\widehat{\text{VaR}}(\alpha) = -\hat{q}(\alpha) \tag{2.3}$$

$$\widehat{\text{ES}}(\alpha) = -\frac{\sum_{t=1}^{T} y_t \cdot I\{y_t < \hat{q}(\alpha)\}}{\sum_{t=1}^{T} I\{y_t < \hat{q}(\alpha)\}} \tag{2.4}$$

Parametric

The standard parametric technique makes use of the normal distribution. Based on a sample vector of size T we model $\boldsymbol{y}$ as a random variable following a normal distribution of fixed mean μ and variance σ^2, e.g., $Y \sim N(\mu, \sigma^2)$. Since $Y \overset{\text{iid}}{\sim} N(\mu, \sigma^2)$, we compute the quantile q_α using the well known expression:

$$\widehat{\text{VaR}}(\alpha) = \hat{\mu} + \hat{\sigma} \times \Phi^{-1}(\alpha) \tag{2.5}$$

where $\Pr(Y \leq q_\alpha) = \alpha$ and $\Phi^{-1}(\alpha)$ is the inverse of the cumulative distribution function (c.d.f) of a standard normal distribution. To obtain $\Phi^{-1}(\alpha)$, we can use the `qnorm` R function, whose first parameter is the probability α. Table 2.1 and Figure 2.3 show the quantiles estimated for values of $\alpha \in \{0.01, 0.02, \ldots, 0.99\}$ using two standard parametric methods vs. the empirical quantiles. In the case of the normal distribution, the unbiased estimators below are commonly used since the Maximum Likelihood (MLE) estimator of σ^2 does converge to the unbiased estimator when $T \to \infty$.

$$\hat{\mu} = \frac{1}{T} \sum_{t=1}^{T} y_t \tag{2.6}$$

$$\hat{\sigma}^2 = \frac{1}{T-1} \sum_{t=1}^{T} (y_t - \hat{\mu})^2 \tag{2.7}$$

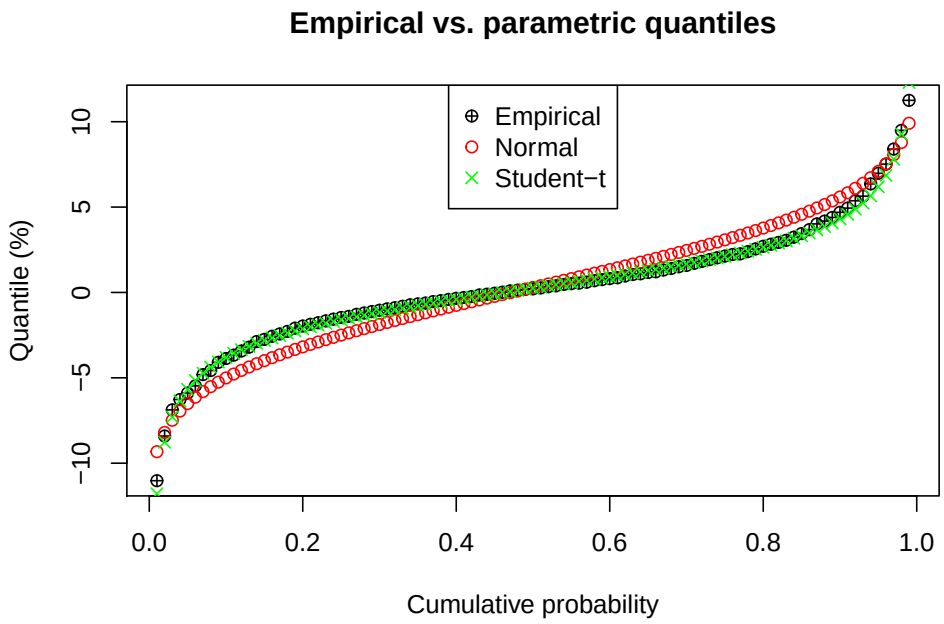

Figure 2.3: Parametric vs. empirical quantiles (BTC/USD).

Table 2.1: Parametric vs. empirical quantiles.

	1%	5%	10%
Empirical	-11.03	-5.89	-3.87
Student	-11.80	-5.68	-3.80
Normal	-9.33	-6.51	-5.01

where μ in Equation (2.6) is frequently assumed to be 0 for high frequency financial applications. The expected shortfall $\mathrm{ES}_\alpha(Y)$ is defined as

$$\mathrm{ES}_\alpha(Y) = \frac{1}{\alpha}\int_0^\alpha \mathrm{VaR}_\gamma(Y)d\gamma, \tag{2.8}$$

which translates into the following closed-form expression in the case of a normal distribution:

$$\widehat{\mathrm{ES}}(\alpha) = -\hat{\mu} + \hat{\sigma}\frac{\phi(\Phi^{-1}(\alpha))}{\alpha} \tag{2.9}$$

where $\phi(x)$ in Equation (2.9) is the probability density function (p.d.f) of a standard normal distribution. As evidenced in Figure 2.1, periods of extreme returns are clustered and exhibit autocorrelation, e.g., extreme negative returns are more likely to happen during highly volatile periods. The generalized autoregressive conditional heteroskedastity (GARCH) model introduced by Bollerslev (1986) tackles this problem with the GARCH(p,q) model shown below:

$$Y_t \sim N(0, \sigma_t^2) \tag{2.10}$$

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i y_{t-1}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-1}^2 \tag{2.11}$$

k-step ahead forecast

$$\sigma_t^2(k) = \begin{cases} \hat{\omega} + (\hat{\alpha}_1 \hat{y}_t^2 + \hat{\beta}_1)\hat{\sigma}_t^2 & k = 1 \\ \hat{\omega} + (\hat{\alpha}_1 + \hat{\beta}_1)\hat{\sigma}_t^2(k-1) & k > 1 \end{cases} \tag{2.12}$$

The popular GARCH(1,1) model in Equation (2.13) below describes the variance σ_t^2 using three pararameters only since $p = q = 1$.

$$\sigma_t^2 = \omega + \alpha y_{t-1}^2 + \beta \sigma_{t-1}^2 \tag{2.13}$$

In this case our parameter vector is $\boldsymbol{\theta} = (\omega, \alpha, \beta) = (\theta_1, \theta_2, \theta_3)$.The well-known *RiskMetrics* model (Morgan and Reuters, 1996) is a specific case of Equation (2.13) where $\omega = 0$, $\alpha = (1 - \lambda)$ and $\beta = \lambda$:

$$\sigma_t^2 = (1 - \lambda) y_{t-1}^2 + \lambda \sigma_{t-1}^2. \tag{2.14}$$

Semiparametric

The CAViaR model (Engle and Manganelli, 2004) describes the behaviour of a quantile q_α at level $\alpha \in \{0, \ldots, 1\}$ as an autoregressive process taking one of the following forms :

- Adaptive:

$$f_t(\boldsymbol{\theta}) = f_{t-1}(\theta_1) + \theta_1 \left(\frac{1}{1 + e^{k \cdot (y_{t-1} + f_{t-1}(\theta_1))}} - \alpha \right) \tag{2.15}$$

- Symmetric absolute value:

$$f_t(\boldsymbol{\theta}) = \theta_1 + \theta_2 f_{t-1}(\boldsymbol{\theta}) + \theta_3 |y_{t-1}| \tag{2.16}$$

- Asymmetric slope:

$$f_t(\boldsymbol{\theta}) = \theta_1 + \theta_2 f_{t-1}(\boldsymbol{\theta}) + \theta_3 |y_{t-1}| I_{(y_{t-1} > 0)} + \theta_4 |y_{t-1}| I_{(y_{t-1} < 0)} \tag{2.17}$$

- Indirect GARCH(1,1):

$$f_t(\boldsymbol{\theta}) = \left[\theta_1 + \theta_2 f_{t-1}^2(\boldsymbol{\theta}) + \theta_3 y_{t-1}^2 \right]^{1/2} \tag{2.18}$$

- Threshold CAViaR (Gerlach et al., 2011):

$$f_t(\boldsymbol{\theta}) = \begin{cases} \theta_1 + \theta_2 f_{t-1}(\boldsymbol{\theta}) + \theta_3 |y_{t-1}|, & y_{t-1} \leq 0 \\ \theta_4 + \theta_5 f_{t-1}(\boldsymbol{\theta}) + \theta_6 |y_{t-1}|, & y_{t-1} > 0 \end{cases} \tag{2.19}$$

where Gerlach et al. (2011) proposed a generalized form for the CAViaR model through Equation (2.19). The threshold variable y_{t-1} in the Threshold CAViaR (T-CAViaR) model above could also be an exogenous variable, while the threshold value could be $\neq 0$. It is important to emphasize that $f_t(\boldsymbol{\theta})$ returns the quantile of interest at level α in contrast to the standard GARCH model in Equation (2.11) which models σ_t^2. This direct modelling of VaR_α at time t can be expressed as follows:

$$\text{VaR}_t = f_t(\boldsymbol{\theta} | \mathcal{I}_{t-1}) + \epsilon_t \tag{2.20}$$

Regardless of the specifics of each model, we also need to find a way to get an estimate $\hat{\theta}$. In the following sections, we present the general optimization and Monte Carlo methods we tested for estimating the CAViaR model based on a vector of observed returns $\boldsymbol{y} = (y_1, y_2, \ldots, y_{t=T})$.

2.2 Maximum Likelihood Estimation

The likelihood $L(\boldsymbol{\theta})$ function of a model is defined as

$$L(\boldsymbol{\theta}) = \prod_{t=1}^{T} f_{\boldsymbol{\theta}}(y_t), \tag{2.21}$$

where $\boldsymbol{y} = (y_1, y_2, \ldots, y_{t=T})$ represents the observed data vector. To find a maximum likelihood estimation (MLE) of $\boldsymbol{\theta}$, we need to find a vector $\theta \in \Theta$ which maximizes $L(\theta)$ [2] given $\boldsymbol{y}$. For this, we use the following objective function at quantile level α:

$$f(\boldsymbol{\theta}, \alpha, \boldsymbol{y}) = \frac{1}{T} \sum_{t=1}^{T} \left(\alpha - I_{(y_t < f_t(\boldsymbol{\theta}))} \right) \left(y_t - f_t(\boldsymbol{\theta}) \right) \tag{2.22}$$

Function (2.22) is the *Regression Quantile Criterion* proposed by Engle and Manganelli (2004). When using adaptive MCMC methods, the likelihood function is slightly modified as

[2] In practice, the natural logarithm of the likelihood function is used for easier computation and better floating point precision.

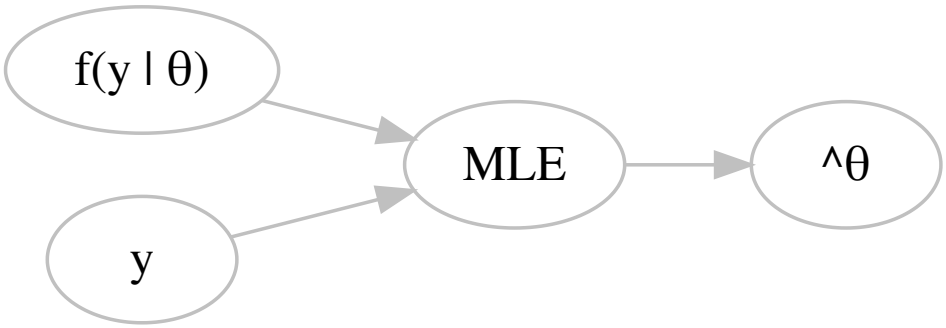

Figure 2.4: Maximum likelihood estimation of the parameter vector.

shown by Gerlach et al. (2011) under the assumption that the error term in Equation (2.20) follows a Skewed-Laplace (SL) distribution (Yu and Moyeed, 2001). In order to estimate the parameters of the CAViaR model, the natural logarithm of Equation (2.21) must be given as objective function to an optimizer and the values of θ which maximize the log-likelihood is our estimate $\hat{\theta}$.

The performance of this method depends on the complexity of the target distribution. Multiple mathematical assumptions on the shape of the likelihood function are needed by most numerical optimization methods whose validity depend on the asymptotic behaviour of the estimators, see Dorsey and Mayer (1995) for a summary of such difficulties.

Starting values

When looking for the global maximum of a likelihood function, we must pick a starting point $\theta_1 \in \Theta$. The optimizer will then apply some sort of algorithm (Nelder-Mead, genetic, simulated annealing, etc.) and stop when one of the following happens: the maximum number of iterations has been reached, an error caused by the shape of the function has occurred or the algorithm has converged.

Given the non-linearity of the CAViaR model, there are good chances that any optimizer will get stuck at a local maximum. To handle this, it is common practice to create a grid in Θ and run the numerical procedure several times from each point in the grid. After doing this, we end up with as many local maxima as points in the grid. The global maximum is then identified from this set as shown in Figure 2.5.

Optimization algorithms

In finding the maximum likelihood estimator of the CAViaR family of models, we tested the following algorithms: [3]

[3]The `pso::psoptim()` and `GA::ga()` algorithms require minimum input, but may take longer to converge.

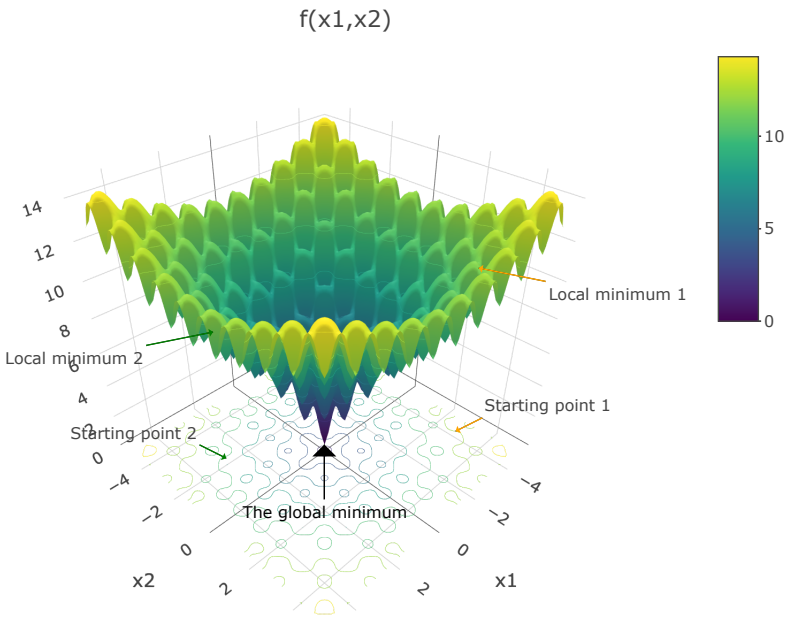

Figure 2.5: Local and global minima of a sample three-dimensional function.

- Quasi-Newton (variable metric algorithm) as implemented by the `stats::optim()` function.
- One-dimensional optimization by Brent (1973) for Equation (2.15) as implemented by `stats::optim()`.
- Genetic algorithm as implemented by the `GA` package (Scrucca, 2021) through the `GA::ga()` function.
- Particle swarm optimization by Clerc (2010) as implemented in the `pso` package (Bendtsen., 2012) through the `pso::psoptim()` function.

2.3 Markov Chain Monte Carlo

In contrast to the frequentist (or classical) MLE method, MCMC algorithms are Bayesian methods which allow us to get a sample of $\boldsymbol{\theta}$. It is this sample that will be used to draw probabilistic conclusions about $\boldsymbol{\theta}$ instead of asymptotic results as it is the case with the MLE method. The Bayesian estimation method requires the use of the following expression (Ruppert and Matteson, 2011):

$$\pi(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{\pi(\boldsymbol{\theta})f(\boldsymbol{y}|\boldsymbol{\theta})}{f(\boldsymbol{y})} = \frac{\pi(\boldsymbol{\theta})f(\boldsymbol{y}|\boldsymbol{\theta})}{\int_{\Theta}\pi(\boldsymbol{\theta})f(\boldsymbol{y}|\boldsymbol{\theta})d\boldsymbol{\theta}} \propto \pi(\boldsymbol{\theta})f(\boldsymbol{y}|\boldsymbol{\theta}) \tag{2.23}$$

where $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ is called the *posterior density*, a function returning the distribution of $\boldsymbol{\theta}$ con-

ditional on the observed data $\boldsymbol{y}$. Here $\pi(\boldsymbol{\theta})$ is the *prior density*, which represents our prior beliefs about the parameter vector $\boldsymbol{\theta}$. In the MCMC implementations below uninformative priors, following a uniform distribution, were used. The functions $f(\boldsymbol{y})$ and $f(\boldsymbol{y}|\boldsymbol{\theta})$ are the marginal and conditional densities respectively.

Equation (2.23) is an important tool in Bayesian statistics as it tells us how to update the prior probabilities after observing the data, this relationship is sometimes referred to as:

$$\text{Posterior} \propto \text{Prior} \times \text{Likelihood} \tag{2.24}$$

where $\propto$ means proportionality up to a constant, i.e., $1/\int \pi(\boldsymbol{\theta}) f(\boldsymbol{y}|\boldsymbol{\theta}) d\boldsymbol{\theta}$ in Equation (2.23). Using Equation (2.23) and a vector of log-returns $\boldsymbol{y}$, we obtain a Markov chain $\boldsymbol{\theta}_{N\times d}$ with each column representing a variable $\boldsymbol{\theta}_i, \forall i \in \{1, \ldots, d\}$ and each row an iteration. This chain is the output sample of interest used for creating *posterior intervals*.

Before we go any deeper into the MCMC algorithm, we might wonder why using MCMC methods instead of classical frequentist methods. An important difference between frequentist and Bayesian methods is that the former consider the parameter vector $\boldsymbol{\theta}$ as fixed (within a random interval) while the latter treat it as a random variable (Efron, 1986). An important reason for choosing to work with Markov chains comes from the minimal requirements on the target distribution f (Robert and Casella, 2009). The properties of this sequence $\{\boldsymbol{\theta}^{(t)}\}$ make their use possible in Bayesian analysis since they enjoy a strong *stability* property and a *stationary distribution* f exists by construction such that:

$$\boldsymbol{\theta}^{(t)} \sim f, \text{ then } \boldsymbol{\theta}^{(t+1)} \sim f \tag{2.25}$$

The stationarity property due to the existence of f imposes a constraint of *irreducibility* on the transition kernel $K(\boldsymbol{\theta}^{(t)}, \boldsymbol{\theta}^{(t+1)})$, which means that the function K must allow the sequence to visit every region of the state-space Θ. An important computational property of such Markov chains is called *ergodicity*, which means that the *limiting distribution* of $\boldsymbol{\theta}^{(t)}$ will be f regardless of the algorithm's initial value $\boldsymbol{\theta}^{(0)}$, meaning that simulating a long enough chain $\boldsymbol{\theta}^{(t)}$ from K with stationary distribution f will produce simulations from f which can be used to draw probabilistic conclusions about $\boldsymbol{\theta}$.

The Markov Chain Monte Carlo (MCMC) methods allow us to get a sample of $\boldsymbol{\theta}$ from a target distribution using a Markov chain. When using this method, we obtain a sample of $\boldsymbol{\theta}$ based on a batch of size NN with chains of length N. Instead of finding a vector which maximizes Equation (2.21), this method outputs multiple sample chains which are then used to obtain an estimate $\hat{\boldsymbol{\theta}}$. The general algorithm is as follows:

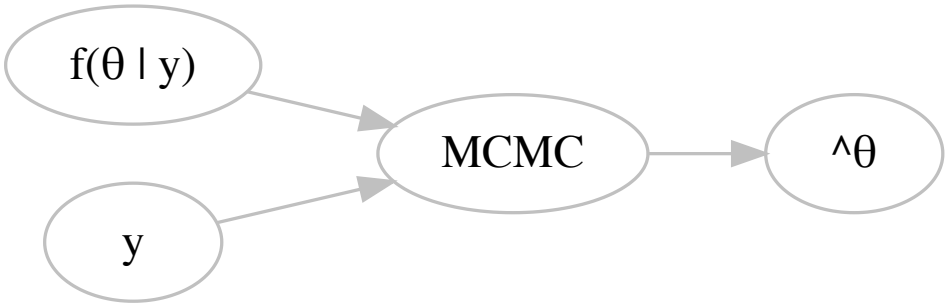

Figure 2.6: MCMC algorithm

Algorithm:

Step 1: Pick the length of the chain N and the number of batches NN
Step 2: Obtain a sample matrix $\boldsymbol{\theta_{1:N}}$
Step 3: Set $\boldsymbol{\theta_1} = \bar{\boldsymbol{\theta}}_{\boldsymbol{1:N}}$
Step 4: Repeat Step 1 to Step 3 NN times
Step 5: Set $\hat{\boldsymbol{\theta}} = \bar{\boldsymbol{\theta}}_{\boldsymbol{1:NN}}$

where $\bar{\boldsymbol{\theta}}_{\boldsymbol{1:N}}$ denotes the mean values of the columns of $\boldsymbol{\theta_{1:N}}$. The MCMC algorithms we used to estimate the CAViaR family of models fall in a general classification of the MCMC methods: the Metropolis–Hastings algorithm (Metropolis and Ulam, 1949). Under this category, the following implementations are tested:

1. Adaptive random walk Metropolis-Hastins with Gaussian proposal following Atchadé and Rosenthal (2005) and Chen and So (2006).
2. Adaptive random walk Metropolis–Hastings with student-t proposal during burn-in and independent kernel for sampling (Gerlach et al., 2011).
3. Robust adaptive Metropolis (Vihola, 2012).

For the first two implementations, a Metropolis within Gibbs strategy is used during adaptation following Roberts and Rosenthal (2009). In the following algorithms, we will use x and y to denote the current point in the chain $\theta^{(t)}$ and the proposal Y_t following Robert and Casella (2009).

Standard Metropolis-Hastings

Algorithm 4 (Robert and Casella, 2009)

Step 1: Generate $Y_t \sim q\left(y|\theta^{(t)}\right)$.
Step 2: Set

$$\theta^{(t+1)} = \begin{cases} Y_t & \text{with probability } \rho\left(\theta^{(t)}, Y_t\right) \\ \theta^{(t)} & \text{with probability } 1 - \rho\left(\theta^{(t)}, Y_t\right) \end{cases} \tag{2.26}$$

where

$$\rho(x,y) = \min\left\{\frac{f(y)}{f(x)}\frac{q(x|y)}{q(y|x)}, 1\right\} \tag{2.27}$$

Below we show the equivalent lines of `C++` code from our implementation in the `caviarma` package:

```
q_x = 1.0;
q_y = 1.0;
for (int i = burn_in; i < nsim; i++) {  // Sampling
  x.row(i) = x.row(i-1);
  Y = arma::mvnrnd(burn_in_mu, burn_in_sigma, 1);
  f_x = BetaPosterior(x.row(i).t());
  f_y = BetaPosterior(Y);

  prob = exp(f_y + q_x - f_x - q_y);
  rho = std::min(1.0, prob);

  if (arma::randu() < rho) {
    x.row(i) = Y.t();
    sampling_counter++;
  }
}  // End of Sampling
```

Where `x.row(i)` is the i-th row of an `arma::mat` object (Sanderson and Curtin, 2016) which stores the value of $\theta^{(t)}$. The output obtained by running the generic MCMC algorithm above is a $\boldsymbol{\theta}_{N\times d}$ matrix. This implementation happens to be a special case of the more generic algorithm 4 since the proposal q is a multivariate *centrally symmetric* (around zero) distribution, i.e., $q(\boldsymbol{\theta} - \mathbf{0}) = q(\mathbf{0} - \boldsymbol{\theta})$. We call this symmetric function g, examples of which are the standard multivariate normal and standard student-t distributions (Serfling, 2014). This property implies that $q(x|y)/q(y|x) = 1$ in Equation (2.26). Since the proposed values for $\theta^{(t+1)}$ in the above code are independent of $\theta^{(t)}$, this is an independent Random Walk Metropolis-Hastings algorithm:

Independent Random Walk Metropolis

Algorithm 6 (Robert and Casella, 2009)

Step 1: Generate $Y_t \sim g(y)$.
Step 2: Set

$$\theta^{(t+1)} = \begin{cases} Y_t & \text{with probability } \min\left\{\frac{f(y)}{f(x)}, 1\right\} \\ \theta^{(t)} & \text{with probability } 1 - \min\left\{\frac{f(y)}{f(x)}, 1\right\} \end{cases}$$

Posterior Intervals

We now turn our attention to the output matrix $\boldsymbol{\theta}_{N\times d}$ and define the following scalar valued functions $\psi = \psi(\boldsymbol{\theta})$ (Ruppert and Matteson, 2011):

$$\bar{\psi} = \frac{1}{N}\sum \psi_i \qquad\qquad s_\psi = \left[\frac{1}{N-1}\sum \left(\psi_i - \bar{\psi}\right)^2\right]^{1/2} \tag{2.28}$$

where $\psi_i = \psi_i(\boldsymbol{\theta}_i), \forall i \in \{1, \ldots, N\}$. The statistics defined in Equation (2.28) are the MCMC sample mean ψ_i, also called the posterior expectation $E(\psi|\boldsymbol{y})$, and the Bayesian standard error s_ψ. The *ergodic theorem* (Robert and Casella, 2009) allow us to create the following interval for $\boldsymbol{\theta}$:

$$\bar{\psi} \pm z_{\alpha/2} s_\psi \tag{2.29}$$

Convergence

The convergence of the Markov chain is guaranteed when following the standard versions of the Metropolis-Hastings algorithm. However, serious mathematical issues arise when the posterior distribution (2.23) is not properly defined, i.e., when the following condition is not respected (Robert, 2007):

$$\int_\Theta \pi(\boldsymbol{\theta}) f(\boldsymbol{y}|\boldsymbol{\theta}) d\boldsymbol{\theta} < \infty \tag{2.30}$$

As a consequence, the posterior distribution is not integrable and the chains cannot converge. To avoid this issue and since the use of improper priors is behind the improper posteriors,

we choose to work with uninformative priors following an integrable uniform distribution. Autocorrelation plots are a popular tool for visually verifying the convergence of a particular sample. For this, we use the `coda::acfplot` function from the `coda` package. However, Equation (2.30) must always hold `true` for the algorithm to be of any use in inference.

2.4 Adaptive MCMC Algorithms

Adaptive MCMC algorithms are used to speed up convergence and improve mixing of the Markov chains. At each step of the simulation, a decision will be made concerning the parameters of the proposal function $q(x, y)$. For instance, increasing the variance σ^2 of a normal proposal would result in larger steps being taken when going from $\theta^{(t)} \to \theta^{(t+1)}$. The adaptation step uses the chain's acceptance rate to decide whether a larger or a smaller step must be taken, i.e., the value of σ_{t+1} in Equation (2.4).

Adaptive RW-Metropolis

A standard case of the adaptive MCMC family is the adaptive Random Walk Metropolis Hastings algorithm with proposal density $q(x, y) = N(x, \sigma^2 I_d)$.

Algorithm (Atchadé and Rosenthal, 2005)

Step 1: Generate $Y_t \sim N\left(\theta^{(t)}, \sigma_t^2 I\right)$.
Step 2: Set

$$\theta^{(t+1)} = \begin{cases} Y_t & \text{with probability } \min\left\{\frac{f(y)}{f(x)}, 1\right\} \\ \theta^{(t)} & \text{with probability } 1 - \min\left\{\frac{f(y)}{f(x)}, 1\right\} \end{cases}$$

Step 3: Compute

$$\sigma_{t+1} : \epsilon_1 \leq \sigma_t + \gamma_t(\rho(\theta^{(t)}, Y_t) - \bar{\tau}) \leq A_1 \tag{2.31}$$

Where $0 < \epsilon_1 < A_1$, $\rho(x,y) = \min\{f(y)/f(x), 1\}$, (γ_t) is a positive sequence of real numbers and $\bar{\tau}$ is the acceptance rate in stationarity. The source code implementing this algorithm can be found in the `caviarma::AMCMC()` function, where the `lower_bound` and `upper_bound` variables correspond to ϵ_1 and A_1 in Equation (2.31), i.e., the limits of σ_t. Note that either a Normal or a Student-t distribution can be used for the proposal since both are centrally symmetric functions.

Robust Adaptive Metropolis

This was the best performing algorithm for estimating the CAViaR model, both in terms of speed and precision. The Robust Adaptive Metropolis (RAM) algorithm avoids the use of the empirical covariance matrix of $\boldsymbol{\theta}_{N\times d}$, which avoids problems caused by target functions without finite second moment.

RAM Algorithm (Vihola, 2012)

Step 1: Compute $Y_t = \theta^{(t-1)} + S_{t-1}U_t$, where $U \overset{\text{iid}}{\sim} q$.
Step 2: Set

$$\theta^{(t+1)} = \begin{cases} Y_t & \text{with probability } \min\left\{\frac{f(Y_t)}{f(\theta^{(t-1)})}, 1\right\} \\ \theta^{(t)} & \text{with probability } 1 - \min\left\{\frac{f(Y_t)}{f(\theta^{(t-1)})}, 1\right\} \end{cases}$$

Step 3: Compute the S_t matrix satisfying:

$$S_t S_t^T = S_{t-1}\left(I + \eta_t(\rho(\theta^{(t-1)}, Y_t) - \alpha_\star)\frac{U_t U_t^T}{\|U_t\|^2}\right)S_{t-1}^T \tag{2.32}$$

Where $I \in \mathbb{R}^{d\times d}$ is the identity matrix, q is a spherically symmetric proposal density, $S_t \in \mathbb{R}^{d\times d}$ is a lower-diagonal matrix, $\{\eta_t\}_{t\geq 1} \subset (0,1]$ is a sequence decaying to zero and $\alpha_\star$ is the target acceptance rate of the algorithm, i.e., 0.234 for $d = 6$ (Gelman et al., 1997).

The following `C++` implementation is nothing but a translation of the `R` implementation available through the `adaptMCCM` package (Scheidegger, 2021). Here we chose to drop any outlier parameter vector which might eventually arise due to memory overflow in the Step 3 of the RAM algorithm. In the implementation below, it is important to note that `M` always exists since it is the Cholesky decomposition of the right-hand side of Equation (2.32), verified to be symmetric and positive definite in *Proposition 1* of Vihola (2012).

```
for (int i = 1; i < nsim; i++) { // Sampling
  U = rt(theta_d, nu);
  Y = x.row(i - 1).t() + (S * U);
  p_val_prop = test_mode ? arma_p_log(Y): armaBetaPosterior(Y);
  double prob = exp(p_val_prop - p_val(i - 1));
  double alpha = std::min(1.0, prob);
```

```
  if (!std::isfinite(alpha)) alpha = 0;
  if (arma::randu() < alpha) {
    x.row(i) = Y.t();  // Accept
    p_val(i) = p_val_prop;
    k++;
  } else {
    x.row(i) = x.row(i - 1);
    p_val(i) = p_val(i - 1);
  }
  ii = i + n_start;
  if (ii < n_adapt) {
    adapt_rate = std::min(1.0, theta_d * std::pow(ii, -gamma));
    M = S * (I_mat + adapt_rate * (alpha - acc_rate) * U * U.t() /
             arma::accu(arma::pow(U, 2))) * S.t();
    eig_val = arma::eig_gen(M);
    tol = M.n_cols * arma::max(arma::abs(eig_val)) * double_eps;
    if (!M.is_sympd() || !arma::imag(eig_val).is_zero() ||
        !arma::all(arma::real(eig_val) > tol)) {
      // Shouldn't happen as per Equation (1) in Vihola (2012)
      // If this  happens due to memory overflow the vector is dropped in R
    }
    S = arma::chol(M).t();
  }
}  // End of Sampling
```

Automatically Tuned Adaptive Metropolis

The last adaptive MCMC algorithm that will be tested was introduced by Yang and Rosenthal (2017) and is available through the `atmcmc` package (Yang, 2014). It works by first using a Metropolis within Gibbs strategy (Roberts and Rosenthal, 2009) as with the `C++` implementation of Equation (2.4) above, this first step is called the *first adaptive phase*. Once an optimal one dimensional acceptance rate of 0.44 (Roberts and Rosenthal, 2009) has been reached by $\theta_j, \forall j \in \{1,2,\ldots,d\}$, a new *transient phase* starts where a non-adaptive Metropolis within Gibbs algorithm is used until the chain reaches the mode of the target distribution. For this, a linear regression of the chain values is used. Once every coordinate θ_j becomes *flat*, which has been diagnosed by the aforementioned linear regression, the *second adaptive phase* of the algorithm begins during which the full proposal matrix Σ_p is updated. Finally, a last *sampling phase* takes place using a conventional non-adaptive Metropolis algorithm.

Chapter 3

Simulation Study

We now move onto testing how good each MCMC algorithm works for estimating the parameters of the CAViaR family of models. For this, we will simulate a dataset and then apply different algorithms to get an estimate $\hat{\theta}$, which we can compare to the known input vector θ.

3.1 Data generating process

The following `T-GARCH-t` set up (Gerlach et al., 2011) will be used to simulate a vector y of returns :

Algorithm:

Step 1: Set σ_1 and ν.
Step 2: Sample t_ν
Step 3: Set $y_1 = \sigma_1 t_\nu \sqrt{\frac{\nu-2}{\nu}}$
Step 4:
If $y_t \leq 0$
Compute $\sigma_t = 0.2 + 0.03|y_{t-1}| + 0.95\sigma_{t-1}$
Else
Compute $\sigma_t = 0.05 + 0.15|y_{t-1}| + 0.75\sigma_{t-1}$
Step 5: Sample t_ν
Step 6: Set $y_t = \sigma_t t_\nu \sqrt{\frac{\nu-2}{\nu}}$
Step 7: Repeat Step 4 to Step 6

where y $= (y_1, y_2, \ldots, y_t)$ is the simulated vector of interest shown in Figure 3.1. For this setup, the true quantile at $t+1$ is given by $q_\alpha(y_{t+1}|\boldsymbol{\beta}) = \sigma_{t+1} T_\nu^{-1}(\alpha)\sqrt{\frac{\nu-2}{\nu}}$, while the known (true) parameter vector of the T-CAViaR model (2.19) can be found with Equations (3.1) to (3.6).

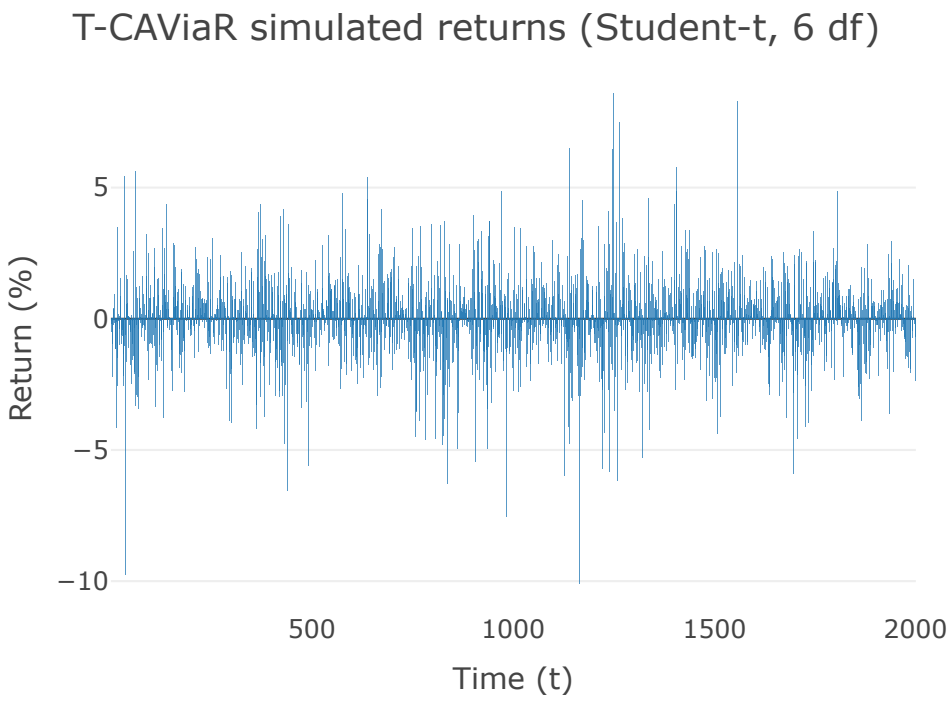

Figure 3.1: Returns from simulated T-GARCH-t set up following Gerlach (2011).

$$\theta_1(\alpha) = 0.2 t_\nu^{-1}(\alpha)\sqrt{\frac{\nu-2}{\nu}} \tag{3.1}$$

$$\theta_2(\alpha) = 0.95 \tag{3.2}$$

$$\theta_3(\alpha) = 0.03 t_\nu^{-1}(\alpha)\sqrt{\frac{\nu-2}{\nu}} \tag{3.3}$$

$$\theta_4(\alpha) = 0.05 t_\nu^{-1}(\alpha)\sqrt{\frac{\nu-2}{\nu}} \tag{3.4}$$

$$\theta_5(\alpha) = 0.75 \tag{3.5}$$

$$\theta_6(\alpha) = 0.15 t_\nu^{-1}(\alpha)\sqrt{\frac{\nu-2}{\nu}} \tag{3.6}$$

3.2 Sample MCMC path

The sample paths below are the last iterations (10%) of different MCMC chains obtained by using the following implementations of the MCMC methods discussed in the Forecasting Methods chapter.

- `amcmc` : ARWMH, Atchade and Rosenthal (2005) - caviarma package.
- `ram` : RAM, Vihola (2012) - caviarma package.

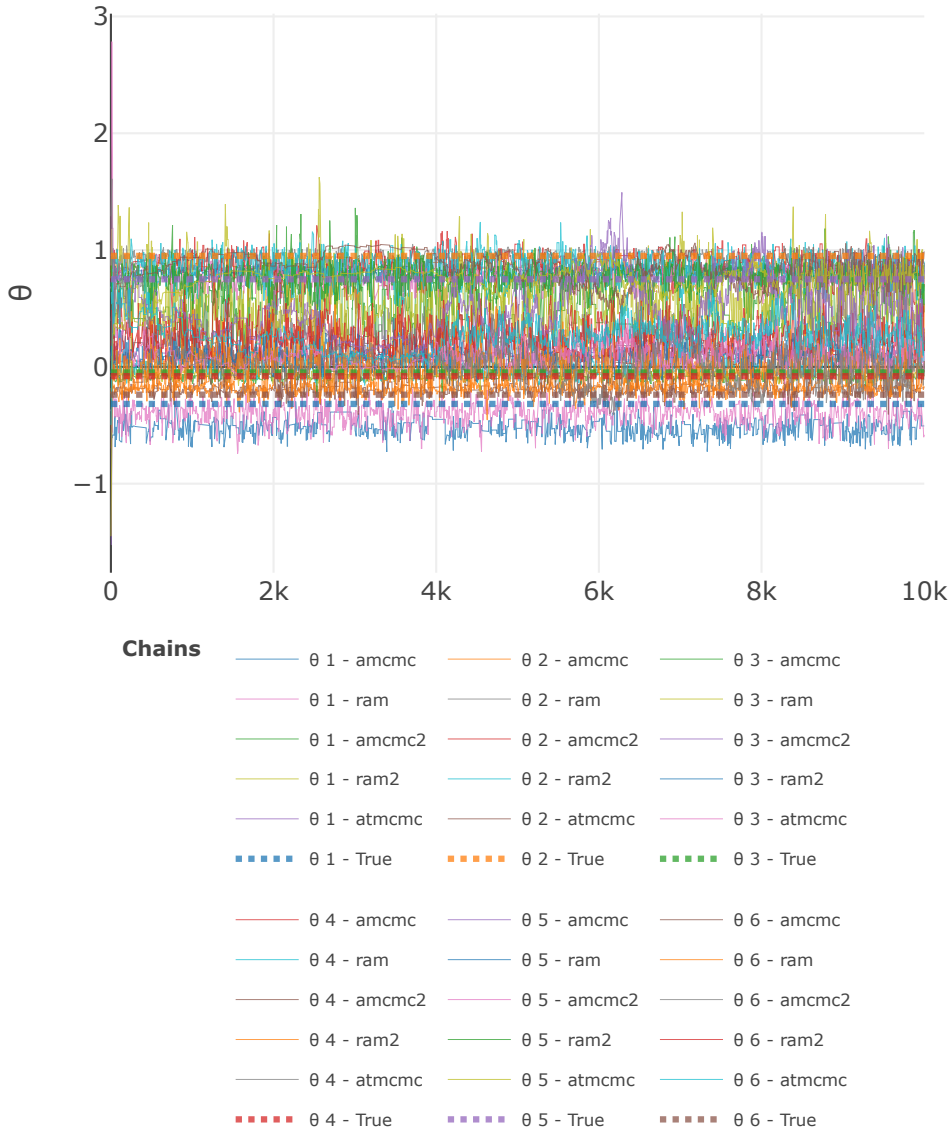

Figure 3.2: Last 1/10 of a sample MCMC path vs. the true (known) parameters.

- `amcmc2` : ARWMH, Atchade and Rosenthal (2005) - fmcmc package.
- `ram2` : RAM, Vihola (2012) - fmcmc package.
- `atmcmc` : ATMCMC, Jinyoung and Rosenthal (2014) - atmcmc package.

These methods were applied to estimate the parameters of the simulated T-CAViaR dataset. Table 3.1 summarizes the performance of these MCMC algorithms when estimating the parameters of the T-CAViaR model.

The summary statistics of each MCMC chain, for each method, are shown in Table 3.1.

Table 3.1: Means of a sample MCMC path of size 1e+05 vs. the true parameter vector.

	1	2	3	4	5	6
True Values	-0.3173	0.9500	-0.0476	-0.0793	0.7500	-0.2380
amcmc	-0.5092	0.8560	-0.0860	0.0919	0.8369	-0.1883
ram	-0.3984	0.9281	-0.0578	0.0150	0.7995	-0.1789
amcmc2	0.5265	0.8467	0.1172	0.0106	0.7525	0.2678
ram2	0.5278	0.8480	0.1113	0.0044	0.7536	0.2723
atmcmc	0.5227	0.8538	0.1104	0.0275	0.7420	0.2703

3.3 Convergence and mixing speed

As we can see through Figure 3.2 and Table 3.1, the `caviarma` implementations (Chaparro Sepulveda, 2019) seems to converge closer to the true known values of $\boldsymbol{\theta}$. Figures 3.3 and 3.5 show the first iterations of these adaptive MCMC methods, i.e., `amcmc` and `ram`, starting from a random point $\boldsymbol{\theta_1}$. Figures 3.7 and 3.8 show the first iterations of the same adaptive MCMC methods, but starting from the true values of $\boldsymbol{\theta}$ to visualize its mixing speed.

An important question one must ask is whether the MCMC chains have converged to their stationary distribution as per equation (2.25). For this, we will analyze the `mcmc` output using the `coda` package (Plummer et al., 2020). We can also see in Figure 3.3 how the AMCMC algorithm behaves during its adaptive phase. The equivalent information for the RAM algorithm is available in Figure 3.5, while its `coda` output can be found in Figure 3.6.

3.4 Simulation results

The results below are obtained from a batch of 30 MCMC chains, each of length 100000, using the RAM, Vihola (2012) method and the T-CAViaR model. Table 3.2 summarizes the results for quantile level $\alpha = 0.01$ and Table 3.3 does the same at level $\alpha = 0.05$. These results show that the standard deviation of the estimates increases with α as expected given that there is more uncertainty (less data) about the location of the lowest/highest quantiles.

Where we test the $\theta_i(\alpha)$ functions described by Gerlach et al. (2011) and we consistently found estimates closer [1] to the true (known) values when compared to Table 1 of Gerlach et al. (2011) even though we used a batch of size 30 instead of 400. We also found contradicting signs for some estimates of Table 1 in the original paper by Engle and Manganelli (2004) when estimating the same models through the RAM implementation by Vihola (2012).

[1] The authors reported $\hat{\theta_5} = -0.734$ even though the true value is $\theta_5 = 0.75$ with $\alpha = 0.05$. We consistently found a value for $\hat{\theta_5} > 0$ in our simulations using the same setup.

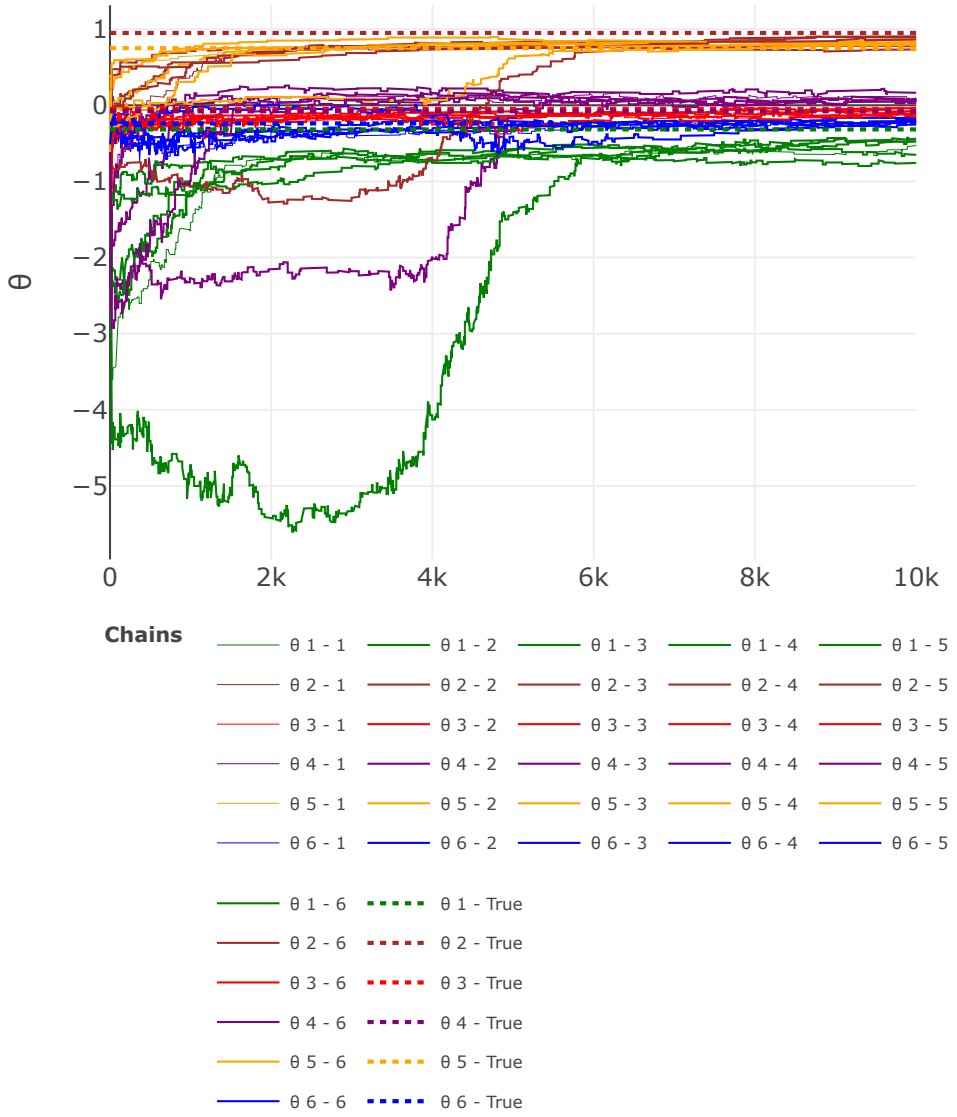

Figure 3.3: Convergence of the AMCMC algorithm - 6 paths starting from random points.

Table 3.2: Statistics of 100000 MCMC draws of size 30 $= 0.01$.

	1	2	3	4	5	6
True Values	-0.5132	0.9500	-0.0770	-0.1283	0.7500	-0.3849
RAM, Vihola (2012)	-0.8317	0.8748	-0.3028	-0.4206	0.6715	-0.3280
Std. Deviation	0.2794	0.0777	0.1251	0.1272	0.0405	0.0862

Table 3.3: Statistics of 100000 MCMC draws of size 30 $= 0.05$.

	1	2	3	4	5	6
True Values	-0.3173	0.9500	-0.0476	-0.0793	0.7500	-0.2380
RAM, Vihola (2012)	-0.3441	0.9498	-0.0304	-0.0769	0.7627	-0.2089
Std. Deviation	0.1150	0.0479	0.0454	0.0551	0.0263	0.0319

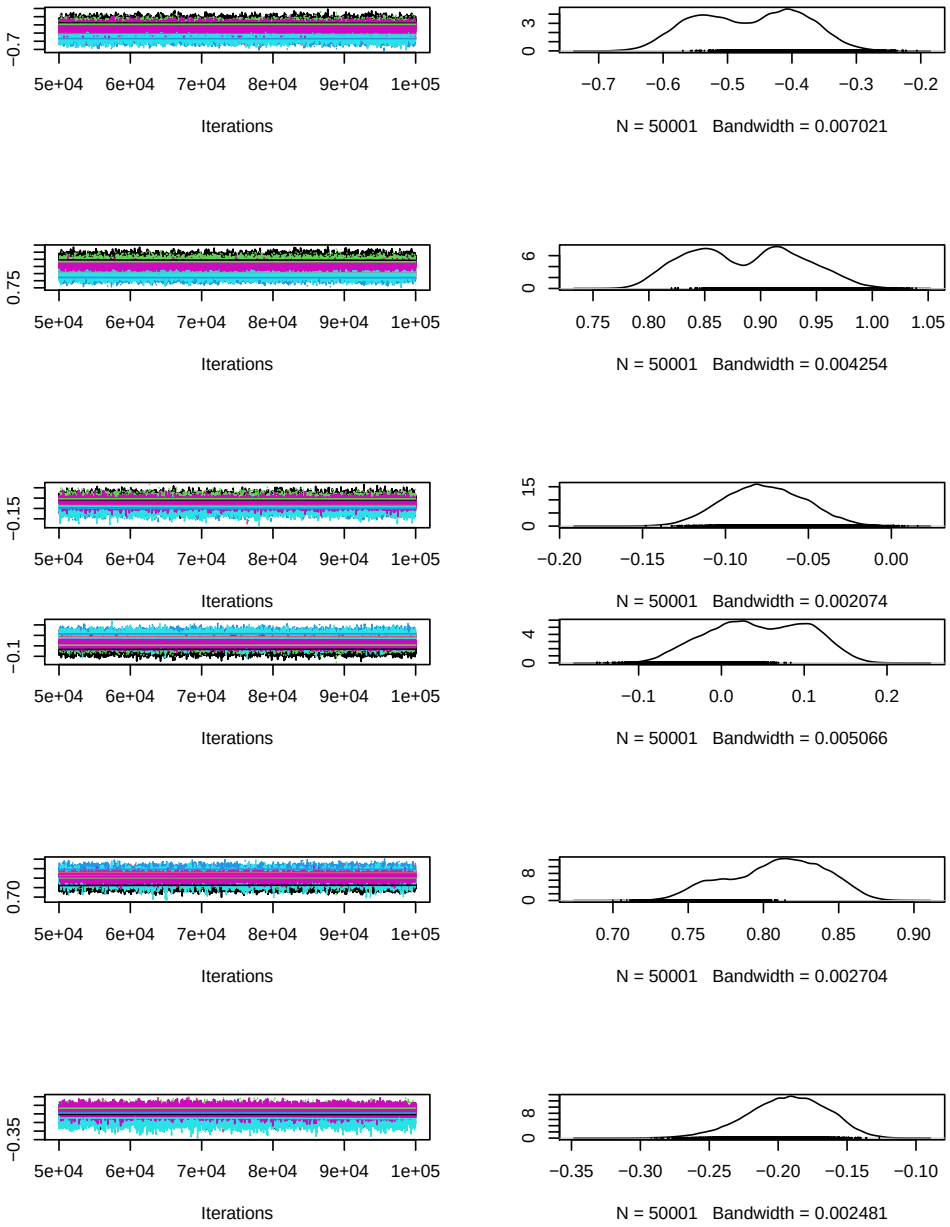

Figure 3.4: Coda MCMC plot (AMCMC).

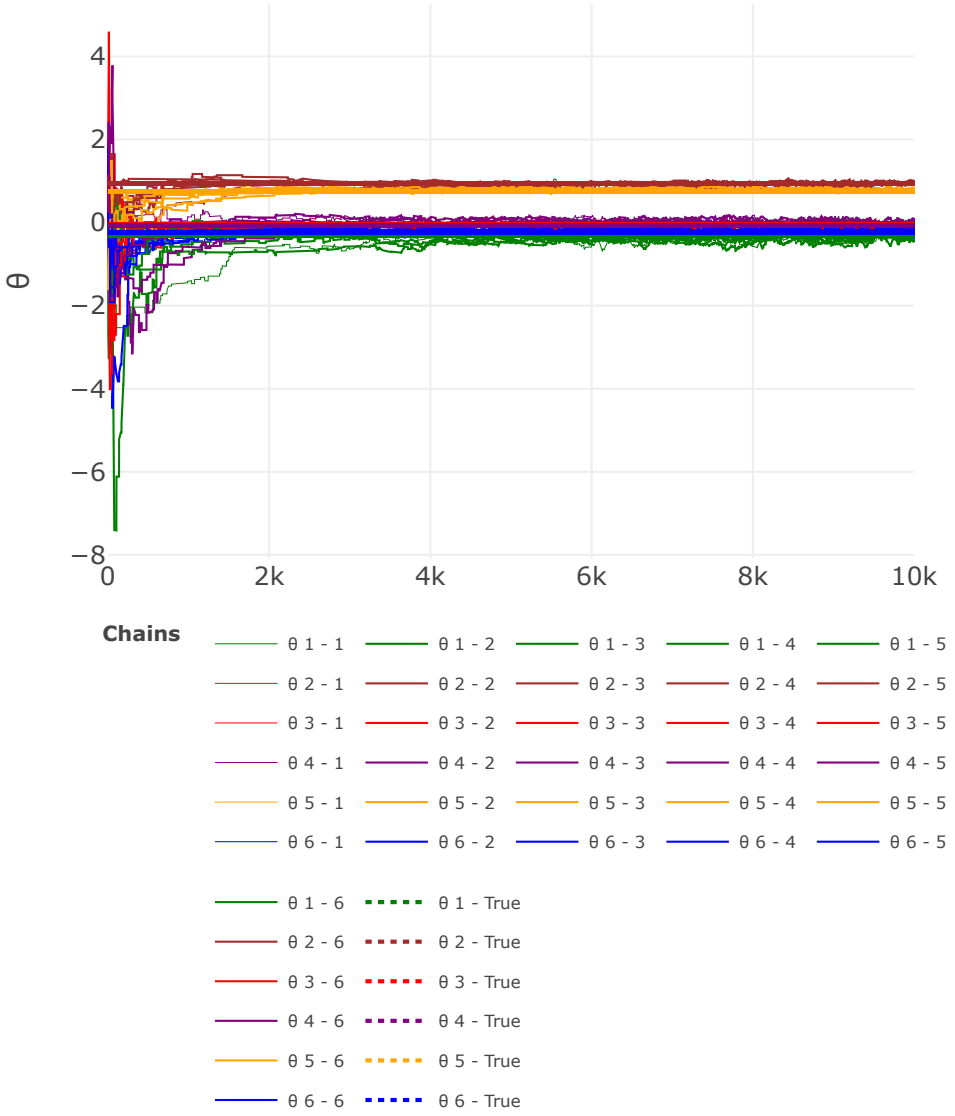

Figure 3.5: Convergence of the RAM algorithm - 6 paths starting from random points.

3.5 Time complexity

We also tested how fast each algorithm performed when getting an estimate of the parameter vector. Table 3.4 shows sample `microbenchmark` output for each method using an input vector of length n equal to 2000. We see that the RAM algorithm is faster than the AM-CMC algorithm depite both samplers running at $\mathcal{O}(d^2)$ speed [2] during adaptation and $\mathcal{O}(n)$ sampling speeds. [3]

Where each function has been evaluated using a chain length of 3×10^4. Since the fastest

[2] d depends on the model and corresponds to the dimension of the parameter vector θ, which depends on the model and is updated one by one on the Gibbs-like RWMH algorithm.

[3] An independent kernel single step update is used after adaptation.

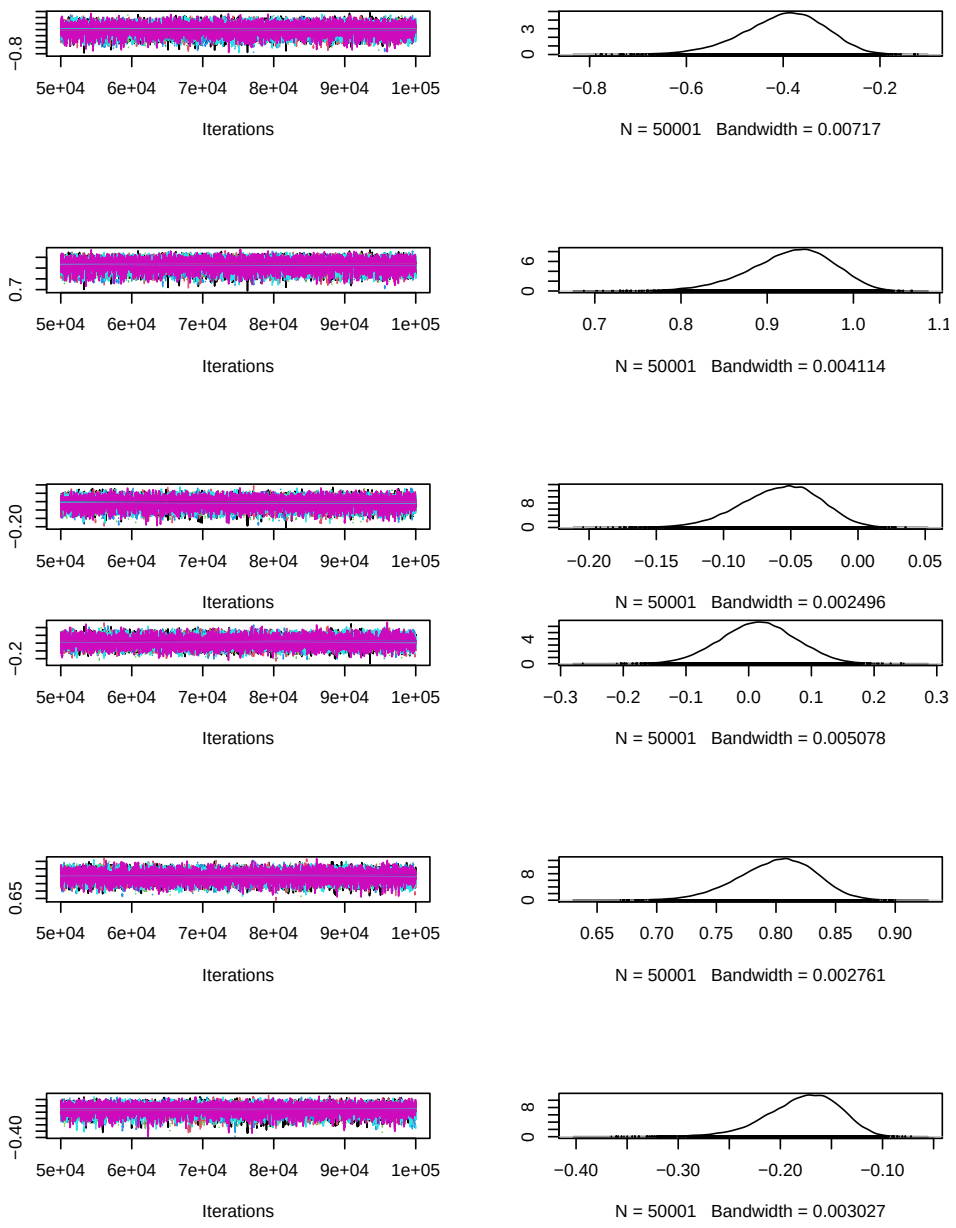

Figure 3.6: Coda MCMC plot (RAM).

Table 3.4: Sample timing of different MCMC implementations (in seconds).

	Min	Lq	Mean	Median	Uq	Max	#eval
RAM (caviarma)	573	576	587	580	595	610	3
AMCMC (caviarma)	1846	1861	1879	1875	1895	1916	3
RAM (fmcmc)	7665	7705	7893	7746	8008	8270	3
AMCMC (fmcmc)	7061	7121	7165	7181	7218	7255	3
ATMCMC (atmcmc)	7081	7203	7557	7325	7795	8265	3

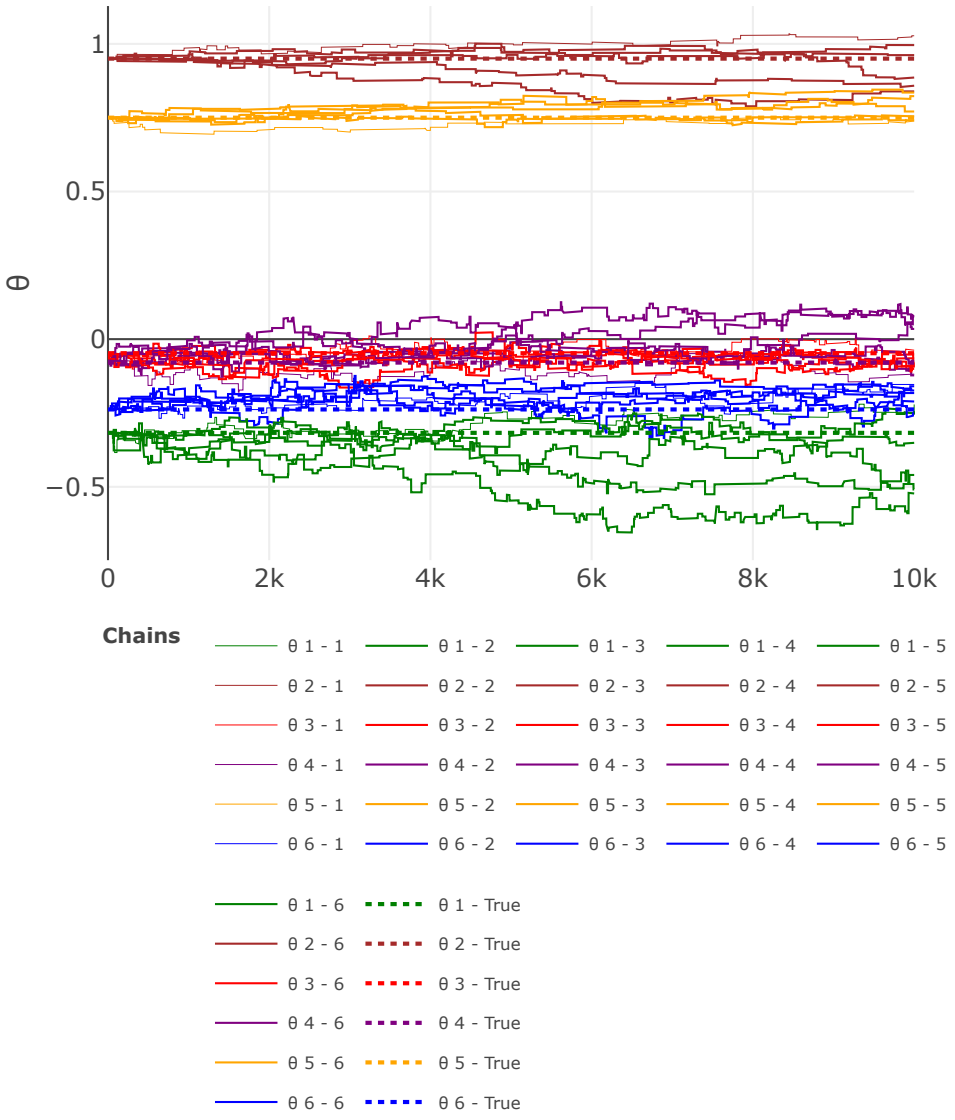

Figure 3.7: Mixing of the AMCMC algorithm - 6 paths starting from the true values.

Table 3.5: Timing of the RAM algorithm, single thread vs. parallel version (in seconds).

	Min	Lq	Mean	Median	Uq	Max	#eval
RAM (caviarma)	40.22	41.0	42.3	41.2	43.6	47.5	30
RAM (caviarma) - parallel	9.98	10.6	10.9	10.9	11.2	12.2	30

implementation appears to be the RAM algorithm in the `caviarma` package, we now time the estimation of the T-CAViaR model using a batch of size 3×10^4 with 3 function evaluations.

With this setup, we can obtain an estimate $\hat{\theta}$ in less than one minute (without parallelization), which would allow us to get a forecast before a new observation becomes available. Using `parallel::mclapply`, the average time is around 10 seconds as seen in Table 3.5.

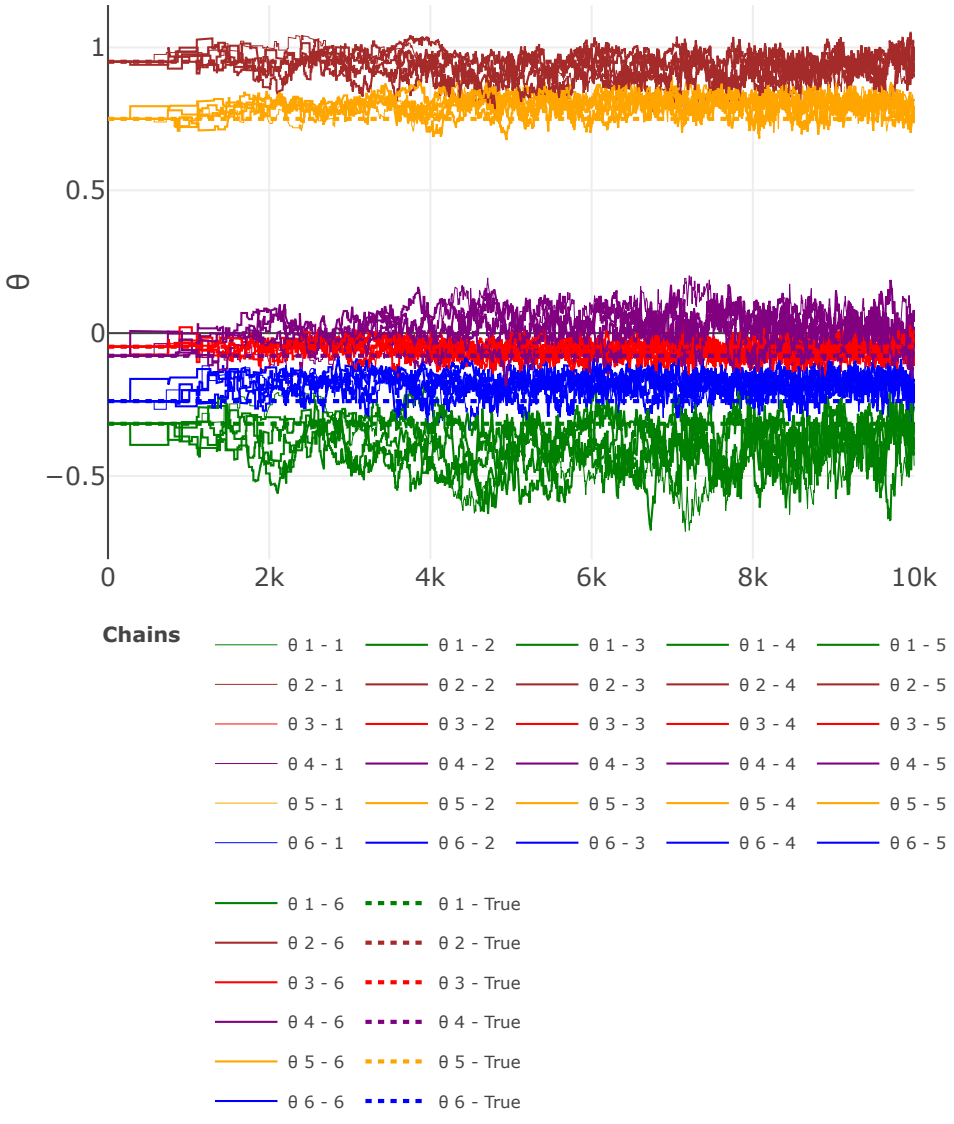

Figure 3.8: Mixing of the RAM algorithm - 6 paths starting from the true values.

Chapter 4

Application

Here we choose the Robust Adaptive MCMC (Vihola, 2012) algorithm as default method when estimating the parameters of the CAViaR family of models. The parameter vector estimate $\hat{\boldsymbol{\theta}}_{\boldsymbol{t}}$ is subsequently used to get a forecast of quantile q_α at time $t+1$. In this section we explain how to set up a computing architecture to serve and manage access to a web application available. This application obtains real time prices from a cryptocurrency exchange through a Websocket API and uses the `shiny` package (Chang et al., 2021) to visualize this data dynamically on the web. The same web application uses a standard HTTP API to estimate the VaR_α of multiple countries using ETF data as a proxy.

4.1 Data

4.1.1 High frequency

The WebSocket protocol is a modern technology that allows two-way communication between a server and a client. It is designed to address important issues that arise when *abusing* the `HTTP` protocol with multiple calls, required when working with high frequency data. Instead of making a different call every k seconds, the server will notify the client whenever new data arrives through the same communication channel.

We will estimate the parameters of the T-CAViaR model (2.19) using an input vector y corresponding to the daily returns of the Bitcoin/USD crypto currency pair. However, our forecast quantile $\hat{q}_\alpha$ will not be available in real time since it takes *time* to get our estimate vector $\hat{\theta}$. The Bitstamp API offers a maximum of 1000 data points of historical data through an HTTP API alongside the above-mentioned real time WebSocket API. Historical data is obtained at a given frequency k, the highest frequency available being 60 seconds. We will hence start by timing the estimation of $\hat{\theta}$ using the `microbenchmark` package (Mersmann, 2019).

The specific alphanumeric pattern of the unique identifiers (symbols or *tickers*) depend on

Table 4.1: The input data structure.

	Length	Class	Mode
high	1000	-none-	character
timestamp	1000	-none-	character
volume	1000	-none-	character
low	1000	-none-	character
close	1000	-none-	character
open	1000	-none-	character

the data source. For instance, the cryptocurrency database follows a `currency1currency2` pattern while the ETF database uses additional symbols such as `^ = .` depending on the asset class. It's important to be aware of these differences when working with multiple data sources since these unique identifiers are usually *unique* and far from standardized.

Historical OHLC data is received in JSON format through the `https://www.bitstamp.net/api/v2/` endpoint. The following R code downloads a JSON file with the latest 1000 hourly prices for `btcusd`:

```
api_call <- sprintf(
  "https://www.bitstamp.net/api/v2/ohlc/%s/?step=%s&limit=%s",
  "btcusd", 86400, 1000
)
out <- jsonlite::fromJSON(api_call)$data
```

Where the JSON data `out` has been parsed into an R `list` of 2 elements: `pair` (BTC/USD) and `ohlc` (the prices).

This data gathering step has been summarized into the `get_price_hist` function, which returns by default the same list of two elements. We can then compute the log-returns vector y which will be passed as first argument to `caviar_methods` when estimating the parameters of the model.

4.1.2 Low frequency

The HTTP protocol uses a different TCP connection every time a `GET` request is made to the server. The highest frequency available through the Yahoo! Finance endpoint is one day.

Yahoo! Finance data can be easily obtained through their website. Alternatively, the following R code downloads the `CSV` file of daily data for `BTC=F`:

```
download_link <- paste0(
  "https://query1.finance.yahoo.com/v7/finance/download/",
  sprintf("BTC=F?period1=1513555200&period2=%s&", as.integer(Sys.time())),
  "interval=1d&events=history&includeAdjustedClose=true"
)
btc_futures <- read.csv(download_link)
summary.default(btc_futures)
```

```
##           Length Class  Mode
## Date      970    -none- character
## Open      970    -none- character
## High      970    -none- character
## Low       970    -none- character
## Close     970    -none- character
## Adj.Close 970    -none- character
## Volume    970    -none- character
```

The resulting `data.frame` has a total of 970 rows and 7 columns as of 2021-10-25 23:09:54. However, several rows need to be ignored since they include no data but a `null` message. [1]

```
idx <- which(btc_futures$Close == "null")
btc_futures <- btc_futures[-idx, ]
dim(btc_futures)
```

```
## [1] 962   7
```

After removing the lines with `null` data, we end up with a total of 962 observations.

Figure 4.1 uses a `plotly` candlestick chart (Sievert et al., 2021) to better help us visualize the data.

The output of applying the `summary` function to the `btc_spot` dataset is shown in Table 4.2. Notice that we first convert the `numeric` data from `character` type by applying the `as.numeric` function to each column of `btc_spot`, excluding the `timestamp` column.

We can also collect additional information such as the `mean` and `quantiles` for different values of $\alpha \in 0.0, 0.1, 0.2, 0.3, 0.4, 0.5$, where the `quantile` at level $\alpha = 0.5$ corresponds to the median of a given vector such as `btc_spot$close`. [2] The mean value of this vector is 2.031×10^4 while the different quantile levels can be found in Table 4.3.

[1] These observations correspond to the days where the futures market is closed.
[2] The original data is still saved as `character`.

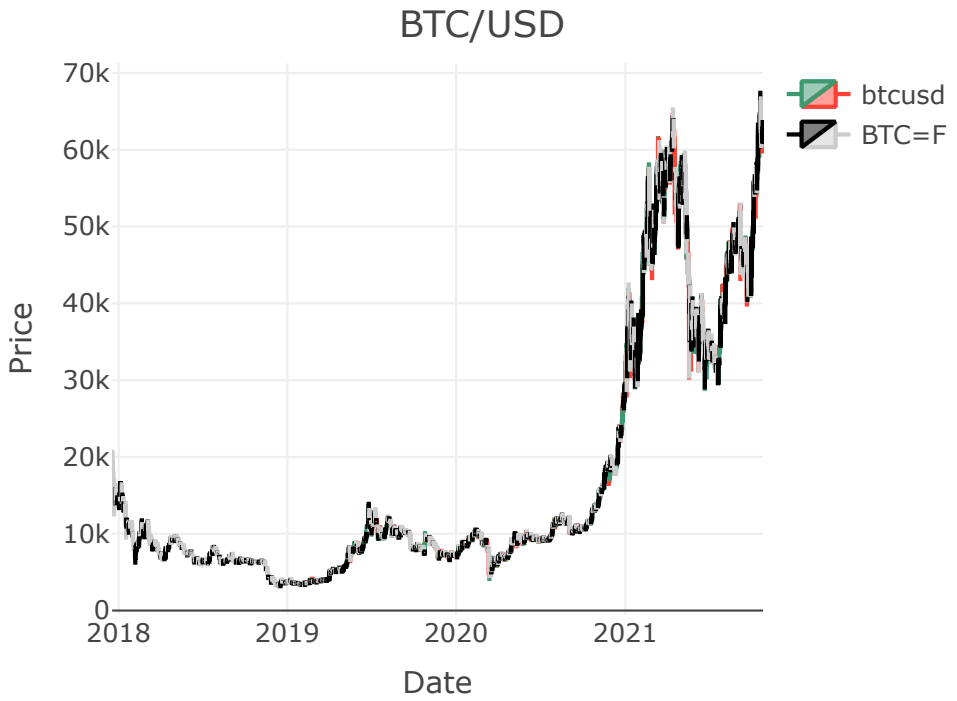

Figure 4.1: Spot and futures prices of the BTC/USD currency pair.

Table 4.2: Summary of the BTC/USD spot prices.

	high	volume	low	close	open
	Min. : 3383	Min. : 162	Min. : 3329	Min. : 3359	Min. : 3358
	1st Qu.: 8411	1st Qu.: 3700	1st Qu.: 8007	1st Qu.: 8212	1st Qu.: 8206
	Median :10575	Median : 5955	Median :10083	Median :10366	Median :10363
	Mean :20868	Mean : 7511	Mean :19564	Mean :20306	Mean :20247
	3rd Qu.:35813	3rd Qu.: 9300	3rd Qu.:33188	3rd Qu.:34636	3rd Qu.:34532
	Max. :67016	Max. :58513	Max. :63529	Max. :65990	Max. :66028

Table 4.3: Closing price quantiles, BTC/USD spot.

0%	10%	20%	30%	40%	50%
3359	5683	7743	8770	9493	10366

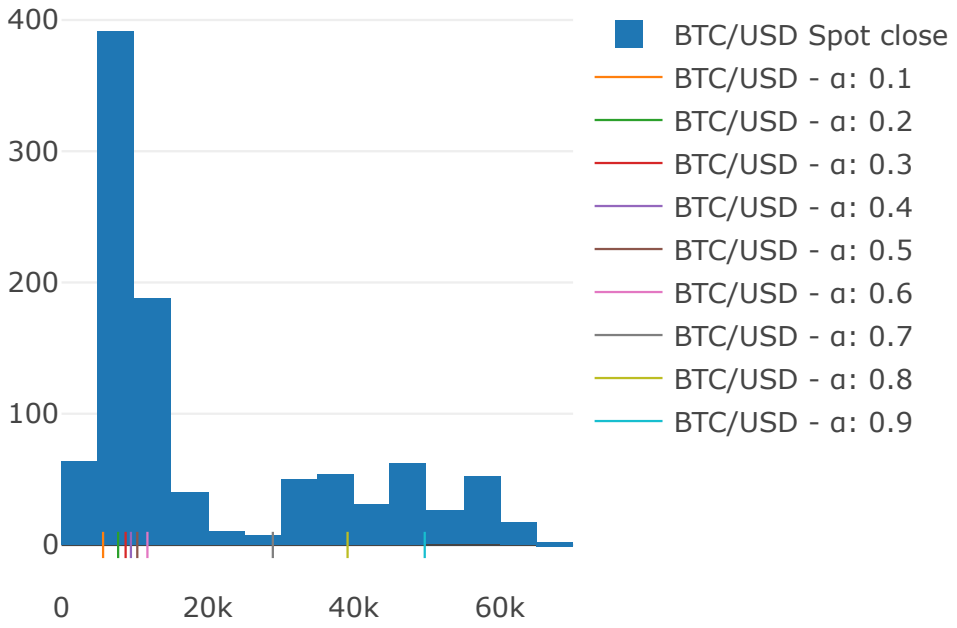

Figure 4.2: Histogram of BTC/USD price quantiles.

This information can also be represented graphically using a `plotly` histogram as shown in Figure 4.2.

```
btc_close <- as.numeric(btc_spot$close)
close_hist <- plotly::plot_ly(
  x = btc_close,type = "histogram", name = "BTC/USD Spot close"
) %>%
add_segments(
  x = quantile(btc_close, probs = seq(0.1, 0.9, 0.1)),
  xend = quantile(btc_close, probs = seq(0.1, 0.9, 0.1)),
  y = -10, yend=10, name = paste("BTC/USD - \U03B1:", seq(0.1, 0.9, 0.1)),
  line = list(width = 1)
)
close_hist
```

And we can also compare this histogram to the corresponding `Close` price of the `btc_futures` table as seen in Figure 4.3.

```
fig <- plotly::plot_ly(alpha = 0.5) %>%
  add_histogram(x = as.numeric(btc_spot$close),
                name = "BTC/USD Spot") %>%
  add_histogram(x = as.numeric(btc_futures$Close),
                 name = "BTC/USD Futures") %>%
```

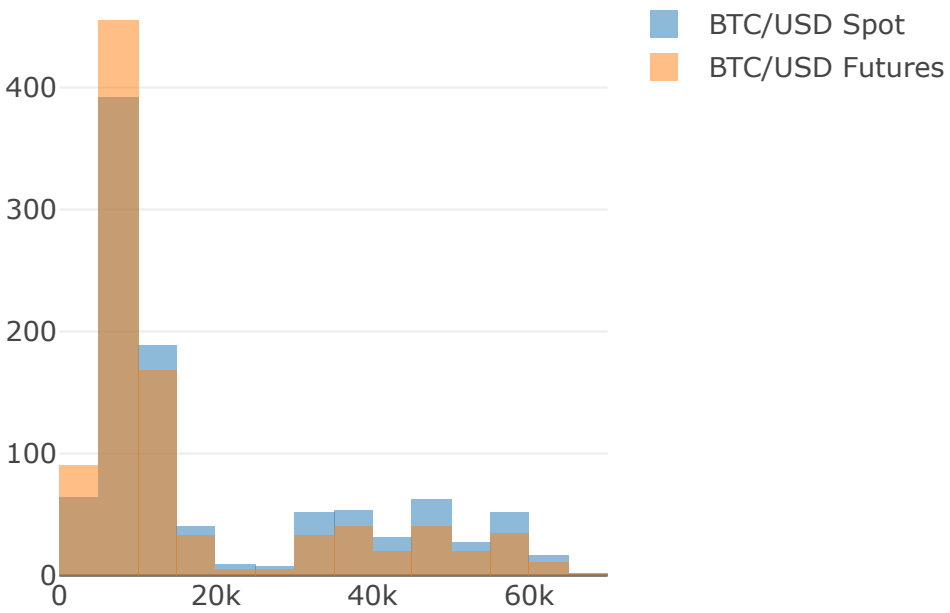

Figure 4.3: Histogram of BTC/USD spot prices overlaying futures prices.

```
  layout(barmode = "overlay")

fig
```

4.1.3 Modelling

For this step we start by computing the log-returns shown in Figure 4.4 as follows:

```
out <- get_price_hist("btcusd", crypto = T)
y <- diff(log(as.numeric(out$ohlc$CLOSE)))
```

After which we pass `y` to the MCMC sampler:

```
out <- caviarma::caviar(y, nsim = 1e5)
```

This step is the bottleneck of the data analysis since it must be repeated multiple times as explained in the Simulation Study section.

4.1.4 Forecasting

After getting our estimate $\hat{\boldsymbol{\theta}}_t$, we obtain our forecast for $t = t + 1$ using Equations (2.15) to (2.19):

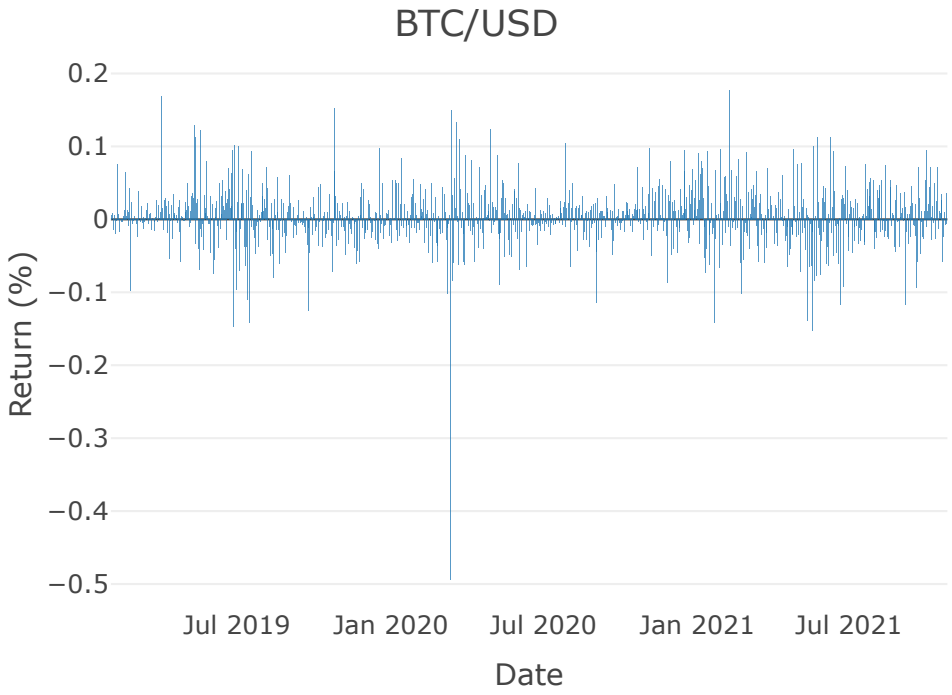

Figure 4.4: Log-returns for the BTC/USD pair, prices (close) from Yahoo! Finance.

Table 4.4: DQ Test statistic for different CAViaR models (GAS package).

	Statistic	P-value
Symmetric Absolute Value	2.74	0.908
Asymmetric Slope	2.10	0.954
Adaptive	26.66	0.000
T-CAViaR	4.58	0.711

```
quantile_forecast <- caviarma::get_forecast(y, out)
```

Figure 4.5 shows sample VaR forecasts obtained using different CAViaR models. We also include the corresponding `tGARCH` and `sGARCH` forecasts for reference (Ardia et al., 2019a). [3] And finally, we show in Table 4.4 the results from the VaR backtest obtained through the `GAS::BacktestVaR()` function.

4.2 Web app

We make use of the `shiny` and `shinyMobile` R packages to build a progressive web app[4] (PWA) that allows a user to gather, analyze and visualize the data introduced in the Data section using the models and methods studied in the Forecasting Methods section. To run

[3] The `tGARCH` and `sGARCH` estimates are obtained using the `MSGARCH::FitMCMC()` function.
[4] This package uses `Framwework7` behind the scenes.

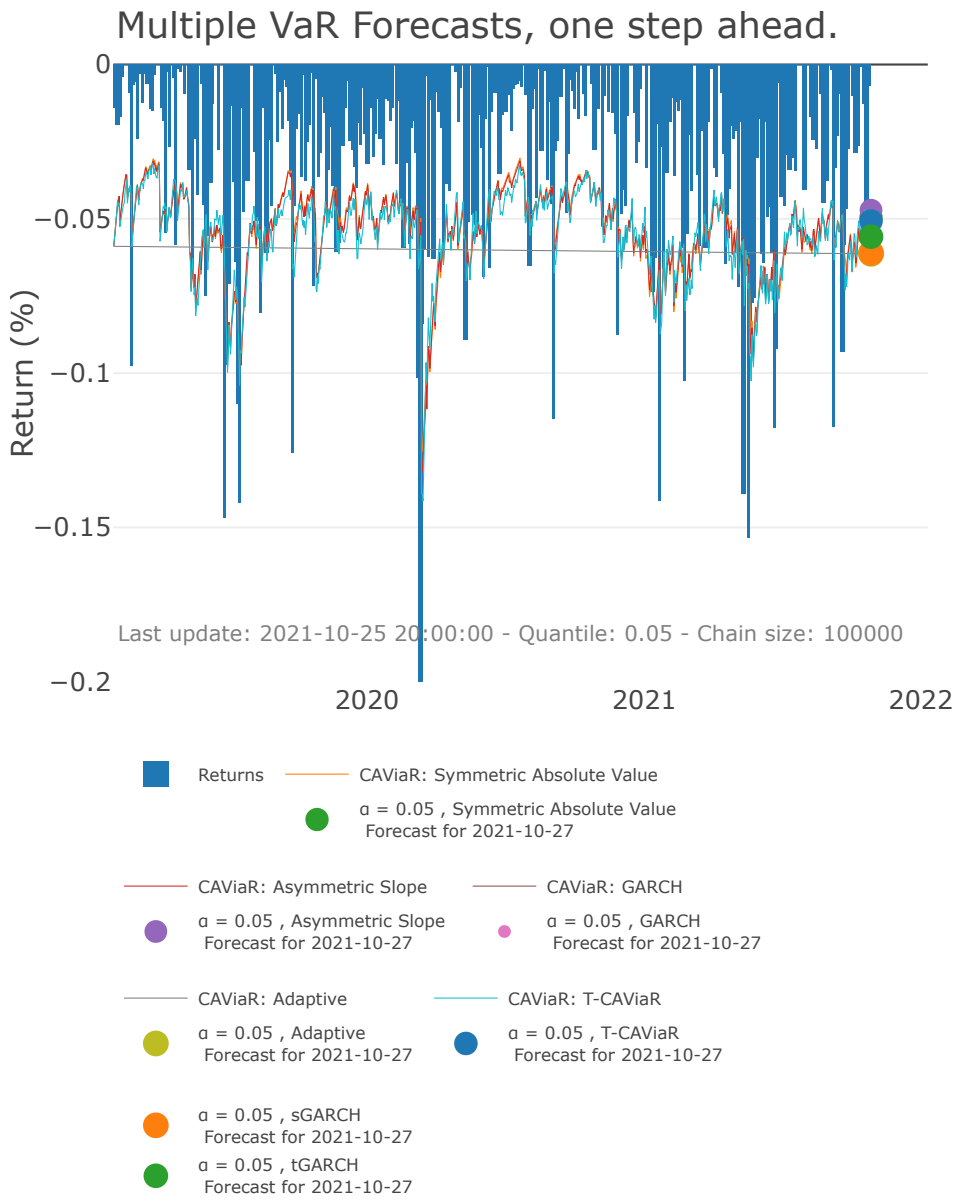

Figure 4.5: VaR (0.05) forecasts using different models.

this application locally, one must first install the `simulr` package (Chaparro Sepulveda, 2021) and then run the app:

```
remotes::install_gitlab("cacsfre/simulr")
simulr::run_simulr()
```

4.2.1 Reactivity

R is a powerful data-centred programming language offering a mathematically intuitive interactive environment which makes data analysis a joyful experience. The `shiny` package offers an intuitive framework for developing web applications using the `R` programming language. A `shiny` application seamlessly connect user interface (UI) input to the back end running R. This makes it straightforward to create a UI to allow users of our code to interact with it without requiring any programming.

This relationship between the input received from the browser (the client) and the output produced by the server is summarized in Figure 4.6 `shiny` handles all the required `css/html/javascript`[5] to create a communication channel between the browser and the server through `R`.

Shiny offers a reactive programming model which makes it easy to make use of an `R` function whenever the `input` object changes. The logic behind each element rendered in the UI is summarized using shiny `modules`. For instance, the code below[6] creates a `plotly` map (the UI element) whenever `input$update_map` changes, i.e., it's clicked.

```
map_cardUI <- function(id) {
  uiOutput(NS(id, "map_card"))
}

map_cardServer <- function(id) {
  moduleServer(id, function(input, output, session) {
    values <- shiny::reactiveValues(map_data = NULL)
    output$map_plot <- plotly::renderPlotly(get_map(values$map_data))
    observeEvent(input$update_map, {values$map_data <- get_map_data()})

    output$map_card <- renderUI({
      shinyMobile::f7Card(
        title = shiny::actionButton("update_map"),
        plotly::plotlyOutput("map_plot")
```

[5] Still, you are likely to write some css/html/javascript in order to customize your shiny apps.
[6] This is a shortened version of the `simulr::map_cardUI()` and `simulr::map_cardServer()` functions.

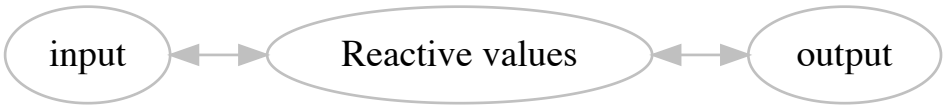

Figure 4.6: Relationship between user input, server output and reactive values.

```
      )
    })
  })
}
```

In the example above, the server recreates the `output$map_plot` session object whenever the user clicks on `input$update_map`. This relationship is due to the `observeEvent(input$update_map, {values$map_data <- get_map_data()})` expression, which invalidates the reactive relationship between `values$map_data` and `output$map_plot` whenever `input$update_map` is clicked by the user.

4.2.2 Deployment

Deploying a `shiny` application can be pretty simple using the `https://www.shinyapps.io/` service by RStudio. This works fine for demos consuming limited resources. However, it requires an upgrade once we need to handle multiple users or greater computing resources. The two typical approaches to scale a shiny app in an enterprise context are Shiny Server and ShinyProxy, where `shinyproxy`[7] tries to fill the gaps of the open-source version of `shiny-server` relying exclusively on the open-source `shiny` package. Another advantage of ShinyProxy is its use of docker, offering each user its own independent shiny app environment and R process. This isolation obtained through `docker` containers is important both for security and performance reasons. This architecture is outlined in Figure 4.7.

Compared to the open-source version of `shiny-server` in Figure 4.8.

[7] ShinyProxy is an open-source technology built with `java`, its source code is available at `https://github.com/openanalytics/shinyproxy/tree/master/src/main/java`.

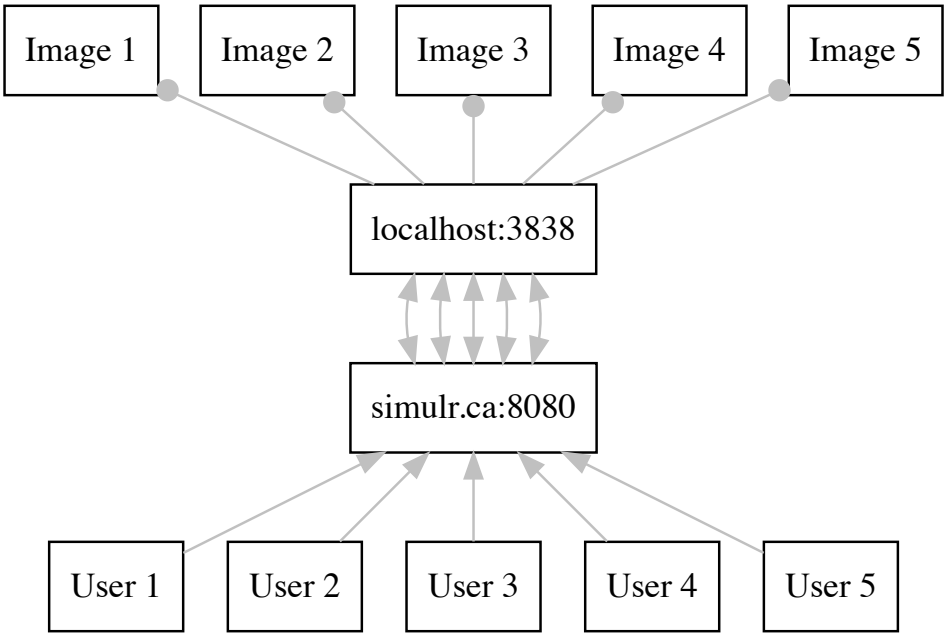

Figure 4.7: Outline of the shinyproxy architecture.

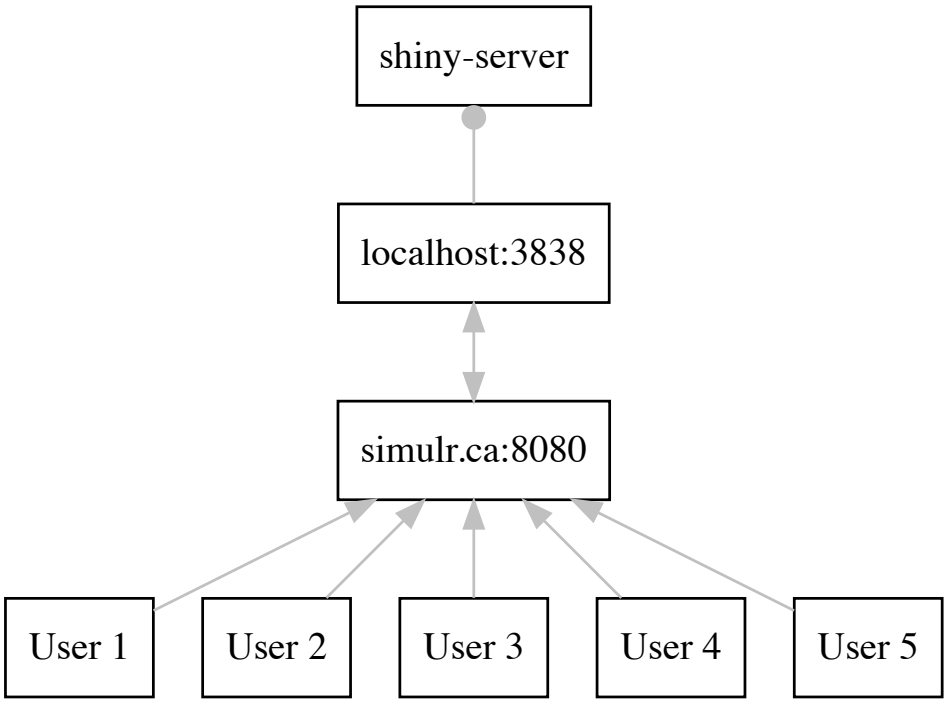

Figure 4.8: Outline of the open-source shiny-server architecture.

Conclusion

We tested several adaptive MCMC algorithms to estimate the parameters of the CAViaR family of models. We found that the Robust Adaptive MCMC algorithm (Vihola, 2012) is best suited to explore the target distribution in the case of a heavy-tailed return vector. Given the time taken to run the Monte Carlo simulations, we used `C++` to speed up key loops which made it possible to get a forecast in under a minute using a standard personal computer . We were also able to build and deploy a web application using exlusively the `R` programming language to gather, analyze and visualize real time financial data. Currently, the app depends entirely on its single R process available through `shiny`, which has serious limitations whenever the server receives multiple messages within a second, which can be pretty intense when markets drop. Since there is only one R process available to treat this incoming data, analyzing it and generating output, the full application can freeze whenever market activity is too high. This issue can currently be handled simply by disabling the WebSocket updates and relying exclusively on HTTP requests happening once per minute instead. However, real time data can be critical for decision-making in which case a possible solution is to delegate this task to the client through Javascript or using a multithreaded web framework to handle the incoming data.

An important portfolio management element has been left aside throughout this document, i.e., the correlation between asset returns. The factor copula models introduced by Krupskii and Joe (2015) offer an intuitive way to model the correlation structure among traditional financial markets and cryptocurrency returns. Another important issue that has been left off this work concerns ensuring non-crossing quantiles as described by Liu and Luger (2017).

Cryptocurrency markets are new and growing exponentially. It is difficult to predict which cryptocurrencies will survive, but it seems like they will all be using some form of blockchain technology for the foreseeable future. This can either mean that organizations will be using this newer and better technology as a replacement to old technology while still relying heavily on human labour or that this newer and better technology will play an important role in the automation of modern labour.

Bibliography

International Energy Agency. Global energy review. 2020. URL https://www.iea.org/reports/global-energy-review-2020/renewables.

David Ardia, Keven Bluteau, Kris Boudt, Leopoldo Catania, and Denis-Alexandre Trottier. Markov-Switching GARCH Models in R: The MSGARCH Package. *Journal of Statistical Software*, 91(4):138, 2019a. doi: 10.18637/jss.v091.i04. URL https://www.jstatsoft.org/index.php/jss/article/view/v091i04.

David Ardia, Keven Bluteau, and Maxime Rüede. Regime changes in Bitcoin GARCH volatility dynamics. *Finance Research Letters*, 29:266–271, 2019b. ISSN 1544-6123. doi: https://doi.org/10.1016/j.frl.2018.08.009. URL https://www.sciencedirect.com/science/article/pii/S1544612318303970.

David Ardia, Keven Bluteau, Leopoldo Catania, and Denis-Alexandre Trottier. *MSGARCH: Markov-Switching GARCH Models*, 2020. URL https://github.com/keblu/MSGARCH. R package version 2.42.

Yves F Atchadé and Jeffrey S Rosenthal. On adaptive Markov Chain Monte Carlo algorithms. *Bernoulli*, 11(5):815–828, 2005.

Dirk G. Baur and Thomas K. McDermott. Is gold a safe haven? International evidence. *Journal of Banking & Finance*, 34(8):1886–1898, 2010. ISSN 0378-4266. doi: https://doi.org/10.1016/j.jbankfin.2009.12.008. URL https://www.sciencedirect.com/science/article/pii/S0378426609003343. New Contributions to Retail Payments: Conference at Norges Bank (Central Bank of Norway) 1415 November 2008.

Claus Bendtsen. *pso: Particle Swarm Optimization*, 2012. URL https://CRAN.R-project.org/package=pso. R package version 1.0.3.

Bloomberg. Argentina's biggest futures market plans to join the bitcoin party. 2017. URL https://www.bloomberg.com/news/articles/2017-11-02/argentina-s-biggest-futures-market-plans-to-join-bitcoin-party.

Bloomberg. Bitcoin surges to highest since july 2019 after paypal embrace. 2020-10-21. URL https://www.bloomberg.com/news/articles/2020-10-21/bitcoin-on-the-brink-of-fresh-year-high-following-paypal-embrace.

Bloomberg. Crypto mining booms on cheap, subsidized energy in argentina. 2021a. URL https://www.bloomberg.com/news/articles/2021-05-31/crypto-mining-booms-on-cheap-subsidized-energy-in-argentina.

Bloomberg. China widens ban on crypto transactions; bitcoin tumbles. 2021b. URL https://www.bloomberg.com/news/articles/2021-09-24/china-deems-all-crypto-related-transactions-illegal-in-crackdown.

Bloomberg. Tesla bets on bitcoin in blue-chip boost to cryptocurrency. 2021c. URL https://www.bloomberg.com/news/articles/2021-02-08/tesla-bets-1-5-billion-on-bitcoin-in-new-policy-crypto-surges.

Bloomberg. Even garbage is using blockchain now. 2021d. URL https://www.bloomberg.com/news/articles/2021-03-18/even-garbage-is-using-blockchain-now.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. ISSN 0304-4076. doi: https://doi.org/10.1016/0304-4076(86)90063-1. URL https://www.sciencedirect.com/science/article/pii/0304407686900631.

Richard P Brent. Some efficient algorithms for solving systems of nonlinear equations. *SIAM Journal on Numerical Analysis*, 10(2):327–344, 1973.

Vitalik Buterin. Ethereum: A next-generation smart contract and decentralized application platform. *URL: https://ethereum.org/en/whitepaper*, 2013.

Winston Chang, Joe Cheng, JJ Allaire, Carson Sievert, Barret Schloerke, Yihui Xie, Jeff Allen, Jonathan McPherson, Alan Dipert, and Barbara Borges. *Shiny: Web Application Framework for R*, 2021. URL https://shiny.rstudio.com/. R package version 1.7.1.

Carlos A. Chaparro Sepulveda. *caviarma: Estimation of the CAViaR model using MCMC algorithms*, 2019. https://gitlab.com/cacsfre/caviarma.

Carlos A. Chaparro Sepulveda. *simulr: A shiny application for the analysis of real time financial data*, 2021. https://gitlab.com/cacsfre/simulr.

Cathy WS Chen and Mike KP So. On a threshold heteroscedastic model. *International Journal of Forecasting*, 22(1):73–89, 2006.

Maurice Clerc. *Particle Swarm Optimization*. 2010. ISBN 9781905209040. doi: 10.1002/9780470612163. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470612163.

Robert E Dorsey and Walter J Mayer. Genetic algorithms for estimation problems with multiple optima, nondifferentiability, and other irregular features. *Journal of Business & Economic Statistics*, 13(1):53–66, 1995.

Bradley Efron. Why isn't everyone a Bayesian? *The American Statistician*, 40(1):1–5, 1986.

Robert F Engle and Simone Manganelli. CAViaR: Conditional autoregressive Value at Risk by regression quantiles. *Journal of Business & Economic Statistics*, 22(4):367–381, 2004.

Andrew Gelman, Walter R Gilks, and Gareth O Roberts. Weak convergence and optimal scaling of random walk metropolis algorithms. *The Annals of Applied Probability*, 7(1): 110–120, 1997.

Richard H Gerlach, Cathy WS Chen, and Nancy YC Chan. Bayesian time-varying quantile forecasting for Value at Risk in financial markets. *Journal of Business & Economic Statistics*, 29(4):481–492, 2011.

Pavel Krupskii and Harry Joe. Structured factor copula models: Theory, inference and computation. *Journal of Multivariate Analysis*, 138:53–73, 2015.

Nikolaos A Kyriazis. A survey on empirical findings about spillovers in cryptocurrency markets. *Journal of Risk and Financial Management*, 12(4):170, 2019.

Xiaochun Liu and Richard Luger. Markov-switching quantile autoregression: a gibbs sampling approach. *Studies in Nonlinear Dynamics & Econometrics*, 22(2), 2017.

Karl Marx. *Das Kapital: kritik der politischen ökonomie*, volume 1. O. Meissner, 1890.

Ralph C Merkle. Protocols for public key cryptosystems. In *1980 IEEE Symposium on Security and Privacy*, pages 122–122. IEEE, 1980.

Olaf Mersmann. *Microbenchmark: Accurate Timing Functions*, 2019. URL https://github.com/joshuaulrich/microbenchmark/. R package version 1.4-7.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

Léopold Migeotte and Janet Lloyd. *The Economy of the Greek Cities: From the Archaic Period to the Early Roman Empire*. University of California Press, 1 edition, 2009. ISBN 9780520253650. URL http://www.jstor.org/stable/10.1525/j.ctt1pn6p2.

J. P. Morgan and Reuters. Riskmetrics. Technical Document Fourth Edition, J. P. Morgan and Reuters, New York, 1996. URL https://www.msci.com/documents/10199/5915b101-4206-4ba0-aee2-3449d5c7e95a.

Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *URL: https://bitcoin.org/bitcoin.pdf*, 2008.

Jeroen Ooms. *Openssl: Toolkit for Encryption, Signatures and Certificates Based on OpenSSL*, 2021. URL https://github.com/jeroen/openssl. R package version 1.4.5.

PayPal. Paypal cryptocurrency terms and conditions. 2021. URL https://www.paypal.com/us/webapps/mpp/ua/cryptocurrencies-tnc.

Martyn Plummer, Nicky Best, Kate Cowles, Karen Vines, Deepayan Sarkar, Douglas Bates, Russell Almond, and Arni Magnusson. *Coda: Output Analysis and Diagnostics for MCMC*, 2020. URL https://CRAN.R-project.org/package=coda. R package version 0.19-4.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. URL https://www.R-project.org/.

Christian Robert. *The Bayesian choice: From decision-theoretic foundations to computational implementation*. Springer Science & Business Media, 2007.

Christian P. Robert and George Casella. *Introducing Monte Carlo Methods with R (Use R)*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2009. ISBN 1441915753.

Gareth O. Roberts and Jeffrey S. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009. doi: 10.1198/jcgs.2009.06134. URL https://doi.org/10.1198/jcgs.2009.06134.

David Ruppert and David S Matteson. *Statistics and Data Analysis for Financial Engineering*, volume 13. Springer-Verlag, New York, 2011.

Conrad Sanderson and Ryan Curtin. Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

Andreas Scheidegger. *adaptMCMC: Implementation of a Generic Adaptive Monte Carlo Markov Chain Sampler*, 2021. URL https://github.com/scheidan/adaptMCMC. R package version 1.4.

Luca Scrucca. *GA: Genetic Algorithms*, 2021. URL https://luca-scr.github.io/GA/. R package version 3.2.1.

Robert J Serfling. Multivariate symmetry and asymmetry. *Wiley StatsRef: Statistics Reference Online*, 2014.

Carson Sievert, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. *plotly: Create Interactive Web Graphics via plotly.js*, 2021. https://plotly-r.com, https://github.com/plotly/plotly.R.

Statista. Number of crypto coins since 2013. 2021. URL https://www.statista.com/statistics/863917/number-crypto-coins-tokens.

Stavros Stavroyiannis. Value at Risk and related measures for the bitcoin. *Journal of Risk Finance*, 19(2):127–136, 2018. URL https://EconPapers.repec.org/RePEc:eme:jrfpps:jrf-07-2017-0115.

Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 4th edition, 2013. ISBN 0321563840.

Matti Vihola. Robust adaptive metropolis algorithm with coerced acceptance rate. *Statistics and Computing*, 22(5):997–1008, 2012.

Gang-Jin Wang, Chi Xie, Danyan Wen, and Longfeng Zhao. When bitcoin meets economic policy uncertainty (EPU): Measuring risk spillover effect from EPU to bitcoin. *Finance Research Letters*, 31(C), 2019. URL https://EconPapers.repec.org/RePEc:eee:finlet:v:31:y:2019:i:c:s1544612318305749.

Halbert White, Tae-Hwan Kim, and Simone Manganelli. VAR for VaR: Measuring tail dependence using multivariate regression quantiles. *Journal of Econometrics*, 187(1):169–188, 2015.

Jinyoung Yang. *atmcmc: Automatically Tuned Markov Chain Monte Carlo*, 2014. URL https://CRAN.R-project.org/package=atmcmc. R package version 1.0.

Jinyoung Yang and Jeffrey S Rosenthal. Automatically tuned general-purpose MCMC via new adaptive diagnostics. *Computational Statistics*, 32(1):315–348, 2017.

Keming Yu and Rana A. Moyeed. Bayesian quantile regression. *Statistics & Probability Letters*, 54(4):437–447, 2001. ISSN 0167-7152. doi: https://doi.org/10.1016/S0167-7152(01)00124-9. URL https://www.sciencedirect.com/science/article/pii/S0167715201001249.