Quantitative finance with R and cryptocurrencies

Dean Fantazzini

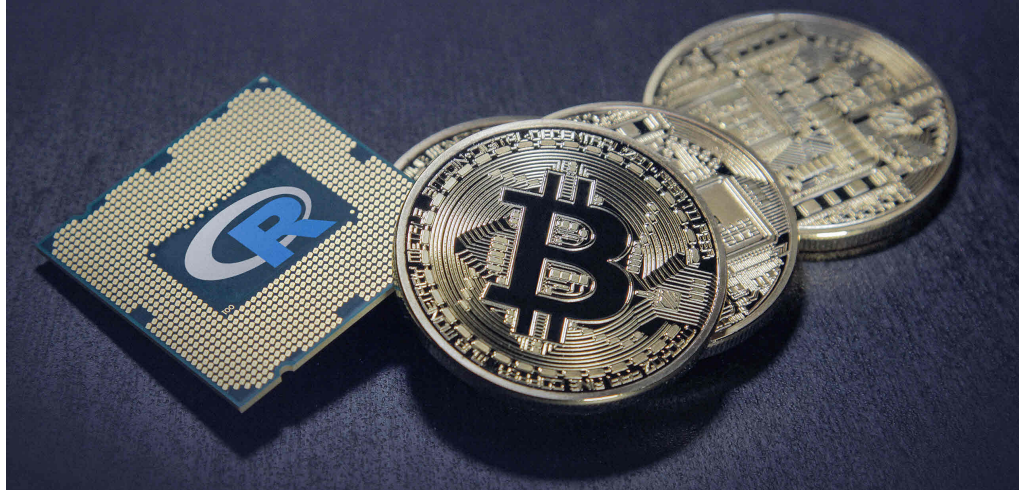
February 2020

QUANTITATIVE

FINANCE WITH R

AND

CRYPTOCURRENCIES



DEAN FANTAZZINI

How the book is structured?

Part I

- A brief review of Cryptocurrencies and Bitcoin
- Where to get (free) Bitcoin and cryptocurrency data?
- Liquidity measures
- Bounds for Bitcoin's (and other cryptocurrencies') value
- Price discovery in the Bitcoin market

Part II

- Univariate time series models
- Multivariate time series models
- Testing for financial bubbles and explosive price behavior
- Univariate volatility modelling
- Multivariate volatility modelling

Part III

- Market Risk Management
- Portfolio Management
- Credit Risk Management
- Conclusions: challenges ahead

Chapter 1: (Very brief) introduction to Bitcoin and other Cripto-currencies

(Very brief) introduction to Bitcoin and other Cripto-currencies

The Bitcoin network uses cryptography to validate transactions during the payment processing and create transaction blocks. In particular, Bitcoin relies on two cryptographic schemes:

1. *digital signatures* (to exchange the payment instructions between the involved parties)
2. a *cryptographic hash function*: to maintain the discipline when recording transactions to the public ledger (known as *blockchain*)

(Very brief) introduction to Bitcoin and other Cripto-currencies

Digital signatures are used to authenticate digital messages between a sender and a recipient, and they provide:

(I) *Authentication*: the receiver can verify that the message came from sender

(II) *Non-repudiation*: the sender cannot deny having sentthe message;

(III) *Integrity*: the message was not altered in transit.

The use of digital signatures includes public key cryptography, where a pair of keys (open and private) are generated with certain desirable properties.

A digital signature is used for signing messages: the transaction is signed using a private key, and transferred to the Bitcoin network.

All the members of the network can verify that the transaction came from the owner of the public key, by taking the message, the signature, the public key and by running a test algorithm.

(Very brief) introduction to Bitcoin and other Cripto-currencies

A *cryptographic hash function* takes as input a string of arbitrary length (the message m), and returns the string with predetermined length (the hash h).

The function is deterministic, which means that the same input m will always give the same output h. In addition,the function must also have the following properties:

(i) *Pre-image resistance:* for a given hash h, it is difficult to find a message m such that hash(m)=h.

(ii) *Collision resistance*: for a given message m_1 it is hard to find another message m_2, such that hash (m_1) = hash(m_2). In other words, a change in the message leads to a change in the hash.

The output of the hash function looks like to be random, although it is completely deterministic. The Bitcoin network mainly uses the secure hash algorithm SHA-256

(Very brief) introduction to Bitcoin and other Cripto-currencies

From a technical standpoint, bitcoins stay in the Bitcoin network on bitcoin-addresses.

The ownership of a certain number of bitcoins is represented by the ability to send payments via the Bitcoin network using the bitcoins attached to these addresses.

In particular, every bitcoin address is indexed by a unique public ID, which is an alphanumeric identifier, which corresponds to the public key.

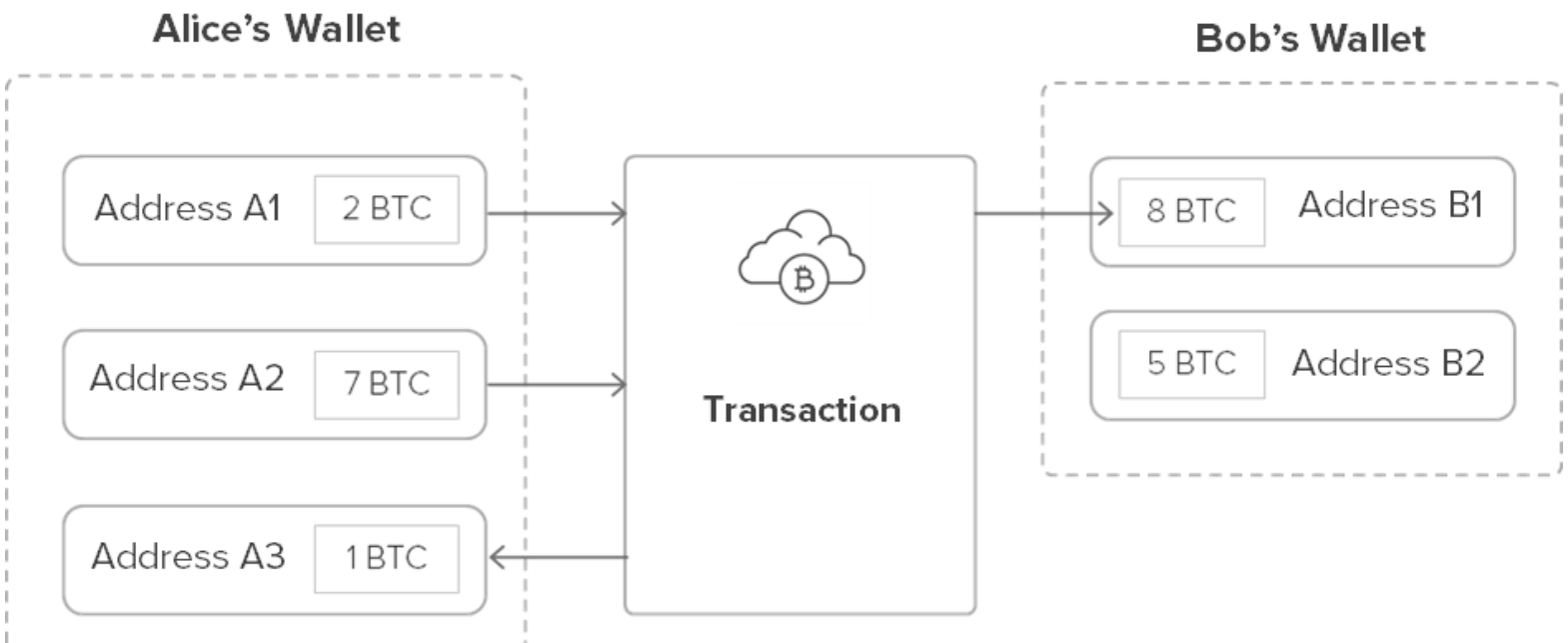
The private key controls the bitcoins stored at that address. Any payment (i.e. a *message*) which involved this address as the sending address must be signed by the corresponding private key to be valid.

$\Rightarrow$ In straight terms,the possession of bitcoins at a specified bitcoin address is given by the knowledge of the private key corresponding to that address.

(Very brief) introduction to Bitcoin and other Cripto-currencies

The agents who process transactions in the Bitcoin network use a set of bitcoin addresses called the *wallet*, which is the set of bitcoin addresses that belong to a single person/entity.

Each transaction record includes one or more sending addresses (inputs) and one or more receiving addresses (outputs), as well as the information about how much each of these addresses sent and received:



(Very brief) introduction to Bitcoin and other Cripto-currencies

After the initial check of the transaction signed messages, validation nodes in the Bitcoin network begin to compete for the opportunity to record a transaction in the blockchain.

1. Competing nodes start putting together transactions in a new block, which were executed since the last record in the blockchain.
2. The block is used to define a complex computing task based on the hash function. The node that first solves this task records the transactions on the blockchain and collects a reward.
3. The implementation of this scheme is the so-called *Hashcash* - a proof that the system is operating properly (proof-of-work), and whose aim is to ensure that the computers use a certain amount of computing power to perform a task (see Beck (2002) for more details).

(Very brief) introduction to Bitcoin and other Cripto-currencies

The nodes that perform the process of the proof-of-work in the Bitcoin network are called *miners*.

These miners use their computing resources in this process with the goal to obtain the reward offered by the Bitcoin Protocol.

Usually the reward is a predetermined number of newly created bitcoins (currently 12.5 BTCs).

The rest of the reward (which is currently smaller), is a voluntary transaction fee paid by those executing the transaction to the miners for transaction processing.

The quantity of BTC issued after the validation of a new block is predetermined and decreases geometrically over time, halving every 210,000 blocks (approximately every 4 years)

(Very brief) introduction to Bitcoin and other Cripto-currencies

After an initial check of the transactions, the *miners* begin to compete to record the new transactions in a new block of the Blockchain. This new block is used as input for a cryptographic hash function to obtain a hash called *digest*.

$\rightarrow$ This digest, together with a one-time random code known as *nounce* -which is an alphanumeric string-, and the hash of the previous block are then used in another hash function to obtain the hash for the new block. The miners have to find a nounce so that the hash of the new Blockchain is numerically smaller than the network's *difficulty* target.

The first miner to solve this computational problem transfers this information to the other nodes in the Bitcoin network, and the Blockchain is updated.

(Very brief) introduction to Bitcoin and other Cripto-currencies

Every 2016 blocks the difficulty target is adjusted to keep the average time between new blocks at 10 minutes, thus automatically adapting to the new total amount of mining power on the network given by the *hashrate* and measured in hash/s.

The number of transactions which can be recorded in a single block is limited by its size (1Mb or about 1500 transactions): given an average time of 10 minutes to generate, a new block this implies a theoretical limit of 7 transactions per second or about 600 thousand transactions per day.

Originally, the Bitcoin network had no block size limit, but this was changed to avoid DOS (Denial Of Service) attacks in the form of large blocks with fake transactions (a slow computer would never catch up in the presence of massive blocks, and its owner would be unable to spend his/her bitcoins).

(Very brief) introduction to Bitcoin and other Cripto-currencies

There is a hot debate in the bitcoin community on how to increase the block size. This first agreed solution was the *Segregated Witness* (SegWit) update, which is a system by which the signature data is separated from other transaction data so that the block size can increase up to 2 MB.

All bitcoin mining pools signaled support for SegWit by 8 August 2017, while SegWit was finally implemented in the network on 21 August 2017.

The SegWit update is an example of a *soft fork*, that is a change of rules which is also recognized by the older software. This is different from a *hard fork*, where the new rules allow to create blocks which are not considered valid by older software.

Finally, visit https://blockchain.info/charts for a quick presentation of the main data related to the Bitcoin blockchain.

Other Cryptocurrencies

The textbook also provides a very brief recap of some of the most interesting alternative cryptocurrencies.

A good starting point for a wider analysis is https://en.wikipedia.org/wiki/List_of_cryptocurrencies, while if the reader wants to see the whole list of available cryptocurrencies, https://coinmarketcap.com and https://coinlib.io/ are the best available sources.

These provides various data, such as price, coin supply, trade volume, or market capitalization (= price $\times$ total supply). Prices are computed by averaging the prices at the major exchanges weighted by volume and are updated every 5 minutes (or less).

News aggregators about cryptocurrencies are *coindesk.com*, *cointelegraph.com* and *cryptopanic.com*, while for a full list see *www.quora.com/What-are-the-best-news-sources-for-cryptocurrency-traders-and-investors*

Chapter 2: Where to get (free) Bitcoin and cryptocurrency data?

Where to get (free) Bitcoin and cryptocurrency data?

There are several cryptocurrency exchanges available nowadays to trade cryptocurrencies for fiat currencies or other cryptocurrencies.

Many of them allow the free download of market data, and there are also many data aggregators which distribute free price data from several exchanges.

An example: Bitcoinity

The website *data.bitcoinity.org* is likely the best place to begin looking at bitcoin market data. It has a well-made and user-friendly interface, and market data can be immediately downloaded as CSV or xlsx files.

For example, if we want to download the latest 30 days of hourly prices for the BTC/EUR pair traded at the Bitstamp exchange with volumes expressed in BTC, we can use the following (alternative) R commands:

Where to get (free) Bitcoin and cryptocurrency data?

- Method 1:

```
url = paste0("http://data.bitcoinity.org/export_data.csv?",
      "currency=EUR&data_type=price_volume&exchange=bitstamp&r=hour",
      "&t=lb&timespan=30d")
x = read.csv(file=url)
head(x)
```

```
##                   Time    price    volume
## 1 2019-05-04 23:00:00 UTC 5161.323 367697.63
## 2 2019-05-05 00:00:00 UTC 5109.705 320224.17
## 3 2019-05-05 01:00:00 UTC 5120.957  14430.50
## 4 2019-05-05 02:00:00 UTC 5130.345 113584.80
## 5 2019-05-05 03:00:00 UTC 5128.678  19751.94
## 6 2019-05-05 04:00:00 UTC 5112.641  45443.53
```

Where to get (free) Bitcoin and cryptocurrency data?

- Method 2:

```
URL <- paste0("http://data.bitcoinity.org/export_data.csv?",
        "currency=EUR&data_type=price_volume&exchange=bitstamp&r=hour",
       "&t=lb&timespan=30d")
download.file(URL, destfile = "data.csv")
x = read.csv(file="data.csv")
head(x)
```

```
##                   Time    price    volume
## 1 2019-05-04 23:00:00 UTC 5161.323 367697.63
## 2 2019-05-05 00:00:00 UTC 5109.705 320224.17
## 3 2019-05-05 01:00:00 UTC 5120.957  14430.50
## 4 2019-05-05 02:00:00 UTC 5130.345 113584.80
## 5 2019-05-05 03:00:00 UTC 5128.678  19751.94
## 6 2019-05-05 04:00:00 UTC 5112.641  45443.53
```

Where to get (free) Bitcoin and cryptocurrency data?

- Method 3:

```
library(RCurl)
URL <- paste0("http://data.bitcoinity.org/export_data.csv?",
       "currency=EUR&data_type=price_volume&exchange=bitstamp&r=hour",
       "&t=lb&timespan=30d")
x <- getURL(URL)
x <- read.csv(textConnection(x))
head(x)
```

```
##                   Time    price    volume
## 1 2019-05-04 23:00:00 UTC 5161.323 367697.63
## 2 2019-05-05 00:00:00 UTC 5109.705 320224.17
## 3 2019-05-05 01:00:00 UTC 5120.957  14430.50
## 4 2019-05-05 02:00:00 UTC 5130.345 113584.80
## 5 2019-05-05 03:00:00 UTC 5128.678  19751.94
## 6 2019-05-05 04:00:00 UTC 5112.641  45443.53
```

Where to get (free) Bitcoin and cryptocurrency data?

The three methods deliver the same data, but the first two may not work properly on platforms different from Windows.

Note that I used the `paste0()` function to divide a long code line over multiple lines to satisfy publication margins. This is not necessary for real-life programming, but it makes the code organized and concise.

Using the first method, I wrote a small function called `bitcoinity_download` which is included in the **bitcoinFinance** package.

Where to get (free) Bitcoin and cryptocurrency data?

```
bitcoinity_download <- function(currency="USD",
                                data_type="price_volume",
                                exchange="bitstamp",
                                freq="hour",
                                time_length="30d",
                                sd=NULL){
  baseurl = "http://data.bitcoinity.org/export_data.csv?"
  if(is.null(exchange)){
    url=paste0(baseurl,"currency=",currency,"&data_type=",data_type,
               "&r=", freq, "&timespan=",time_length)
  }
  if(is.null(exchange)&!is.null(sd)){
    url=paste0(baseurl,"currency=",currency,"&data_type=",data_type,
               "&r=", freq, "&timespan=",time_length, "&sd=", sd)
  }
  if(!is.null(exchange)){
  url=paste0(baseurl,"currency=",currency,"&data_type=",data_type,
             "&exchange=",exchange,"&r=", freq, "&timespan=",time_length)
  }
  if(!is.null(exchange)&!is.null(sd)){
    url=paste0(baseurl,"currency=",currency,"&data_type=",data_type,
    "&exchange=",exchange,"&r=", freq, "&timespan=",time_length, "&sd=", sd)
  }
  x = read.csv(file=url)
  return(x)
}
```

Where to get (free) Bitcoin and cryptocurrency data?

An example is reported below:

```
dat<-bitcoinity_download(currency="EUR")
head(dat)
```

```
##                  Time                price   volume
1 2018-08-18 12:00:00 UTC 5693.920 169829.5
2 2018-08-18 13:00:00 UTC 5574.136 1307744.4
3 2018-08-18 14:00:00 UTC 5568.665 800261.7
4 2018-08-18 15:00:00 UTC 5546.452 242329.5
5 2018-08-18 16:00:00 UTC 5542.700 159812.0
6 2018-08-18 17:00:00 UTC 5541.787 129611.9
```

Chapter 3: Liquidity measures

Liquidity measures

Liquidity is notoriously tricky to define and to measure, see e.g. Goodhart (2008) and Persaud (2006). This is why scholars define it more often as a set of features rather than as a unidimensional concept (Nikolaou (2009)).

It has so many facets that, according to Goodhart (2008) "it is often counter-productive to use it without further and closer definition".

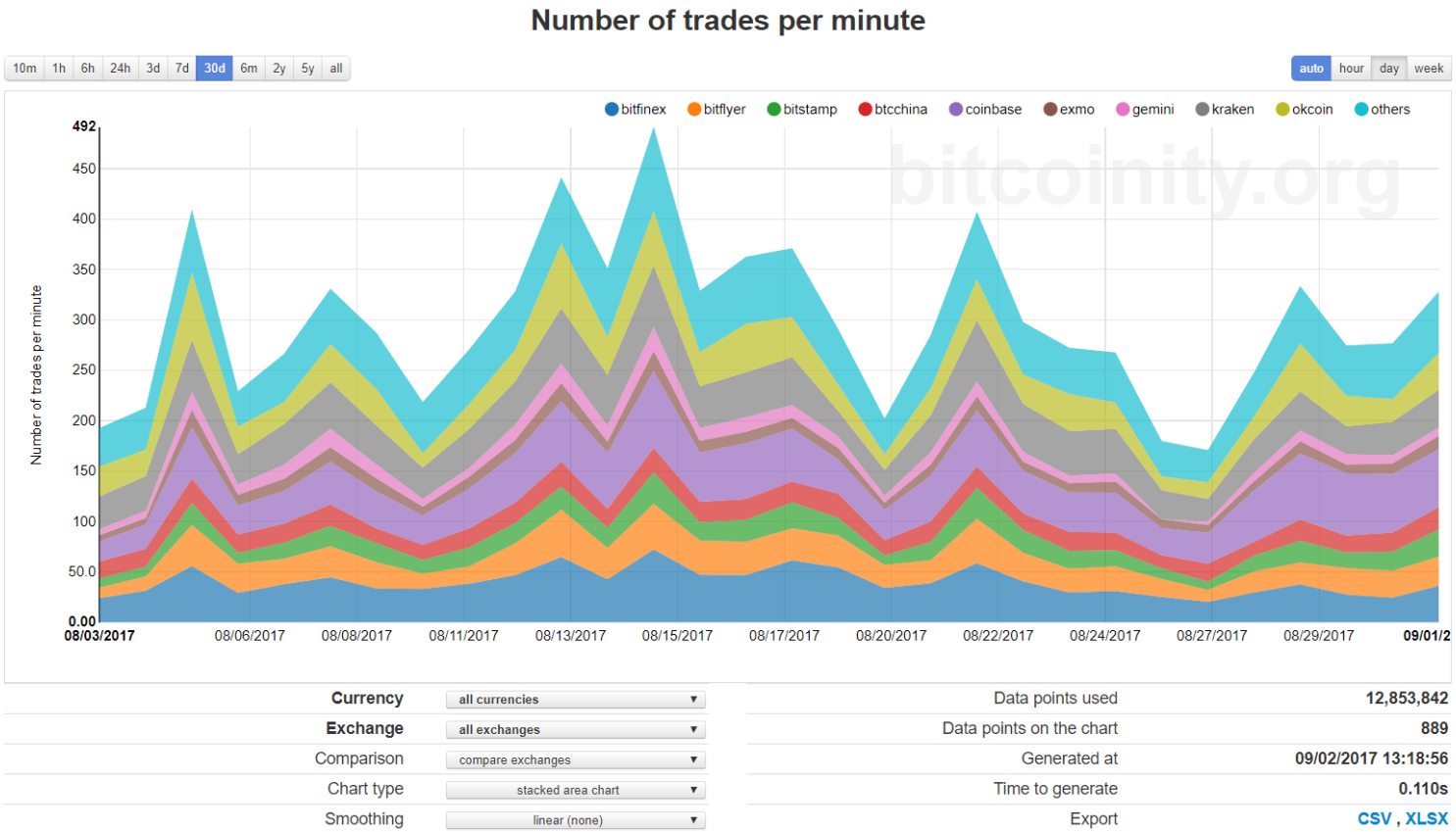
The textbook in chapter 3 provides a review of the main liquidity measures and shows several examples with R:

- Volume-related Liquidity Measures
- Time-related Liquidity Measures
- Spread-related Liquidity Measures
- Multi-dimensional Liquidity Measures

A small example: liquidity measures from Bitcoinity

We already met the interesting website https://data.bitcoinity.org which is probably one of the most user-friendly websites dedicated to Bitcoin exchanges. We now focus on one liquidity measure which is freely available on this website.

One available measure is the *average number of trades per minute* N_t, which was the first time-related measure we saw during the previous theoretical review, see the Figure below for an example.



A small example: liquidity measures from Bitcoinity

This measure can be computed with an hourly, daily and weekly data frequency: hourly data can be downloaded for the last 30 days, while daily and weekly data are available for the whole time span possible for each exchange.

To download the last 30 days of the hourly average number of trades per minute for BTC/EUR at Bitspamp, we can use the function `bitcoinity_download()` from the **bitcoinFinance** package:

```
dat<-bitcoinity_download(currency="EUR", data_type="tradespm",
                         exchange="bitstamp")
head(dat)
                    Time         X
1 2017-08-03 22:00:00 UTC 2.0166667
2 2017-08-03 23:00:00 UTC 2.4500000
3 2017-08-04 00:00:00 UTC 1.1333333
4 2017-08-04 01:00:00 UTC 1.1166667
5 2017-08-04 02:00:00 UTC 0.1333333
6 2017-08-04 03:00:00 UTC 0.1000000
```

Chapter 4: What is bitcoin fundamental value? A review of financial and economic approaches

A long-term upper bound: *Market Sizing*

Market sizing is basically the process of estimating the potential of a market and this is widely used by companies which intend to launch a new product or service.

Woo et al. (2013) in a Bank of America Merrill Lynch report estimated separately the value of bitcoin as a A) *medium of exchange* and as B) *store of value* and then summed them up to get a rough estimate of bitcoin fair value.

This method is fully discussed in the textbook and implemented in the `bitcoinFinance` package. Note that many methods proposed in recent years are basically variants of this approach.

A short-term lower bound: *the marginal cost of bitcoin production*

Garcia et al. (2014) were the first to suggest that the fundamental value of one bitcoin should be at least equal to the cost of the energy involved in its production through mining.

$\Rightarrow$ lower bound estimate of bitcoin fundamental value.

More recently, a more refined model for the cost of bitcoin production was developed by Hayes (2015a,b). Variables to consider:

1) the cost of electricity, measured in cents per kilowatt-hour;
2) the energy consumption per unit of mining effort, measured in watts per GH/s (1 W/GH/s=1 Joule/GH);
3) the bitcoin market price;
4) the difficulty of the bitcoin algorithm;
5) the block reward (currently 12,5 BTC), which halves approx. every 4 years

A lower bound: *the marginal cost of bitcoin production*

In a competitive commodity market, an agent would undertake mining if the marginal cost per day (electricity consumption) were less than or equal to the marginal product (the number of bitcoins found per day on average multiplied by the dollar price of bitcoin).

Hayes (2015a,b) develops his model by assuming that a miner's daily production of bitcoin depends on its own rate of return, measured in expected bitcoins per day per unit of mining power.

The expected number of bitcoins expected to be produced per day can be calculated as follows:

$$BTC/day^{*} = [(\beta \cdot \rho)/(\delta \cdot 2^{32})] \cdot sec_{hr} \cdot hr_{day} \qquad (1)$$

where β is the block reward (currently 12,5 BTC/block), ρ is the hashing power employed by a miner, and δ is the difficulty (which is expressed in units of GH/block).

A lower bound: *the marginal cost of bitcoin production*

The constant sec_{hr} is the number of seconds in an hour (3600), while hr_{day} is the number of hours in a day (24).

The constant 2^{32} relates to the normalized probability of a single hash per second solving a block, and is a feature of the 256-bit encryption at the core of the SHA-256 algorithm.

These constants which normalize the dimensional space for daily time and for the mining algorithm can be summarized by the variable θ, given by $\theta = 24hr_{day} \cdot 3600/2^{32}sec_{hr} = 0.00002011656761169 43$. Equation (1) can thus be rewritten compactly as follows:

$$BTC/day^{*} = \theta \cdot (\beta \cdot \rho)/\delta \tag{2}$$

Hayes (2015a,b) sets $\rho = 1000$ GH/s even though the actual hashing power of a miner is likely to deviate greatly from this value. However, Hayes (2015a,b) argues that this level tends to be a good standard of measure.

A lower bound: *the marginal cost of bitcoin production*

The cost of mining per day, E_{day} can be expressed as follows:

$$E_{day} = (\text{price per kWh} \cdot 24\, hr_{day} \cdot \text{W per GH/s})(\rho/1000\, GH/s) \quad (3)$$

Assuming that the bitcoin market is a competitive market, the marginal product of mining should be equal to its marginal cost, so that the \$/BTC (equilibrium) price level is given by the ratio of (cost/day) / (BTC/day):

$$p^* = E_{day}/(BTC/day^*) \quad (4)$$

$\Rightarrow$ This price level can be though as a price lower bound, below which a miner would operate at a marginal loss and would probably stop mining.

A lower bound: *the marginal cost of bitcoin production*

Example: use the world average electricity cost ≈ 13.5 cents/KWh, the average energy efficiency of bitcoin mining hardware$\approx$ 0.25J/GH

$\Rightarrow$ the average cost per day for a 1000 GH/s mining rig is:

$$\begin{aligned} E_{day} &= (\text{price per kWh} \cdot 24\, hr_{day} \cdot \text{W per GH/s})(\rho/1000\, GH/s) \\ &= (0.135 \cdot 24 \cdot 0.25) \cdot (1,000/1,000) = 0.81\$/day \end{aligned}$$

The number of bitcoins that a 1000 GH/s of mining power can find in a day with a current difficulty of 2227847638504 is equal to

$$\begin{aligned} BTC/day^* &= \theta \cdot (\beta \cdot \rho)/\delta = \\ &= 0.0000201165676116943 \cdot (12,5 \cdot 1\mathrm{e}^{12})/2227847638504 \\ &= 0.000112869969561757\, BTC/day. \end{aligned}$$

The \$/BTC price is given by equation (4):

$$\begin{aligned} p^* &= E_{day}/(BTC/day^*) = \\ &= (0.81\$/day)/(0.000112869969561757 BTC/day) \\ &\approx 7176.40\, \$/BTC \end{aligned}$$

A lower bound: *the marginal cost of bitcoin production*

We can compute the bitcoin lower bound with inputs given by the user using the function `btc.lower.bound.user()` from the **bitcoinFinance** package:

```
btc.lower.bound.user=function(block.reward = 12.5,
  hashing.power.miner = 10^12, Difficulty = 559970892890,
  price.kWh=0.125, W.GHs=0.25){
  theta<-(24*3600)/(2^(32))
  BTCday  <- theta*block.reward*hashing.power.miner/Difficulty
  cost.mining.day <- price.kWh*24*W.GHs/(hashing.power.miner/10^12)
  price.bitcoin <- cost.mining.day/BTCday
  return(price.bitcoin)
}
```

As an example, I replicate the example in Fantazzini et al. (2016):

```
btc.lower.bound.user(Difficulty=60883825480, price.kWh=0.135,
                     W.GHs=0.75)
[1] 588.3616
```

Chapter 5: Bitcoin Market Price Discovery

Bitcoin Market Price Discovery

Brandvold et al. (2015) are the first (and so far the only ones) to study the price discovery process in the Bitcoin market, which consists of several independent exchanges.

This topic is frequently discussed in the bitcoin community because knowing which exchange reacts most quickly to new information (thus reflecting the value of Bitcoin most precisely), is clearly of outmost importance for both short-term traders and long-term investors.

The price discovery literature employs mainly three methodologies:

- the information share method by Hasbrouck (1995),
- the permanent-transitory decomposition by Gonzalo and Granger (1995)
- the structural multivariate time series model by de Jong et al. (2001) which is an extension of Harvey (1989).

Bitcoin Market Price Discovery

Brandvold et al. (2015) used the method by de Jong et al. (2001) because

- it has the advantage that the information share is uniquely defined, unlike the information share computed with the Hasbrouck's (1995) model,
- and it takes the variance of innovations into account, unlike Gonzalo and Granger (1995), so that a price series with low innovation variance gets a low information share.

This multivariate model by de Jong et al. (2001) was proposed to estimate the information share of various exchanges with respect to the information generated by the whole market.

Bitcoin Market Price Discovery

$\Rightarrow$ The prices are composed of two components, one common (unobserved) underlying random walk and an idiosyncratic specific noise for each exchange.

$\Rightarrow$ The random walk component is interchangeably referred to either as the efficient price or the fundamental news component.

$\Rightarrow$ It follows immediately from this model structure that the exchanges' prices are cointegrated by construction, while the idiosyncratic component can be due to specific conditions at an exchange, traders' strategic behaviour, or other shocks.

The theoretical setup in Brandvold et al. (2015) assumes n individual exchanges and m corresponding markets, with $m = n$, whereas a market for an exchange is defined as all the other exchanges combined.

Bitcoin Market Price Discovery: an R example

Brandvold et al. (2015) used data from seven exchanges: Bitfinex, Bitstamp, BTC-e (Btce), BTC China(Btcn) and Mt.Gox (Mtgox), Bitcurex and Canadian Virtual Exchange (Virtex). Data covered the period April 1st 2013–February 25th 2014, till bankruptcy of Mtgox.

They found that the two exchanges with positive ψ for the entire period were Btce and Mtgox, thus indicating that these exchanges were more informative than their competitors.

Similar evidence was provided by the information share, which was highest for Btce and Mtgox (0,322 and 0.366, respectively).

$\Rightarrow$ Information shares change over time: for example, the information share of Btcn first increased from 0.040 in April 2013 to 0.325 in December 2013 because some large Chinese companies (like Baidu) started accepting Bitcoin as payment, but then its information share fell to 0.124 in January 2014 after the Chinese government banned payment companies from clearing Bitcoin.

Bitcoin Market Price Discovery: an R example

$\Rightarrow$ An empirical example with R using the function `information_shares()` from the **bitcoinfinance** package.

I show an example using the bitcoin prices from five exchanges covering the time sample [2016-10-20/2017-04-20]: Bitstamp, Itbit, Gdax, Kraken, and Localbitcoins.

The latter is not formally an exchange, but an online service which facilitates over-the-counter trading of local currency for bitcoins, that is it gives the opportunity to a buyer and a seller to conduct direct transactions.

Bitcoin Market Price Discovery: an R example

```
data_file<-system.file("extdata", "btcusd_IS.csv", package = "bitcoinFinance")
dat<-read.csv(file = data_file,header = TRUE,sep = ";",dec = ".")

# Vector of activity shares based on trading volumes and trades frequency
pivector<-c(0.33,0.06,0.48,0.11,0.02)
bitcoinFinance::information_shares(dat,pi=pivector, opt_method="nlminb")

  Information shares PSI_coefficients
1         0.24368444    -8.440675e-04
2         0.05667536    -1.788117e-04
3         0.42498757    -3.698466e-04
4         0.11338134     9.919682e-05
5         0.16127130     2.279428e-02

# Robustness check: vector of activity shares set to 1/n for all five exchanges
n<-ncol(dat)-1
pivector<-c(rep(1/n,n))
bitcoinFinance::information_shares(dat,pi=pivector, opt_method="nlminb")

  Information shares PSI_coefficients
1          0.1028536    -0.0015674684
2          0.1074254    -0.0014937014
3          0.1411473    -0.0009495954
4          0.1244755    -0.0012185958
5          0.5240982     0.0052293611
```

Chapter 6: Univariate time series models

Univariate time series models: ARIMA models

A time series Y_t is called *integrated* to the d^{th} degree, if it has to be differenced d-times to become covariance stationary.

In short we write $Y_t \sim I(d)$ then its d^{th} difference $\Delta^d Y_t \equiv (1-L)^d Y_t$ is stationary. $(1-L)^d$ is a lag polynomial of order d. Since the d roots of this polynomial are one, an integrated process is called a unit root process.

In general, a time series Y_t is called an *AutoRegressive Integrated Moving Average (ARIMA)* process of order (p, d, q) if it is of the following form,

$$\Phi_p(L)(1-L)^d Y_t = c + \Theta_q(L)\varepsilon_t,$$

where $(1-L)^d Y_t$, that is the d-th difference of Y_t, is itself an $ARMA(p, q)$-process.

Univariate time series models: ARIMA models

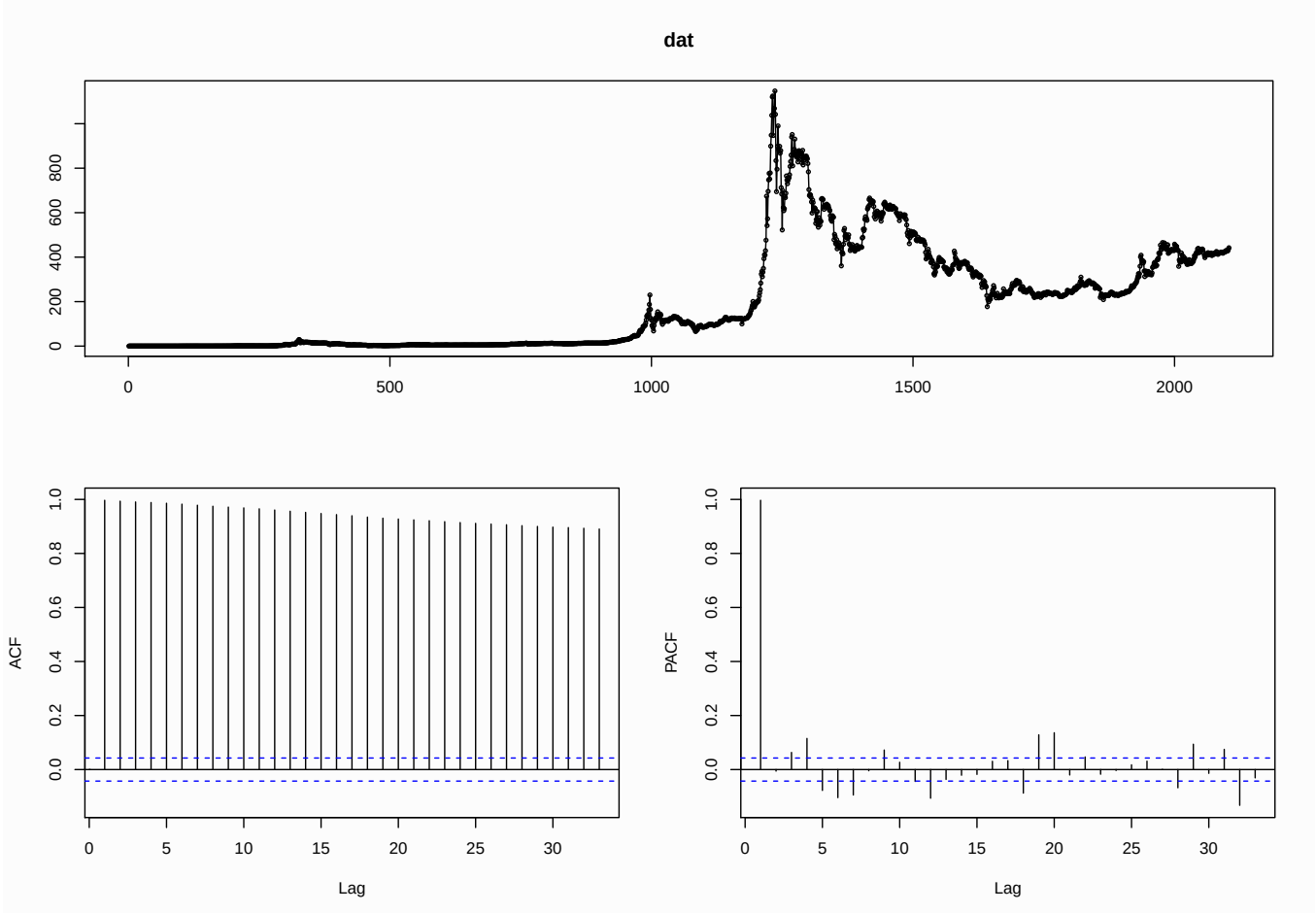
One classical methodology proposed to select the best ARIMA model for a specific dataset is the *Box and Jenkins modelling philosophy* which consists of the following steps (see Box et al. (2015) for more details):

- Model identification,
- Model estimation,
- (In-sample) Model evaluation, and re-specification of the model if necessary,
- Forecasting and (out-of-sample) model evaluation.

Univariate time series models: ARIMA models

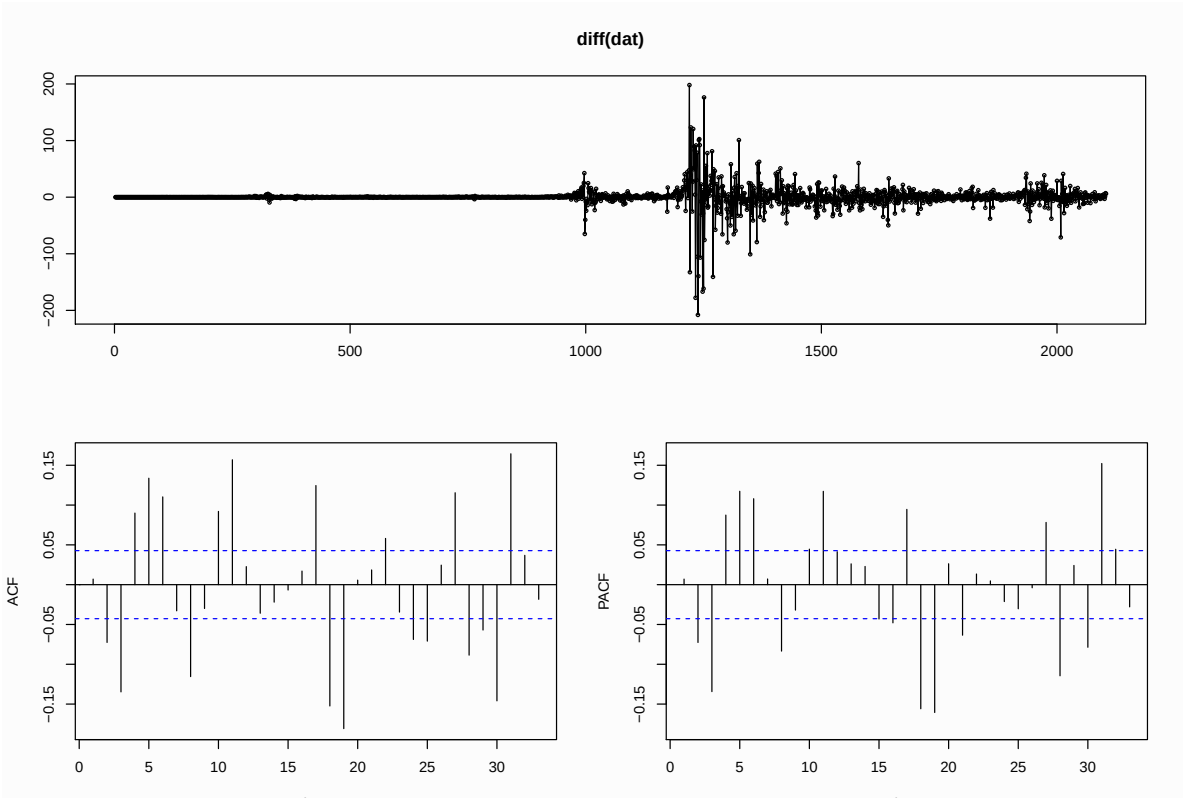
Let's use this algorithm to our bitcoin historical price series:

```
path.bit<-system.file("extdata","coindesk-bpi-USD-close.csv",package="bubble")
dat <- read.table(path.bit, dec = ".", sep =",", header = TRUE)
dat <- xts::xts(dat[,2], order.by=as.Date(dat[,1]))
forecast::tsdisplay(dat)
```



Univariate time series models: ARIMA models

```
> tseries::adf.test(dat)
    Augmented Dickey-Fuller Test
data:  dat
Dickey-Fuller = -2.9349, Lag order = 12, p-value = 0.1825
alternative hypothesis: stationary
> tseries::kpss.test(dat)
    KPSS Test for Level Stationarity
data:  dat
KPSS Level = 10.477, Truncation lag parameter = 10, p-value = 0.01
> tseries::kpss.test(diff(dat))
    KPSS Test for Level Stationarity
data:  diff(dat)
KPSS Level = 0.029084, Truncation lag parameter = 10, p-value = 0.1
```



Univariate time series models: ARIMA models

Bitcoin prices are clearly not stationary, whereas their first difference definitely is.

The next step is to find the best ARIMA model using the Hyndman and Khandakar (2008) algorithm: to decrease the computational burden, I consider a max of 60 lags for both the autoregressive and moving parts, I avoid using any seasonal ARIMA, I use log-prices to stabilize the variance and improve the model fit, and parallel computation is employed.

```
fit <- forecast::auto.arima(log(dat), max.p = 60, max.q = 60,
        stepwise = FALSE,seasonal = FALSE, parallel = TRUE, num.cores= 8)
fit
Series: log(dat)
ARIMA(2,2,3)
Coefficients:
          ar1      ar2     ma1      ma2      ma3
      -1.6232  -0.7789  0.6831  -0.8270  -0.8196
s.e.   0.1431   0.1272  0.1306   0.0361   0.1128

sigma^2 estimated as 0.00385:  log likelihood=2861.28
AIC=-5710.57   AICc=-5710.53   BIC=-5676.66

forecast::Acf(residuals(fit), lag.max = 200)
```

Univariate time series models: ARIMA models

```
>Box.test(residuals(fit), fitdf=35, lag=200, type="Ljung")
Box-Ljung test
data:  residuals(fit)
X-squared = 260.56, df = 195, p-value = 0.001184

exp(as.data.frame(forecast::forecast(fit, h=5)))
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2105       442.7872 408.9425 479.4329 392.0863 500.0442
2106       443.5516 395.0247 498.0398 371.5237 529.5437
2107       444.2546 385.3125 512.2134 357.3453 552.3010
2108       444.9573 377.0823 525.0499 345.4491 573.1294
2109       445.7125 369.9175 537.0377 335.1602 592.7303
```

It seems that long-range seasonal and non-seasonal dependence and (several) structural breaks remain to be modelled: we leave this task to the interested reader as an (hopefully) interesting exercise.

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

The general structure of this framework is to decompose a time series Y into three components:

- a trend (T) (= the long-term component of Y),
- a seasonal pattern (S),
- and an error term (E).

These components can enter the model specification as,

- additive terms (for example $Y = T + S + E$),
- multiplicative (for example $Y = T \cdot S \cdot E$)
- or both (for example, $Y = (T \cdot S) + E$).

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

Moreover, the trend component can be decomposed into a level term (l) and a growth term (b) and it can be "dampened" by using an additional parameter $0 < \phi < 1$, so that five different trend types are possible:

- None: $T_h = l$
- Additive: $T_h = l + bh$
- Additive damped: $T_h = l + b\phi_h$
- Multiplicative: $T_h = l \cdot b^h$
- Multiplicative damped: $T_h = l \cdot b^{\phi_h}$

where T_h is the the trend forecast h periods out, while $\phi_h = \sum_{s=1}^{h} \phi^s$, see Hyndman et al. (2008).

Therefore, the single components of an ETS model can have these specifications (for a total of $30 = 5 \cdot 3 \cdot 2$ possible ETS models):

Error		Addittive (A)	Multiplicative (M)		
Trend	None (N)	Addittive (A)	Multiplicative (M)	Addittive Damped	Multiplicative Damped
Seasonal	None (N)	Addittive (A)	Multiplicative (M)		

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

The detailed equations for all the 30 possible ETS models are reported in Hyndman et al (2008) - Tables 2.2 and 2.3, p. 21-22, and I refer the interested reader to that textbbok for more details. Some examples are below:

- ETS(A, N, N): this is the simple exponential smoothing model, where the current value of the level term $l_t = \alpha y_t + (1-\alpha)\hat{y}_t$ is the weighted average of y_t and its forecasted value $\hat{y}_t$, with $\hat{y}_t = l_{t-1}$. By straightforward substitution, the full model can be written according to the following state space specification:

$$\begin{aligned} \text{Observation equation:} \quad y_t &= l_{t-1} + \epsilon_t \\ \text{State equation:} \quad l_t &= l_{t-1} + \alpha\epsilon_t \end{aligned}$$

- ETS(A,A,A): this is the additive Holt-Winters method with additive errors. Its state space representation is given below:

$$\begin{aligned} \text{Observation equation:} \quad y_t &= l_{t-1} + b_{t-1} + s_{t-m} + \epsilon_t \\ \text{State equations:} \quad l_t &= l_{t-1} + b_{t-1} + \alpha\epsilon_t \\ b_t &= b_{t-1} + \beta\epsilon_t \\ s_t &= s_{t-m} + \gamma\epsilon_t \end{aligned}$$

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

Hyndman et al. (2002, IJF) suggest to use the following automatic forecasting procedure:

- Apply each of the 30 models that are appropriate to the dataset, and estimate parameters and initial values using maximum likelihood or other methods;
- Select the best model using AIC (or other criteria);
- Produce forecasts using best method;
- Obtain prediction intervals using underlying state space model

We are now ready to see an example in R, using the `ets()` function from the **forecast** package. This function automatically chooses the best ETS model according to the AIC, AIC_c or BIC and produce forecast intervals for every model.Let's see it in more details:

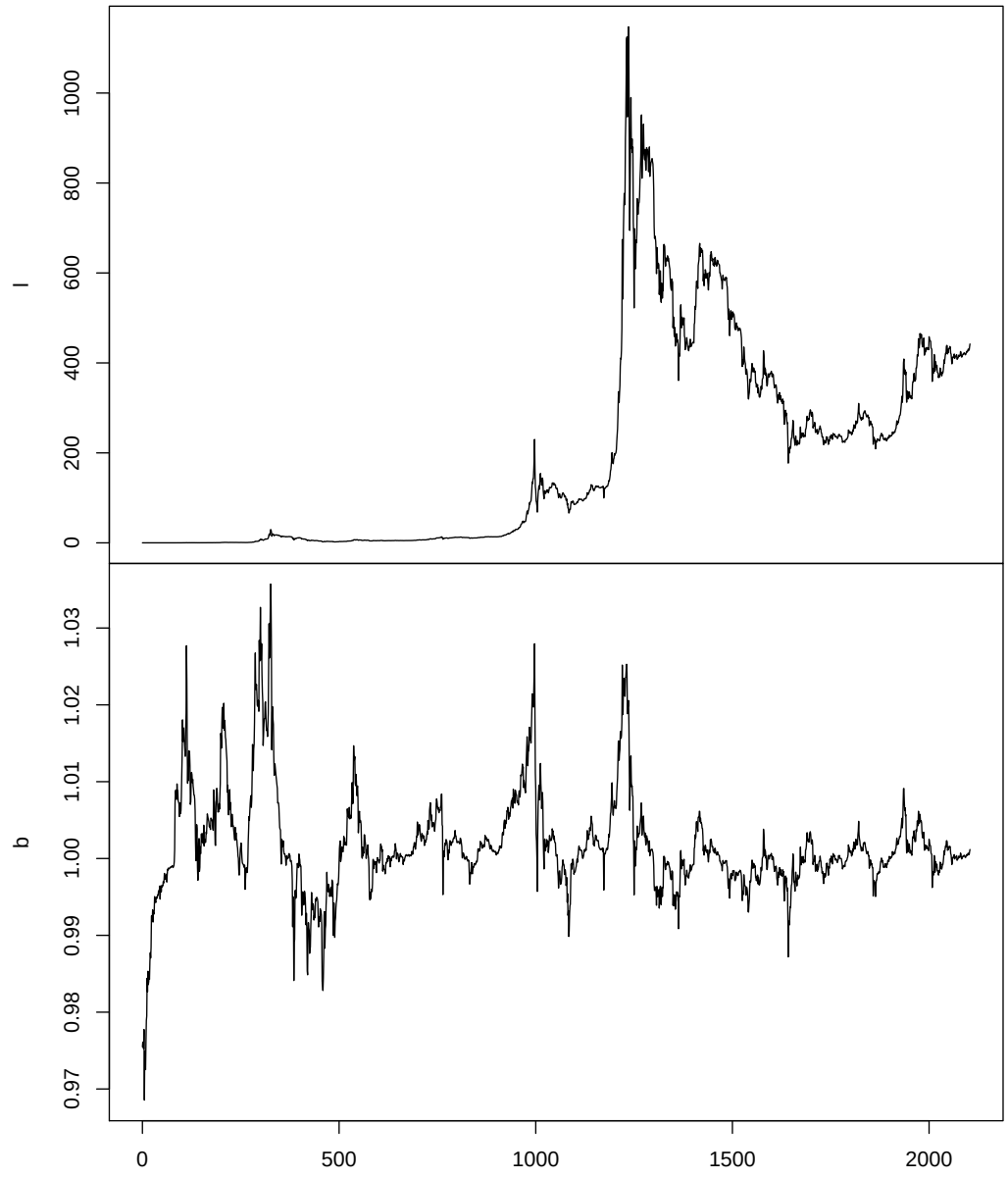
```
ets(y, model = "ZZZ", damped = NULL, alpha = NULL, beta = NULL,
    gamma = NULL, phi = NULL, additive.only = FALSE, lambda = NULL,
    biasadj = FALSE, lower = c(rep(1e-04, 3), 0.8), upper = c(rep(0.9999,
        3), 0.98), opt.crit = c("lik", "amse", "mse", "sigma",
        "mae"), nmse = 3, bounds = c("both", "usual", "admissible"),
    ic = c("aicc", "aic", "bic"), restrict=TRUE,allow.multiplicative.trend=FALS
    use.initial.values = FALSE, ...)
```

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

An example with our log-transformed bitcoin data is reported below:

```
path.bit <- system.file("extdata","coindesk-bpi-USD-close.csv",package="bubble"
dat <- read.table(path.bit, dec = ".", sep =",", header = TRUE)
dat <- xts::xts(dat[,2], order.by=as.Date(dat[,1]))
#Choose the optimal ETS model
fit.ets <- forecast::ets(log(dat))
fit.ets

ETS(A,Ad,N)
Call:
 forecast::ets(y = log(dat))
  Smoothing parameters:
    alpha = 0.9999
    beta  = 0.0234
    phi   = 0.9745
  Initial states:
    l = -2.4312
    b = -0.0248
sigma:  0.062
     AIC     AICc      BIC
4407.144 4407.184 4441.054

#Plot the ETS model states
plot(exp(fit.ets$states), main="")
```

ETS (Error-Trend-Seasonal or ExponenTial Smoothing)



ETS (Error-Trend-Seasonal or ExponenTial Smoothing)

```
# Forecast 3-step ahead with confidence intervals
exp(as.data.frame(forecast::forecast(fit.ets, h=3)))
     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
2105       442.6608 408.8725 479.2412 392.0430 499.8140
2106       443.1538 395.5769 496.4528 372.4952 527.2157
2107       443.6348 385.4269 510.6334 357.7720 550.1041
```

The `ets()` function also allows refitting model to a new dataset, which can be handy if accuracy measures have to be computed:

```
fit.ets1 <- forecast::ets(log(dat[1:2000]))
forecast::accuracy( exp(forecast(fit.ets1, h=10)$mean) , dat[2001:2010])
                ME     RMSE      MAE       MPE     MAPE
Test set -40.66515 53.23313 40.66515 -10.36078 10.36078
```

where `accuracy` is a function which computes a range of summary measures of the forecast accuracy.

Bayesian structural time series models

The Bayesian structural time series (BSTS) approach proposed by Scott and Varian (2014) and Scott and Varian (2015) is an integrated system which combines three methods: a structural model for trend and seasonality estimated with a Kalman filter, a "spike and slab" regression for variable regression, and Bayesian model averaging to obtain the final forecast.

A structural time series model is composed of two equations:

$$\begin{aligned} \text{Observation equation:} \quad y_t &= Z_t'\alpha_t + \epsilon_t \quad \epsilon_t \sim N(0, H_t) \\ \text{Transition equation:} \quad \alpha_t &= T_t\alpha_{t-1} + R_t\eta_t \quad \eta_t \sim N(0, Q_t) \end{aligned}$$

where y_t is the observed data at time t, α is a vector of latent variables (the so-called "state"), Z_t, H_t, T_t , R_t and Q_t are structural parameters (some of which are known).

$\Rightarrow$ The observation equation links the observed data y_t to the unobserved state α_t, while the transition equation models the dynamics of the latent state.

Bayesian structural time series models

A basic structural model with a trend, a seasonal pattern τ_t and a regression component $\beta' \boldsymbol{x}_t$ can be written as follows:

$$
\begin{aligned}
y_t &= \mu_t + \tau_t + \beta' \boldsymbol{x}_t + \epsilon_t \\
\mu_t &= \mu_{t-1} + \delta_{t-1} + u_t \\
\delta_t &= \delta_{t-1} + \nu_t \\
\tau_t &= -\sum_{s=1}^{S-1} \tau_{t-s} + \omega_t
\end{aligned}
$$

where $\eta_t = (u_t, \nu_t, \omega_t)$ consists of independent normally distributed white noise processes, Q_t is a diagonal matrix with constant elements $\sigma_u^2, \sigma_\nu^2, \sigma_\omega^2$, and H_t is a constant scalar σ_ϵ^2.

μ_t can be interpreted as the current "level" of the (local) linear trend, while the current "slope" of the trend is represented by δ_t.

The seasonal component is modelled with S dummy variables with time varying coefficients, whose sum has zero expectation.

Bayesian structural time series models

The simplest structural time series model is the *local level model*:

$$\begin{aligned} y_t &= \mu_t + \epsilon_t \quad \epsilon_t \sim N(0, \sigma^2) \\ \mu_t &= \mu_{t-1} + \eta_t \quad \eta_t \sim N(0, \tau^2) \end{aligned}$$

If $\sigma^2 = 0$ we have the random walk model, while $\tau^2 = 0$ gives the constant mean model. The higher the ratio σ^2/τ^2 the closer the model is to the constant mean model. This model can be expressed in state space form with $T_t = 1$, $Z_t = 1$, $R_t = 1$, $H_t = \sigma^2$, $Q_t = \tau^2$.

```
path.bit<-system.file("extdata","coindesk-bpi-USD-close.csv",package="bubble")
dat <- read.table(path.bit, dec = ".", sep =",", header = TRUE)
dat <- xts::xts(dat[,2], order.by=as.Date(dat[,1]))
ss <- bsts::AddLocalLevel(list(), dat)
model <- bsts::bsts(dat, state.specification = ss, niter = 500)
#Forecasts and confidence interval
pred <- predict(model, horizon = 5, burn = 100)
cbind(pred$mean,t(pred$interval))
                    2.5%    97.5%
 [1,] 443.9637 411.2513 479.2376
 [2,] 444.5837 401.9589 491.8797
 [3,] 445.9039 392.7868 501.5097
 [4,] 446.1968 384.8065 508.2831
 [5,] 446.5179 377.0805 511.4161
```

Bayesian structural time series models

The *local linear trend model* can be useful if the time series is trending in a certain direction and forecasts should reflect this trend observed in recent observations:

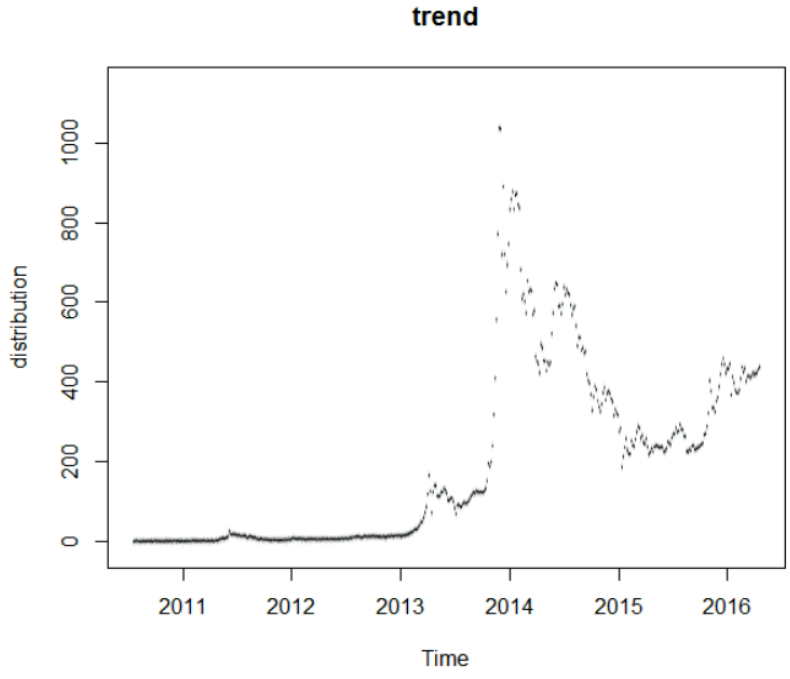
$$
\begin{aligned}
y_t &= \mu_t + \epsilon_t \quad \epsilon_t \sim N(0, \sigma^2) \\
\mu_t &= \mu_{t-1} + \delta_{t-1} + \eta_{\mu,t} \quad \eta_{\mu,t} \sim N(0, \tau_\mu^2) \\
\delta_t &= \delta_{t-1} + \eta_{\delta,t} \quad \eta_{\delta,t} \sim N(0, \tau_\delta^2)
\end{aligned}
$$

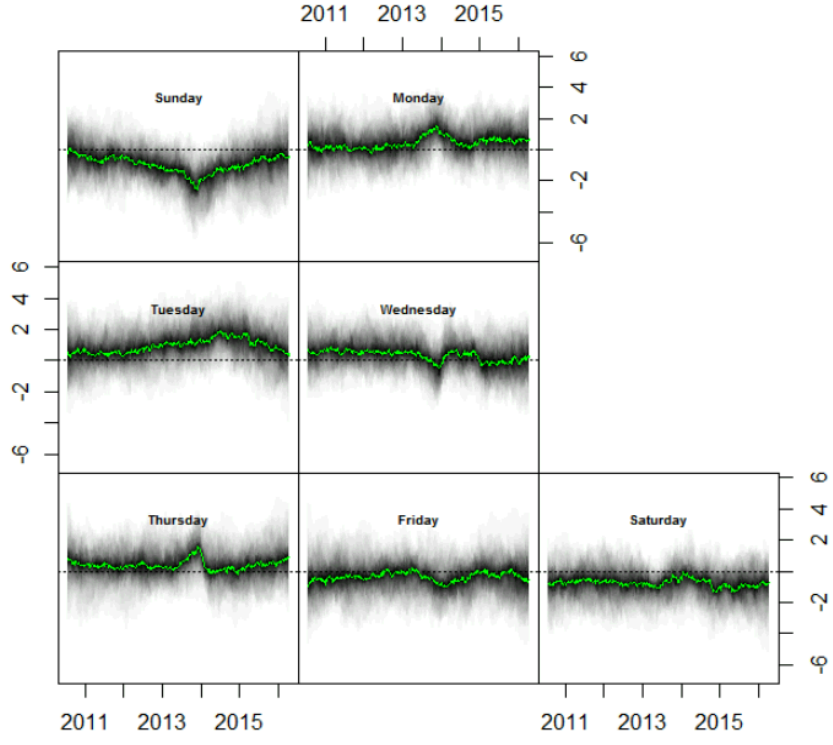
The local linear trend model can be used to find short term changes in the trend, but it forget the past rather quickly by construction and it is extremely volatile.

Moreover, if necessary, a seasonal component $\tau_t = -\sum_{s=1}^{S-1} \tau_{t-s} + \eta_{\omega,t}$ with $\eta_{\omega,t} \sim N(0, \tau_\omega^2)$ can be added.

Bayesian structural time series models

```
path.bit<-system.file("extdata","coindesk-bpi-USD-close.csv",package="bubble")
dat <- read.table(path.bit, dec = ".", sep =",", header = TRUE)
dat <- xts::xts(dat[,2], order.by=as.Date(dat[,1]))
ss <- bsts::AddLocalLinearTrend(list(), dat)
# A weekly seasonal component is added
ss<-bsts::AddSeasonal(ss,dat,nseasons = 7)
model <- bsts::bsts(dat, state.specification = ss, niter = 500)
plot(model, "comp", same.scale = FALSE) ## See the model "components"
```





Bayesian structural time series models

```
#Forecasts and confidence interval
pred <- predict(model, horizon = 5, burn = 100)
cbind(pred$mean,t(pred$interval))
```

```
                  2.5%    97.5%
[1,] 443.8838 407.8434 480.3156
[2,] 442.7382 396.2476 488.5442
[3,] 444.3841 388.4584 501.1000
[4,] 446.8170 382.3488 513.6280
[5,] 448.4238 370.3549 525.2556
```

As expected, the seasonal component of the bitcoin price is estremely small compared to the local trend component.

Chapter 7: Multivariate time series models

Modelling bitcoin price dynamics: VAR & VECM

Most macro-financial analyses devoted to bitcoin prices employ:

1) Vecto-AutoRegression (VAR) models,

$$\Delta \mathbf{Y}_{t-1} = \alpha + \Phi_1 \Delta \mathbf{Y}_{t-1} + \Phi_2 \Delta \mathbf{Y}_{t-2} + ... + \Phi_p \Delta \mathbf{Y}_{t-p} + \varepsilon_t \quad (5)$$

2) Vector Error Correction (VEC) models,

$$\Delta \mathbf{Y}_{t-1} = \alpha + \mathbf{B}\Gamma \mathbf{Y}_{t-1} + \zeta_1 \Delta \mathbf{Y}_{t-1} + \zeta_2 \Delta \mathbf{Y}_{t-2} + ... + \zeta_{p-1} \Delta \mathbf{Y}_{t-(p-1)} + \varepsilon_t \quad (6)$$

where $\mathbf{B}$ are the factor loadings, while Γ the cointegrating vector.

Kristoufek (2013) is the first author to propose a multivariate approach: hefound a significant bidirectional relationship, where Google trends search queries influence prices and viceversa, suggesting that speculation and trend chasing dominate the bitcoin price dynamics.

Modelling bitcoin price dynamics: VAR & VECM

Glaser et al. (2014) extended previous research by studying the aggregated behavior of new and uninformed Bitcoin users within the time span from 2011 to 2013, to identify why people gather information about Bitcoin and their motivation to subsequently participate in the Bitcoin system.

The main novelty is the use of regressors that are related to both bitcoin **attractiveness** and bitcoin **supply and demand**:

- *daily BTC price data*,
- *daily exchange volumes* in BTC,
- *Bitcoin network volume*, which includes all Bitcoin transfers caused by monetary transactions within the Bitcoin currency network,
- *daily views on the English Bitcoin Wikipedia page* as a proxy for measuring user attention,
- *dummy variables for 24 events gathered from* `https://en.bitcoin.it/wiki/History`.

Modelling bitcoin price dynamics: VAR & VECM

$\rightarrow$ Glaser et al. (2014) are the first to consider both exchange (EV) and network volumes (NV): their idea is that if a customer want to buy bitcoin to pay for goods or services, exchange and network volumes will share similar dynamics, otherwise only exchange-based volumes will be affected.

$\Rightarrow$ They found that the both increases in Wikipedia searches and in exchange volumes do not impact network volumes, and there is no migration between exchange and network volumes, so that they argued that (uninformed) users mostly stay within exchanges, holding Bitcoin only as an alternative investment and not as a currency.

$\Rightarrow$ Glaser et al. (2014) found that Bitcoin users seem to be positively biased towards Bitcoin, because important negative events, like thefts and hacks, did not lead to significant price corrections.

Modelling bitcoin price dynamics: VAR & VECM

Bouoiyour and Selmi (2015), Bouoiyour et al. (2015) and Kancs et al. (2015) are the first studies to consider three sets of drivers to model bitcoin price dynamics:

- **technical drivers (bitcoin supply and demand)**,
- **attractiveness indicators**
- and **macroeconomic variables**.

In general, all papers confirm that bitcoin attractiveness factors are still the main drivers of bitcoin price, followed by traditional supply and demand related variables, while global macro-financial variables play no role.

Example: Bouoiyour and Selmi (2015) use these variables: ...

Modelling bitcoin price dynamics: VAR & VECM

Variable	***Explanation***
	Technical drivers
The exchange-trade ratio (ETR)	Bitcoins are used primarily for two purposes: purchases and exchange rate trading. The Blockchain website provides the total number of transactions and their volume excluding the exchange rate trading. In addition, the ratio between volume of trade (primarily purchases) and exchange transactions is also provided.
Bitcoin monetary velocity (MBV)	It is the frequency at which one unit of bitcoin is used to purchase tradable or non-tradable products for a given period. In the Bitcoin system, the monetary velocity of BitCoin circulation is proxied by the so-called *BitCoin days destroyed*. This variable is calculated by taking the number of BitCoins in transaction and multiplying it by the number of days since those coins were last spent.
The estimated output volume (EOV)	It is similar to the total output volume with the addition of an algorithm which tries to remove change from the total value. This estimate should reflect more accurately the true transaction volume. A negative relationship between the estimated output volume and bitcoin price is expected.
The Hash Rate	The estimated number of giga-hashes per second (billions of hashes per second) the bitcoin network is performing. It is an indicator of the processing power of the Bitcoin network
	Attractiveness indicators
Investors' attractiveness (TTR)	daily Bitcoin views from Google, because it is able to properly depict the speculative character of users
	Macroeconomic variables
The gold price (GP)	Bitcoin does not have an underlying value derived from consumption or production process such as gold.
The Shangai market index (SI)	The Shangai market is considered one of the biggest player in Bitcoin economy and it is considered as a potential source of Bitcoin price volatility.

Modelling bitcoin price dynamics: VAR & VECM

$\Rightarrow$ Using a dataset spanning between 05/12/2010 and 14/06/2014, Bouoiyour and Selmi (2015) found that in the short-run, the investors attractiveness, the exchange-trade ratio, the estimated output volume and the Shangai index have a positive and significantly impact on Bitcoin price, while the monetary velocity, the hash rate and the gold price have no effect.

$\Rightarrow$ Instead, in the long-run, only the exchange-trade ratio and the hash rate have a significant impact on bitcoin price dynamics.

These results hold also with the inclusion of a dummy variable to account for the bankruptcy of a major Chinese bitcoin trading company in 2013, with oil prices, the Dow Jones index and a dummy variable to consider the closure of the Road Silk by the FBI in October 2013.

Modelling bitcoin price dynamics: VAR & VECM

Most macro-financial analyses devoted to bitcoin prices employ:

1) Vecto-AutoRegression (VAR) models,

$$\Delta \mathbf{Y}_{t-1} = \alpha + \Phi_1 \Delta \mathbf{Y}_{t-1} + \Phi_2 \Delta \mathbf{Y}_{t-2} + ... + \Phi_p \Delta \mathbf{Y}_{t-p} + \varepsilon_t \quad (7)$$

2) Vector Error Correction (VEC) models,

$$\Delta \mathbf{Y}_{t-1} = \alpha + \mathbf{B}\Gamma \mathbf{Y}_{t-1} + \zeta_1 \Delta \mathbf{Y}_{t-1} + \zeta_2 \Delta \mathbf{Y}_{t-2} + ... + \zeta_{p-1} \Delta \mathbf{Y}_{t-(p-1)} + \varepsilon_t \quad (8)$$

where $\mathbf{B}$ are the factor loadings, while Γ the cointegrating vector.

Kristoufek (2013) is the first author to propose a multivariate approach: hefound a significant bidirectional relationship, where Google trends search queries influence prices and viceversa, suggesting that speculation and trend chasing dominate the bitcoin price dynamics.

Modelling bitcoin price dynamics: VAR & VECM

Glaser et al. (2014) extended previous research by studying the aggregated behavior of new and uninformed Bitcoin users within the time span from 2011 to 2013, to identify why people gather information about Bitcoin and their motivation to subsequently participate in the Bitcoin system.

The main novelty is the use of regressors that are related to both bitcoin **attractiveness** and bitcoin **supply and demand**:

- *daily BTC price data*,
- *daily exchange volumes* in BTC,
- *Bitcoin network volume*, which includes all Bitcoin transfers caused by monetary transactions within the Bitcoin currency network,
- *daily views on the English Bitcoin Wikipedia page* as a proxy for measuring user attention,
- *dummy variables for 24 events gathered from* `https://en.bitcoin.it/wiki/History`.

Modelling bitcoin price dynamics: VAR & VECM

$\rightarrow$ Glaser et al. (2014) are the first to consider both exchange (EV) and network volumes (NV): their idea is that if a customer want to buy bitcoin to pay for goods or services, exchange and network volumes will share similar dynamics, otherwise only exchange-based volumes will be affected.

$\Rightarrow$ They found that the both increases in Wikipedia searches and in exchange volumes do not impact network volumes, and there is no migration between exchange and network volumes, so that they argued that (uninformed) users mostly stay within exchanges, holding Bitcoin only as an alternative investment and not as a currency.

$\Rightarrow$ Glaser et al. (2014) found that Bitcoin users seem to be positively biased towards Bitcoin, because important negative events, like thefts and hacks, did not lead to significant price corrections.

Modelling bitcoin price dynamics: VAR & VECM

Bouoiyour and Selmi (2015), Bouoiyour et al. (2015) and Kancs et al. (2015) are the first studies to consider three sets of drivers to model bitcoin price dynamics:

- **technical drivers (bitcoin supply and demand)**,
- **attractiveness indicators**
- and **macroeconomic variables**.

In general, all papers confirm that bitcoin attractiveness factors are still the main drivers of bitcoin price, followed by traditional supply and demand related variables, while global macro-financial variables play no role.

Example: Bouoiyour and Selmi (2015) use these variables: ...

Modelling bitcoin price dynamics: VAR & VECM

Variable	***Explanation***
	Technical drivers
The exchange-trade ratio (ETR)	Bitcoins are used primarily for two purposes: purchases and exchange rate trading. The Blockchain website provides the total number of transactions and their volume excluding the exchange rate trading. In addition, the ratio between volume of trade (primarily purchases) and exchange transactions is also provided.
Bitcoin monetary velocity (MBV)	It is the frequency at which one unit of bitcoin is used to purchase tradable or non-tradable products for a given period. In the Bitcoin system, the monetary velocity of BitCoin circulation is proxied by the so-called *BitCoin days destroyed*. This variable is calculated by taking the number of BitCoins in transaction and multiplying it by the number of days since those coins were last spent.
The estimated output volume (EOV)	It is similar to the total output volume with the addition of an algorithm which tries to remove change from the total value. This estimate should reflect more accurately the true transaction volume. A negative relationship between the estimated output volume and bitcoin price is expected.
The Hash Rate	The estimated number of giga-hashes per second (billions of hashes per second) the bitcoin network is performing. It is an indicator of the processing power of the Bitcoin network
	Attractiveness indicators
Investors' attractiveness (TTR)	daily Bitcoin views from Google, because it is able to properly depict the speculative character of users
	Macroeconomic variables
The gold price (GP)	Bitcoin does not have an underlying value derived from consumption or production process such as gold.
The Shangai market index (SI)	The Shangai market is considered one of the biggest player in Bitcoin economy and it is considered as a potential source of Bitcoin price volatility.

Modelling bitcoin price dynamics: VAR & VECM

$\Rightarrow$ Using a dataset spanning between 05/12/2010 and 14/06/2014, Bouoiyour and Selmi (2015) found that in the short-run, the investors attractiveness, the exchange-trade ratio, the estimated output volume and the Shangai index have a positive and significantly impact on Bitcoin price, while the monetary velocity, the hash rate and the gold price have no effect.

$\Rightarrow$ Instead, in the long-run, only the exchange-trade ratio and the hash rate have a significant impact on bitcoin price dynamics.

These results hold also with the inclusion of a dummy variable to account for the bankruptcy of a major Chinese bitcoin trading company in 2013, with oil prices, the Dow Jones index and a dummy variable to consider the closure of the Road Silk by the FBI in October 2013.

Modelling bitcoin price dynamics: Bayesian VAR

3) Bayesian VARs models

Bayesian methods treat the value of an unknown model parameter vector θ as a probability distribution $\pi(\theta|Y)$, which is the called the *posterior* distribution of θ given the data Y.

The prior distribution, $\pi(\theta)$, is set externally and reflects the researcher's *prior* ideas on the unknown parameter vector, while $l(Y|\theta)$ is the *likelihood* function, which depends on the information from the given data Y.

The Bayes' theorem is then used to link all these distributions by means of this formula:

$$\pi(\theta|Y) = \frac{\pi(\theta)l(Y|\theta)}{\int \pi(\theta)l(Y|\theta)d\theta}$$

Given that the denominator is a normalizing constant, the posterior is proportional to the product of the likelihood and the prior, that is $\pi(\theta|Y) \propto \pi(\theta)l(Y|\theta)$.

Modelling bitcoin price dynamics: Bayesian VAR

Let consider the following reduced form VAR,

$$Y_t = \Phi_0 + \Phi_1 Y_{t-1} + \ldots + \Phi_p Y_{t-p} + \varepsilon_t, \quad \varepsilon_t \sim N(\mathbf{0}, \Sigma)$$

where $Y_t = (Y_{1t}, \ldots Y_{nt})$ is a $n \times 1$ vector, Φ_0 is a $n \times 1$ vector of constants, Φ_l with $l = 1, \ldots, p$ are the usual autoregression $n \times n$ coefficient matrices.

The previous equation can be written more compactly as $Y_t = \Phi' X_t + \varepsilon_t$ using $X_t = [1 \; Y'_{t-1}, \ldots, Y'_{t-p}]'$ and $\Phi = [\Phi_0 \; \Phi_1 \ldots \Phi_p]$. If the variables and shocks are further grouped as follows $Y = [Y_1, \ldots, Y_T]'$, $X = [X_1, \ldots, X_T]'$, $E = [\varepsilon_1, \ldots, \varepsilon_T]'$, we can write the VAR model even more compactly:

$$Y = X\Phi + E$$

A Bayesian VAR combines the likelihood function $L(Y|\Phi, \Sigma)$ with a prior distribution $p(\Phi, \Sigma)$ to get a posterior distribution for the model parameters $p(\Phi, \Sigma|Y)$:

$$p(\Phi, \Sigma|Y) \propto p(\Phi, \Sigma) L(Y|\Phi, \Sigma)$$

Modelling bitcoin price dynamics: Bayesian VAR

There are several possible choices for priors to be used with Bayesian VAR models: I present below the *conjugate normal-inverse Wishart prior* which is a widely used choice and it is implemented into the **bvarr** package.The prior is reported below:

$$\begin{cases} \Sigma \sim \mathcal{IW}(\underline{S}, \underline{\nu}) \\ \Phi|\Sigma \sim N(\underline{\Phi}, \Sigma \otimes \underline{\Omega}) \end{cases}$$

where the scale matrix $\underline{S}$ is diagonal and its non-zero elements assure that the mean of Σ is equal to the fixed covariance matrix of the standard Minnesota prior,

$$(\underline{S})_{ii} = (\underline{\nu} - n - 1)\hat{\sigma}_i^2$$

and σ_i^2 is commonly set to be equal to the variance estimate of residuals in a univariate AR model. The degrees of freedom of the inverse Wishart distribution are set to be greater than or equal to the $\max\{n+2, n+2h-T\}$ to guarantee the existence of the prior variance of the regression parameters and the posterior variances of the forecasts at horizon h.

Modelling bitcoin price dynamics: Bayesian VAR

The matrix $\underline{\Phi}$ is set to $\underline{\Phi} = E(\Phi)$ and the matrices $\underline{\Phi}_l$ are given by:

$$(\underline{\Phi}_l)_{ij} = \begin{cases} \delta_i, & i = j, \quad l = 1 \\ 0 & \text{otherwise} \end{cases}$$

The matrix $\underline{\Omega}$ is diagonal and it depends on the following hyperparameters:

$$\begin{aligned} \underline{\Omega} &= \text{diag}\{\underline{\Omega}_0, \underline{\Omega}_1, \ldots, \underline{\Omega}_p\} \\ (\underline{\Omega}_l)_{jj} &= \left(\frac{\lambda}{l^{\lambda_l} \hat{\sigma}_j} \right)^2 \quad l = 1, \ldots, p, \quad \underline{\Omega}_0 = \lambda_0^2 \end{aligned}$$

where λ determines the overall tightness of the prior and the relative weight of the prior with respect to the information incorporated in the data, λ_l manages the speed of the decrease of the prior variance with increasing the lag length, while λ_0 controls the relative tightness of the prior for the constant terms.

Modelling bitcoin price dynamics: Bayesian VAR

The posterior distribution formed by combining the previous prior distribution with a likelihood function is also normal - inverse Wishart, see e.g. Zellner (1996):

$$\begin{cases} \Sigma|Y \sim \mathcal{IW}(\overline{S}, \overline{\nu}) \\ \Phi|\Sigma, Y \sim N(\overline{\Phi}, \Sigma \otimes \overline{\Omega}) \end{cases}$$

with the following parameters:

$$\begin{aligned} \overline{\nu} &= \underline{\nu} + T \\ \overline{\Omega} &= (\underline{\Omega}^{-1} + X'X)^{-1} \\ \overline{\Phi} &= \overline{\Omega} \cdot (\underline{\Omega}^{-1}\underline{\Phi} + X'Y) \\ \overline{S} &= \underline{S} + \hat{E}'\hat{E} + \hat{\Phi}'X'X\hat{\Phi} + \underline{\Phi}'\underline{\Omega}^{-1}\underline{\Phi} - \overline{\Phi}'\overline{\Omega}^{-1}\overline{\Phi} \\ \hat{\Phi} &= (X'X)^{-1}X'Y \\ \hat{E} &= Y - X\hat{\Phi} \end{aligned}$$

Modelling bitcoin price dynamics: Bayesian VAR

Doan et al. (1984) and Sims (1993) proposed to add two other priors to the previous prior distribution to include the beliefs that the data may be non-stationary and cointegrated:

$\Rightarrow$ A *sum-of-coefficients prior* assumes that the sum of all the lag parameters for each dependent variable is equal to one. This prior is implemented by combining the previous system with the following artificial dummy-observations:

$$
\begin{aligned}
Y^{SC} &= \frac{1}{\lambda_{sc}}[\text{diag}(\delta_1\mu_1, \ldots, \delta_n\mu_n)] \\
X^{SC} &= \frac{1}{\lambda_{sc}}[0_{n\times 1} \quad (1_{1\times p}) \otimes \text{diag}(\delta_1\mu_1, \ldots, \delta_n\mu_n)]
\end{aligned}
$$

where $(1_{1\times p})$ is a unitary $[1 \times p]$ vector, and μ_i is i-th component of vector μ, which contains the average values of initial p observations of all variables in the sample, $\mu = \frac{1}{p}\sum_{t=1}^{p} Y_t$.

When $\lambda_{sc} \to 0$ no cointegration exists and there are as many unit roots as variables.

Modelling bitcoin price dynamics: Bayesian VAR

$\Rightarrow$ The *dummy initial observation prior* proposed by Sims (1993) models the belief that the variables have a common stochastic trend, so that the average value for a variable is a linear combination of the average values of all the other variables.

A single dummy observation is added such that the values of all variables are set to be equal to the averages of the initial conditions μ_i normalized with a scaling factor λ_{io}:

$$
\begin{aligned}
Y^{IO} &= \frac{1}{\lambda_{io}}[(\delta_1\mu_1, \ldots, \delta_n\mu_n)] \\
X^{IO} &= \frac{1}{\lambda_{io}}[1 \quad (1_{1\times p}) \otimes (\delta_1\mu_1, \ldots, \delta_n\mu_n)]
\end{aligned}
$$

When $\lambda_{io} \to 0$, the model assumes that either all variables are stationary with means equal to sample averages of the initial observations, or non-stationary without drift terms and cointegrated.

Modelling bitcoin price dynamics: High-dimensional VAR models with LASSO

The last years have witnessed an increasing statistical literature dealing with the forecasting of high-dimensional multivariate time series, focusing particularly on the *lasso*, see Tibshirani et al. (1996), and its structured variants like the group lasso proposed by Yuan et al. (2006) and the sparse group lasso by Simon et al. (2013).

The R package **BigVAR** adapted the previous penalized regression solution algorithms to a multivariate time series setting: it considers the VARX-L framework proposed by Nicholson et al. (2017) and the class of Hierarchical Vector Autoregression (HVAR) models suggested by Nicholson et al.(2016) that deals with the issue of VAR lag order selection by imposing a nested group lasso penalty.

Given the increasing dimension of cryptocurrencies datasets, these approaches can be of interest to financial professionals and researchers alike. I focus here on HVAR models.

Hierarchical Vector Autoregression (HVAR) models

4) HVAR class of models: Nicholson et al. (2016) proposed a class of models which include the lag order selection into hierarchical group lasso penalties.

HVAR(p) models induce sparsity and a low maximum lag order. Moreover, lag orders are allowed to change across marginal models, that is across variables.

The HVAR penalty structures are reported in Table 1.

Table 1: HVAR penalty functions

Group Name	$\mathcal{P}_Y(\Phi)$
Componentwise	$\sum_{i=1}^{n}\sum_{l=1}^{p}\|\|\Phi_i^{l:p}\|\|_2$
Own/Other	$\sum_{i=1}^{n}\sum_{l=1}^{p}\left[\|\|\Phi_i^{l:p}\|\|_2+\|\|\Phi_{i,-i}^{l},\Phi_i^{[l+1]:p}\|\|_2\right]$
Elementwise	$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{p}\|\|\Phi_{ij}^{l:p}\|\|_2$
Lag-weighted Lasso	$\sum_{l=1}^{p} l^{\gamma}\|\|\Phi^{l}\|\|_1$

Hierarchical Vector Autoregression (HVAR) models

The *Componentwise HVAR* penalty allows for the maximum lag order to change across marginal models but, within a single variable equation, all components have the same maximum lag. Therefore, we can have at maximum n different lag orders.

The *Own/Other HVAR* penalty is similar to the Componentwise HVAR, but it prioritizes the coefficients of lagged values of the series of forecasting interest (the so-called 'own' lags) over those of other variables.

$\Rightarrow$ This approach is similar to a Bayesian VAR with a Minnesota Prior (Litterman, 1979) where the variable own lags are considered more informative than the lags of other variables.

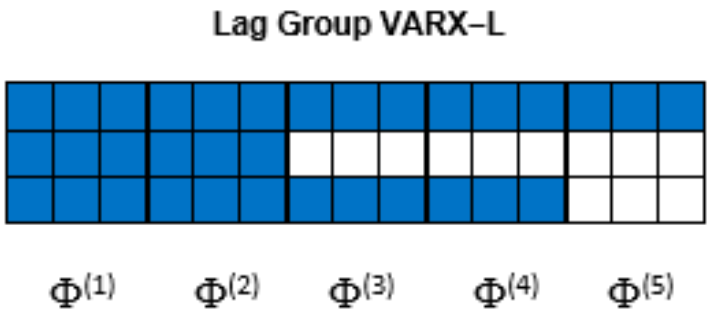
Hierarchical Vector Autoregression (HVAR) models

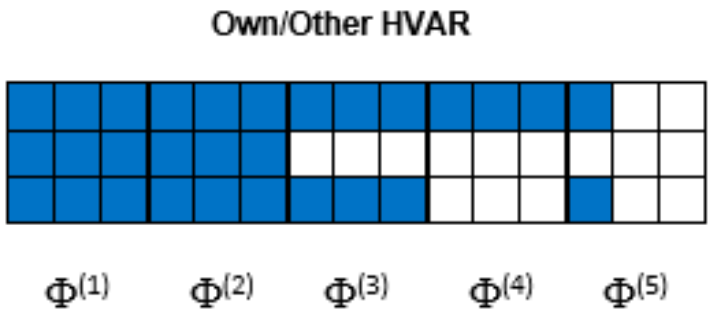
The *Elementwise HVAR* is the most general structure, because every variable in every equation is allowed to have its own maximum lag so that there can be n^2 possible lag orders.

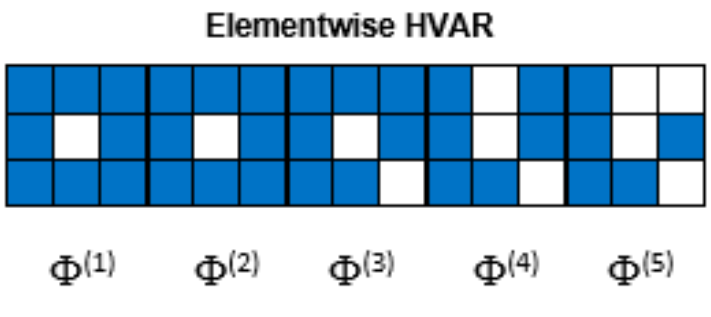
The *Lag-weighted Lasso* penalty structure is a lasso penalty that increases geometrically with lags and the additional penalty parameter $\gamma \in [0, 1]$ is jointly estimated with λ using sequential cross-validation.

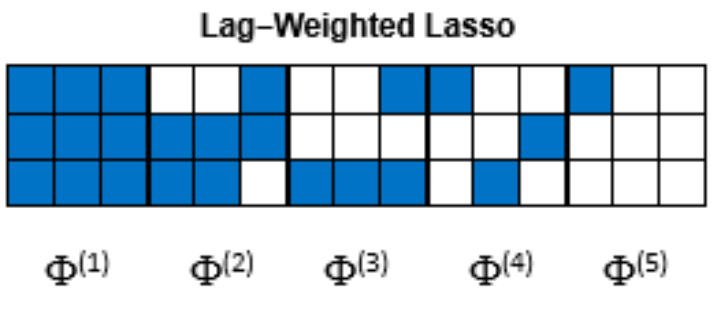
Hierarchical Vector Autoregression (HVAR) models

Examples of the previous four sparsity patterns are reported below:









A (simple) forecasting comparison

I performed a simple exercise to backtest the forecasting performances of the previous multivariate models. I used the dataset `data_bitcoin_multi` from the **bitcoinfinance** package. This is a dataframe of 1447 rows and 12 columns containing the following variables:

- *timestamp*: daily time-stamp;
- *Close*: Average BTCUSD market price across major bitcoin exchanges. Source: blockchain.info;
- *Volume_traded_USD*: The total USD value of trading volume on major bitcoin exchanges. Source: blockchain.info;
- *Google*: Normalized daily Google search data for the word "bitcoin";
- *Transaction_value*: The total estimated value of transactions on the Bitcoin blockchain. Source: blockchain.info;
- *Hash_Rate*: The estimated number of tera hashes per second (trillions of hashes per second) the Bitcoin network is performing. Source: blockchain.info ...

A (simple) forecasting comparison

- *Gold*: Gold price in USD. Source: investing.com;
- *Shanghai_index*: The Shanghai market index. Source: yahoo.finance;
- *total_bitcoins*: The total number of bitcoins that have already been mined; in other words, the current supply of bitcoins on the network. Source: blockchain.info;
- *New_posts*: The number of new posts on online BitCoin forums extracted from bitcointalk.org;
- *New_members*: The number of new members on online BitCoin forums extracted from bitcointalk.org;
- *Dow_Jones*: Dow Jones stock market index. Source: yahoo.finance;

A (simple) forecasting comparison

I used a 250-day rolling window to compute the 1-step and 10-step ahead forecasts for each model, as well as the RMSE and MAE. More specifically, I considered the following models:

- a VAR model with all the variables in levels;
- a VAR model with all the variables in first differences;
- a VAR model with all the variables in log-levels;
- a VAR model with all the variables in first log-differences (= log-returns);
- a VECM model with all the variables in levels/first differences;
- a VECM model with all the variables in log-levels/log-returns;
- a Bayesian VAR model with the conjugate normal-inverse Wishart prior and all the variables in levels;
- a Bayesian VAR model with the conjugate normal-inverse Wishart prior and all the variables in first differences;
- a Bayesian VAR model with the conjugate normal-inverse Wishart prior and all the variables in log-levels;
- a Bayesian VAR model with the conjugate normal-inverse Wishart prior and all the variables in first log-differences (= log-returns);
- a Elementwise HVAR for data in log-returns.

To simplify the computational setting, I considered only multivariate models with lags up to 4 and only one HVAR model.

A (simple) forecasting comparison

Despite the limitations of this forecasting exercise, some interesting results did emerge:

the HVAR model was the best model according to all metrics, thus confirming the positive evidence reported in Nicholson et al. (2016) and Nicholson et al. (2017b).

Bayesian models showed -in general- very good results, whereas cointegrated models had computational problems:

$\rightarrow$ the latter evidence is a well known inference issue and I refer the interested reader to Fantazzini and Toktamysova (2015) - section 4.4 and references therein- for more details.

Chapter 8: Testing for financial bubbles and explosive price behavior

Detecting Bubbles and explosive behavior in bitcoin prices

Tests for financial bubbles can be by grouped into two sets:

1. Tests to detect a single bubble:
   - the Log Periodic Power Law (LPPL) model;
   - the Fry (2014) model and the role of volatility.
2. Tests to detect (potentially) multiple bubbles:
   - the DS LPPLS Confidence and Trust indicators;
   - the Generalized-Supremum ADF (GSADF) test;
   - the EXponential Curve Fitting (EXCF) method.

Due to time constraints, I will briefly present only a couple of them.

Detecting Bubbles and explosive behavior in bitcoin prices

1A) Testing for a single bubble: LPPL models. The expected value of the asset log price in a upward trending bubble according to the LPPL equation is given by,

$$E[\ln p(t)] = A + B(t_c - t)^{\beta} + C(t_c - t)^{\beta} \cos[\omega \ln(t_c - t) - \phi] \quad (9)$$

where $A > 0$ is the value of $[\ln p(t_c)]$ at the critical time t_c which is interpreted as the end of the bubble,

$B < 0$ the increase in $[\ln p(t)]$ over the time unit before the crash

$C \neq 0$ is the proportional magnitude of the oscillations around the exponential growth,

$0 < \beta < 1$ to ensure a finite price at the critical time t_c of the bubble and quantifies the power law acceleration of prices,

ω is the frequency of the oscillations during the bubble,

while $0 < \phi < 2\pi$ is a phase parameter.

Detecting Bubbles and explosive behavior in bitcoin prices

Financial bubbles are defined in the LPPL model as transient regimes of faster-than- exponential price growth resulting from positive feedbacks, and these regimes represent "positive bubbles".

Example: Conditions for a (positive) bubble to occur within this framework:

1. $0 < \beta < 1$, which guarantees that the crash hazard rate accelerates.
2. The second major condition is that the crash rate should be non-negative, as highlighted by van Bothmer and Meister (2003),
$$b \equiv -B\beta - |C|\sqrt{\beta^2 + \omega^2} \geq 0.$$
3. Lin et al. (2014) added a third condition, requiring that the residuals from fitting equation (9) should be stationary.

$\Rightarrow$ MacDonell (2014) used the LPPL model to forecast successfully the bitcoin price crash that took place on December 4, 2013

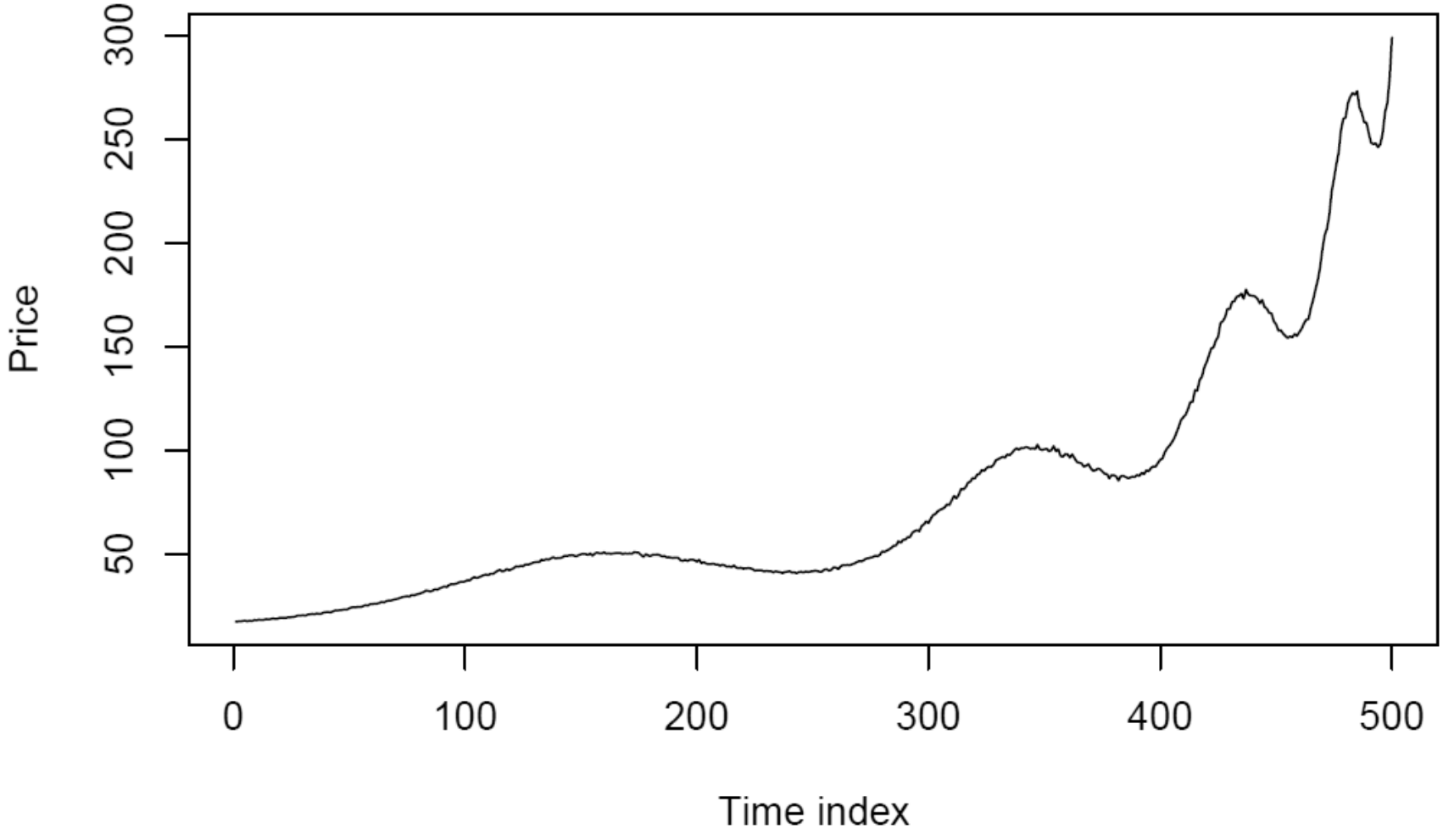
Detecting Bubbles and explosive behavior in bitcoin prices

To have an idea of the LPPL model, let's simulate a price trajectory following this model using the function `lppl_simulate()` from the **bubble** package:

```
lppl_simulate=function(T=500, true_parm){
    bet=true_parm[1]; ome=true_parm[2]; phi=true_parm[3];
    A= true_parm[4]; B =true_parm[5]; C= true_parm[6]; ws=true_parm[7];
    tc=true_parm[8];
    tt_sim=seq(1, T, 1);
    sdum=rep(1,T);
    f_t=(tc - tt_sim)^bet;
    g_t=( (tc - tt_sim)^bet )*cos( ome*log(tc - tt_sim) + phi );
    x=exp(A*sdum +B*f_t + C*g_t +sqrt(ws)*rnorm(T) );
    plot(x, type="l", xlab = "Time index", ylab = "Price")
    return(x)
}
```

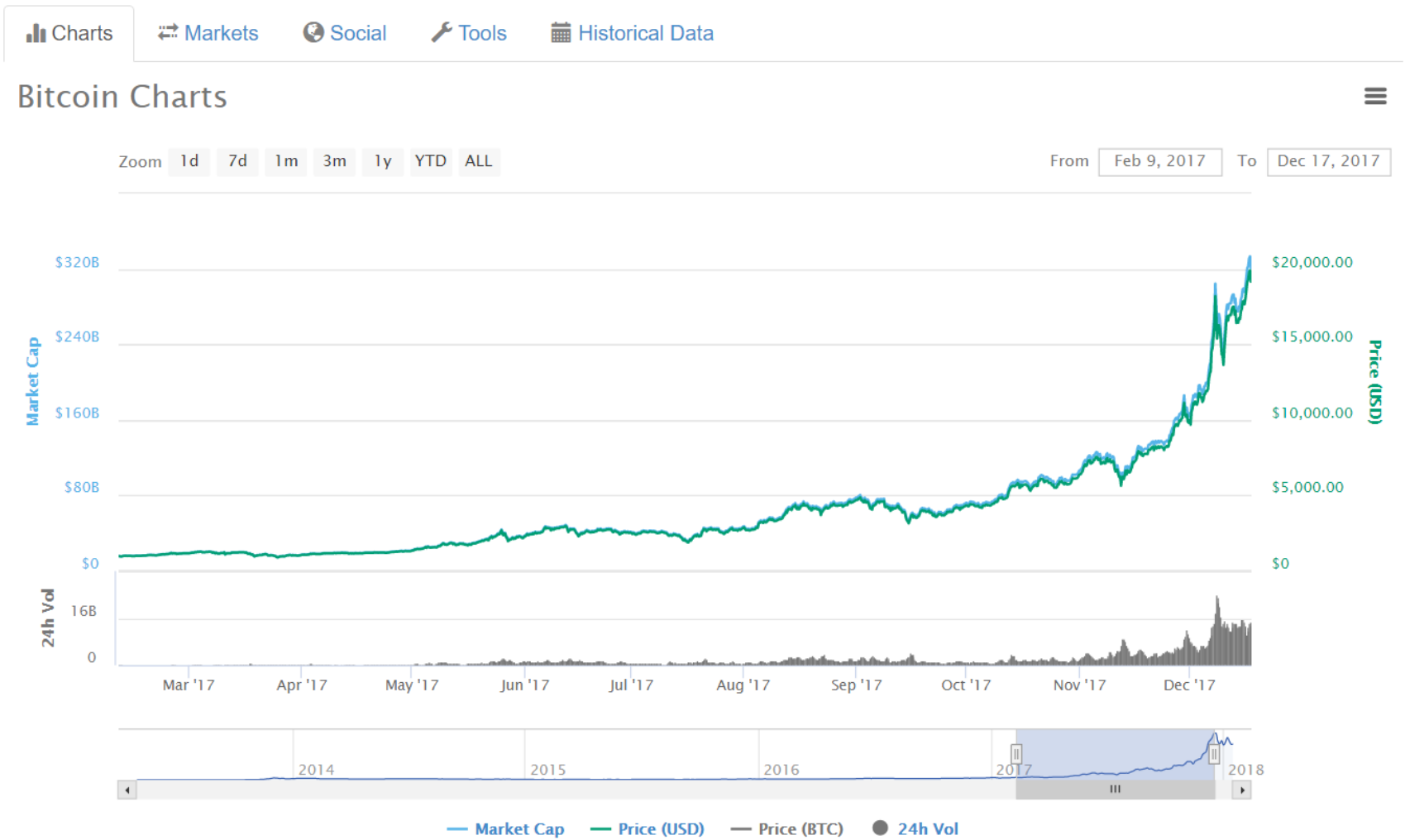
```
tparm=c(0.353689, 9.154368, 2.074608, 7.166421,-0.434324, 0.035405,
        0.000071, 530)
aa=lppl_simulate(500,tparm)
```

Detecting Bubbles and explosive behavior in bitcoin prices



Detecting Bubbles and explosive behavior in bitcoin prices

Sound familiar?



Detecting Bubbles and explosive behavior in bitcoin prices

2B) Testing for a multiple bubbles: the Generalized-Supremum ADF test (GSADF).

Tests specifically designed for detecting multiple bubbles were recently proposed by Phillips and Yu (2011), Phillips et al. (2011) and Phillips et al. (2015) and they share the same idea of using sequential tests with rolling estimation windows.

More specifically, *these tests are based on sequential ADF-type regressions* using time windows of different size, and they can consistently identify and date-stamp multiple bubble episodes even in small sample sizes.

We will focus below on the *Generalized-Supremum ADF test (GSADF)* proposed by Phillips, et al. (2015) -PSY henceforward- which builds upon the work by Phillips and Yu (2011) and Phillips et al. (2011), because it has better statistical properties in detecting multiple bubble than the latter two tests.

Detecting Bubbles and explosive behavior in bitcoin prices

This test employs an ADF regression with a rolling sample, where the starting point is given by the fraction r_1 of the total number of observations, the ending point by the fraction r_2, while the window size by $r_w = r_2 - r_1$. The ADF regression is given by

$$y_t = \mu + \rho y_{t-1} + \sum_{i=1}^{p} \phi_{r_w}^{i} \Delta y_{t-i} + \varepsilon_t \tag{10}$$

where the null hypothesis is of a unit root $\rho = 1$ versus an alternative of a mildly explosive autoregressive coefficient $\rho > 1$.

The backward sup ADF test proposed by PSY (2015) fixes the endpoint at r_2 while the window size is expanded from an initial fraction r_0 to r_2, so that the test statistic is given by:

$$BSADF_{r_2}(r_0) = \sup_{r_1 \in [0, r_2 - r_0]} ADF_{r_1}^{r_2} \tag{11}$$

Detecting Bubbles and explosive behavior in bitcoin prices

The generalized sup ADF (GSADF) test is computed by repeatedly performing the BSADF test for each $r_2 \in [r_0, 1]$:

$$GSADF(r_0) = \sup_{r_2 \in [r_0, 1]} BSADF_{r_2}(r_0) \tag{12}$$

PSY (2015, Theorem 1) provides the limiting distribution of (12) under the null of a random walk with asymptotically negligible drift (vs an alternative of a mildly explosive process), while critical values are obtained by numerical simulation.

If the null hypothesis of no bubbles is rejected, it is then possible to date-stamp the starting and ending points of one (or more) bubble(s) in a second step...

Detecting Bubbles and explosive behavior in bitcoin prices

More specifically,

$\rightarrow$ the *starting point* is given by the date -denoted as T_{r_e}- when the sequence of BSADF test statistics crosses the critical value from below,

$\rightarrow$ whereas the *ending point* -denoted as T_{r_f}- when the BSADF sequence crosses the corresponding critical value from above:

$$\hat{r}_e = \inf_{r_2 \in [r_0,1]} \left\{ r_2 : BSADF_{r_2}(r_0) > cv_{r_2}^{\beta_T} \right\} \quad (13)$$

$$\hat{r}_f = \inf_{r_2 \in [\hat{r}_e + \delta \log(T)/T,1]} \left\{ r_2 : BSADF_{r_2}(r_0) < cv_{r_2}^{\beta_T} \right\} \quad (14)$$

where $cv_{r_2}^{\beta_T}$ is the $100(1-\beta_T)\%$ right-sided critical value of the BSADF statistic based on $\lfloor Tr_2 \rfloor$ observations, $\lfloor \cdot \rfloor$ is the integer fun.

δ is a tuning parameter which determines the minimum duration for a bubble and is usually set to 1, see PSY (2015) and references therein, thus implying a minimum bubble-duration condition of $\ln(T)$ observations.

Detecting Bubbles and explosive behavior in bitcoin prices

Malhotra and Maloo (2014) tested for the presence of multiple bubbles using the GSADF test with data ranging from mid-2011 till February 2014:

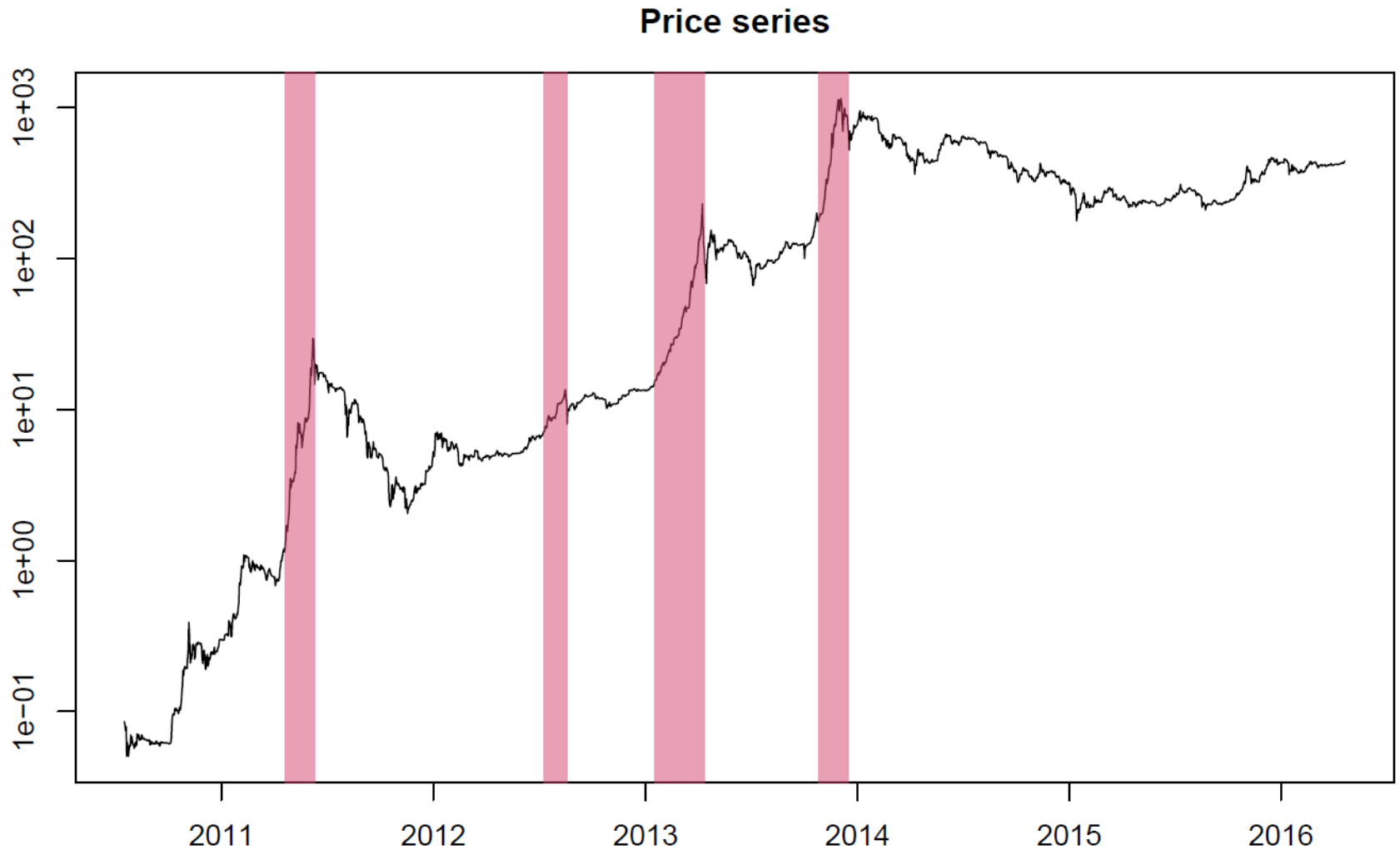
$\Rightarrow$ they found evidence of explosive behaviour in the bitcoin-USD exchange rates during ***August – October 2012*** and ***November, 2013 – February, 2014***.

$\Rightarrow$ They suggested that the first episode of bubble behavior (August – October 2012) could be attributed to the sudden increase in media attention towards bitcoin,

$\Rightarrow$ whereas the second episode to a large set of reasons including the US debt ceiling crisis, the shutdown of Silk Road by the FBI, the rise of Chinese exchange BTC-China, and the increasing number of warnings issued by regulatory authorities and central banks worldwide following the shutdown of the Japanese exchange Mt.Gox.

Detecting Bubbles and explosive behavior in bitcoin prices

Bitcoin price series with periods of explosive behaviour according to the GSADF test highlighted in red, (a minimum bubble duration of 30 days is used).



Chapter 9: Univariate volatility modelling

Univariate volatility modelling

These models can be by grouped into 2 families:

1. Generalized Autoregressive Heteroscedasticity (GARCH)
   - GARCH models
   - Asymmetric and Nonlinear GARCH models
   - Fractionally Integrated Models
2. Realized Volatility models
   - Realized Volatility
   - Realized Volatility and Jumps

GARCH models

A generalization of the ARCH models was developed by Bollerslev (1986) which allowed for a more flexible but parsimonious specification.

A variance process σ_t^2 is called a GARCH(1,1) process, if

$$\sigma_t^2 = \omega + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2,$$

Sufficient conditions to ensure the positivity of σ_t^2 are $\omega > 0$, $\alpha_1 \geq 0 \setminus$ and $\beta_1 \geq 0$. $\diamond$

GARCH models

Properties of the GARCH(1,1) Process

(i) $\varepsilon_t^2 = \omega + (\alpha_1 + \beta_1)\varepsilon_{t-1}^2 - \beta_1 \nu_{t-1} + \nu_t$ with
$\nu_t \equiv \varepsilon_t^2 - \sigma_t^2 = \sigma_t^2(z_t^2 - 1)$

(ii) ε_t^2 is stationary if $|\alpha_1 + \beta_1| < 1$.

(iii) $\sigma_\varepsilon^2 = V[\varepsilon_t] = \frac{\omega}{1-\alpha_1-\beta_1}$

(iv) $\varepsilon_t^2 = \sigma_\varepsilon^2 + (\alpha_1 + \beta_1)(\varepsilon_{t-1}^2 - \sigma_\varepsilon^2) - \beta_1 \nu_{t-1} + \nu_t$

(v) $\sigma_t^2 = \sigma_\varepsilon^2 + \alpha_1(\varepsilon_{t-1}^2 - \sigma_\varepsilon^2) + \beta_1(\sigma_{t-1}^2 - \sigma_\varepsilon^2)$

(vi) $\mathrm{K}_\varepsilon = \frac{E[\varepsilon_t^4]}{E[\varepsilon_t^2]^2} = \frac{3(1-(\alpha_1+\beta_1))}{1-2\alpha_1^2-(\alpha_1+\beta_1)^2} = \frac{6\alpha_1^2}{1-2\alpha_1^2-(\alpha_1+\beta_1)^2} + 3 > 3$
for $z_t \sim$ iid N(0,1) and $2\alpha_1^2 + (\alpha_1 + \beta_1)^2 < 1$.

(vii) $\sigma_t^2 = \sum_{i=1}^{\infty} \beta_1^{i-1}\omega + \alpha_1 \sum_{i=1}^{\infty} \beta_1^{i-1}\varepsilon_{t-i}^2$ if $\beta_1 < 1$

GARCH models

Maximum Likelihood Estimation with Gaussian z_t

Let assume the disturbance term ε_t to follow a GARCH(p, q)-process. It is convenient to condition on the first $m = max(p, q)$ observations ($t = -m+1, -m+2, \ldots, 0$) and to use observations $t = 1, 2, \ldots, T$ for estimation. With Gaussian z_t's the likelihood is given by

$$\ln L(\theta) = -\frac{T}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T}\ln\sigma_t^2 - \frac{1}{2}\sum_{t=1}^{T}\frac{(Y_t - X_t'\gamma)^2}{\sigma_t^2} \qquad (15)$$

where we assume a general model of the type

$$Y_t = X_t'\gamma + z_t\sqrt{\sigma_t^2}, \quad z_t \sim N(0,1)$$
$$\sigma_t^2 = \omega + \alpha_p^*(L)\varepsilon_t^2 + \beta_q^*(L)\sigma_t^2$$

so that the parameter vector is $\theta = (\gamma, \omega, \alpha_1, \ldots, \alpha_p, \beta_1, \ldots, \beta_q)$.

The maximization of $\log L(\theta)$ cannot be done analytically like in the homoscedastic case, since σ_t^2 is a function not only of $(\omega, \alpha_1, \ldots, \alpha_p, \beta_1, \ldots, \beta_q)$, but also of γ through ε_{t-i}^2 and σ_{t-i}^2.

GARCH models

- In the GARCH models positive and negative shocks have the same effect on the conditional variances. In practice we observe that the conditional volatility reacts differently to positive and negative effects. This is known as the “leverage effect". To circumvent this weakness *nonlinear* GARCH models have been developed.

- To study the tail behavior (e.g. positive excess kurtosis) of ε_t, we have to ensure the existence of the fourth moment of ε_t. The conditions therefore are very restrictive e.g. in the ARCH(1) model $\alpha_1^2 \in [0, \frac{1}{3})$.

- Often, one needs to model a high persistence of a past shock (high p and q). $\Rightarrow$ Fractionally integrated or integrated GARCH models.

GARCH models

Exponential GARCH (EGARCH).

The exponential GARCH model (EGARCH) of order (1,1) is of the form:

$$\ln(\sigma_t^2) = \omega + \phi z_{t-1} + \psi\left(|z_{t-1}| - E\,|z_{t-1}|\right) + \beta \ln(\sigma_{t-1}^2)$$

where $E|z_t| = (2/\pi)^{1/2}$ when $z_t \sim N(0,1)$, where the parameters ω, β_i, α_i are not restricted to be nonnegative.\

Let define

$$g(z_t) \equiv \phi z_t + \psi[|z_t| - E|z_t|]$$

by construction $\{g(z_t)\}_{t=-\infty}^{t=\infty}$ is a zero-mean, i.i.d. random sequence.

GARCH models

Threshold GARCH (T-GARCH).

The threshold GARCH model for a GARCH(1,1) specification is of the form:

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 \mathbf{1}_{[\varepsilon_{t-1}>0]} + \gamma\varepsilon_{t-1}^2 \mathbf{1}_{[\varepsilon_{t-1}<0]} + \beta\sigma_{t-1}^2,$$

where $\mathbf{1}_{[.]}$ is the indicator function.

In the original specification by Glosten-Jagannathan-Runkle (1993), it was formulated as

$$\sigma_t^2 = \omega + \alpha\varepsilon_{t-1}^2 + \gamma\varepsilon_{t-1}^2 \mathbf{1}_{[\varepsilon_{t-1}<0]} + \beta\sigma_{t-1}^2,$$

GARCH models

Power GARCH (a.k.a Asymetric Power ARCH - APARCH)

Ding, Granger and Engle's power GARCH model for $d > 0$

$$\sigma_t^d = \omega + \sum_{i=1}^{q} \alpha_i (|\varepsilon_{t-i} - \gamma_i \varepsilon_{t-i}|)^\delta + \sum_{i=1}^{p} \beta_i \sigma_{t-j}^\delta$$

where $d > 0$ and $-1 < \gamma_i < 1$ $(i = 1, ..., q)$. A leverage effect is present if $\gamma_i < 0$

$d = 2$ gives a regular GARCH model with leverage effects (i.e. a TGARCH model)

$d < 2$ gives a model for σ_t and is more robust to outliers than when $d = 2$

d can be fixed at a particular value or estimated by ML. Moreover this specification includes several other ARCH models (see original paper for details):

GARCH models An example with R

I deal with structural breaks in our bitcoin dataset by considering only the last two years of the Coindesk bitcoin data (the dataset starts after the famous bankruptcy of the MtGox exchange):

```
library(rugarch)
path.bit <- system.file("extdata","coindesk-bpi-USD-close.csv",package="bubble")
dat <- read.table(path.bit, dec = ".", sep =",", header = TRUE)
dat <- xts::xts(dat[,2], order.by=as.Date(dat[,1]))
dat<-dat['2014-04/']
bit.ret <- PerformanceAnalytics::CalculateReturns(dat, method="log")
bit.ret <- bit.ret[-1,]

# Asymmetric GARCH models
# EGARCH(1,1) model with Student-t errors
egarch11.spec <- ugarchspec(variance.model=list(model="eGARCH",garchOrder=c(1,1)),
                     mean.model=list(armaOrder=c(0,0)),distribution.model = "std")
bit.egarch11.fit <- ugarchfit(egarch11.spec, bit.ret)

# GJR-GARCH(1,1) model with Student-t errors
gjrgarch11.spec<-ugarchspec(variance.model=list(model="gjrGARCH",garchOrder=c(1,1)),
                    mean.model=list(armaOrder=c(0,0)), distribution.model = "std")
bit.gjrgarch11.fit <- ugarchfit(gjrgarch11.spec, bit.ret)

#APARCH(1,1) model with Student-t errors
aparch11.spec <- ugarchspec(variance.model=list(model="apARCH",garchOrder=c(1,1)),
                    mean.model=list(armaOrder=c(0,0)),distribution.model = "std")
bit.aparch11.fit <- ugarchfit(aparch11.spec, bit.ret)
```

GARCH models An example with R

```
# Fractionally Integrated GARCH models
# FIGARCH(1,d,1) with Student-t errors
figarch11.spec<-ugarchspec(variance.model=list(model="fiGARCH",garchOrder=c(1,1)),
                    mean.model=list(armaOrder=c(0,0)), distribution.model = "std")
bit.figarch11.fit <- ugarchfit(figarch11.spec, bit.ret)

# A simple benchmark: GARCH(1,1) with Student-t errors
garch11t.spec = rugarch::ugarchspec(variance.model = list(garchOrder=c(1,1)),
                  mean.model = list(armaOrder=c(0,0)), distribution.model = "std")
bit.garch11t.fit = rugarch::ugarchfit(spec=garch11t.spec, data=bit.ret,
                                      solver.control=list(trace = 1))

# Compare information criteria
model.list = list(garch11 = bit.garch11t.fit,
                  egarch11 = bit.egarch11.fit,
                  gjrgarch11 = bit.gjrgarch11.fit,
                  aparch11 = bit.aparch11.fit,
                  figarch1d1 = bit.figarch11.fit)
info.mat = sapply(model.list, infocriteria)
rownames(info.mat) = rownames(infocriteria(bit.garch11.fit))
info.mat
                garch11  egarch11 gjrgarch11  aparch11 figarch1d1
Akaike       -4.405188 -4.408857  -4.402561 -4.402217  -4.404138
Bayes        -4.374388 -4.371897  -4.365600 -4.359096  -4.367177
Shibata      -4.405277 -4.408984  -4.402687 -4.402389  -4.404265
Hannan-Quinn -4.393320 -4.394616  -4.388319 -4.385602  -4.389896
```

Realized Volatility

To understand RV, we must introduce the concept of *Integrated Volatility*. Suppose that the model for the variation of the price is a diffusion process

$$dp(t)^* = \mu(t)dt + \sigma(t)dW(t) \tag{16}$$

where $p(t)^*$ is the logarithm of instantaneous price, $dW(t)$ is a standard Brownian process, and $\sigma(t)$ is a stochastic process independent of $dW(t)$.

For this diffusion process, the *Integrated Volatility* (IV) associated with day t is defined as the integral of the instantaneous volatility over the one day integral $(t; t+1)$:

$$IV_{t+1} = \int_t^{t+1} \sigma^2(s)ds \tag{17}$$

Merton (1980) showed that the IV of a Brownian motion (17) can be approximated to an arbitrary precision using the sum of intraday squared returns.

Daily squared returns, as a volatility measure, constitute a poor ex post estimator, because they overestimate the volatility.

Realized Volatility

Integrated volatility is, instead, a good ex post measure and a theoretical benchmark for other volatility estimations.

Andersen et al. (2001a,b, 2003) and Barndorff-Nielsen and Shephard (2002) generalized this results to the class of special (finite mean) semi-martingales by using the quadratic variation theory: this class encompasses processes used in standard asset pricing applications, such as Ito diffusions, jump processes, and mixed jump diffusions.

Under such conditions and as the maximal length of returns go to zero, the sum of intraday squared returns converges to the integrated volatility of the prices, allowing us, in principle, to build an error free estimate of the actual volatility over a fixed-length time interval.

This nonparametric estimator is called *Realized Volatility*.

Realized Volatility

Barndorff-Nielsen and Shephard (2002) has demonstrated that the *quadratic variation* of a semimartingale that is defined as

$$[y_t] = \text{plim} \sum_{j=1}^{t_j<t} (y_{t-j} - y_{t_{j-1}})^2 \tag{18}$$

is equivalent to the integrated volatility when returns move as described in (16) and the drift element is continuous. The sum of successively high-frequency squared returns converges to the quadratic variation of price, (see Meddahi (2002) and Andersen et al. (2001a)).

The realized volatility is a consistent estimator of integrated volatility as the sampling frequency increases.

Let consider a discretely sampled Δ-period return be denoted by $y_t = p(t) - p(t - \Delta)$, and normalize the daily time interval to unity, so that we can label the corresponding discretely sampled daily returns by a single time subscript, $y_{t+1} = y_{t+1,1}$.. Besides, we have a total of n_t subintervals within the day.

Realized Volatility

The daily realized volatility is given by the summation of the corresponding $1/\Delta$ high-frequency intraday squared returns,

$$RV_{t+1} = \sum_{j=1}^{1/\Delta} y^2_{t+j\Delta,\Delta} = \sum_{i=1}^{n_t} y^2_{t,i} \tag{19}$$

As the sampling frequency from a diffusion is increased and even with a non zero mean process, the realized volatility provides a consistent measure of the integrated volatility over the fixed time interval (Andersen et al. (2001a,b), Andersen et al. (2003, 2007)):

$$\text{plim}_{\Delta \longrightarrow 0} \quad RV_{t+1} = \int_t^{t+1} \sigma^2(s) ds \tag{20}$$

Realized Volatility can be viewed over different time horizons longer than a single day d: multi-period volatilities are normalized sums of the one-period volatilities, that is a simple average of the daily quantity $RV^{(d)}$ (Corsi, 2009). For example, a weekly realized volatility w at time t

$$RV_t^{(w)} = (1w)^{-1} \left(RV_{t-1d}^{(d)} + RV_{t-2d}^{(d)} + \cdots + RV_{t-1w}^{(d)} \right) \tag{21}$$

where $1w = 5d$ indicate a time interval of one week , i.e. 5 working days.

Jumps

Recent studies have highlighted the importance of explicitly allowing for *jumps*, or discontinuities, in the estimation of parametric stochastic volatility models as well as in the pricing of options and other derivatives instruments (e.g., Andersen et al. (2002), Chan and Maheu, (2002), Chernov et al. (2003), Eraker et al. (2003), Maheu and McCurdy (2004), Khalaf et al. (2003), Huang and Tauchen (2005)).

The empirical evidence points out that the conditional variance of many assets is best described by a combination of a smooth and very slowly mean-reverting continuous sample path process, along with a much less persistent jump component, see, e.g., Andersen et al. (2007) and Bollerslev et al. (2009).

In order to better understand this phenomenon, we briefly present the basic bi-power variation theory of Barndorff-Nielsen and Shephard (2004,2006).

Jumps

If we denote the time t logarithmic price of the asset with $p(t)^*$, the continuous-time jump diffusion processes traditionally used in asset pricing finance are expressed in the following stochastic differential equation form,

$$dp(t)^* = \mu(t)dt + \sigma(t)dW(t) + k(t)dq(t) \tag{22}$$

where $\mu(t)$ is a continuous and locally bounded variation process, the stochastic volatility process $\sigma(t)$ is strictly positive and c?gl?d[1], $W(t)$ denotes a standard Brownian motion, $q(t)$ is a counting process with $dq(t) = 1$ corresponding to a jump at time t and $dq(t) = 0$ otherwise, while $k(t)$ refers to the size of the corresponding jumps.

The quadratic variation for the cumulative return process, $y(t) = p(t) - p(0)$, is given by (see Barndorff-Nielsen and Shephard (2004,2006), Andersen et al. (2007)):

$$[y, y]_t = \int_0^t \sigma^2(s)ds + \sum_{0<s\leq t} k^2(s) \tag{23}$$

[1] i.e. right continuous and the limit exists. This assumption allows for discrete jumps in the stochastic volatility process.

Jumps

The second term on the right-hand-side disappears when jumps are absent, and the quadratic variation is then simply equal to the integrated volatility.

In this more general framework, the RV of equation (19) converges uniformly in probability to the increment to the quadratic variation process defined above, as the sampling frequency of the returns approaches infinity:

$$\underset{\Delta \longrightarrow 0}{\text{plim}} \; RV_{t+1}(\Delta) = \int_{t}^{t+1} \sigma^2(s) ds + \sum_{t<s\leq t+1} k^2(s) \tag{24}$$

Thus, in the absence of jumps, the realized variation is consistent for the integrated volatility.

However, in general, the realized volatility will inherit the dynamics of both the continuous sample path process and the jump process.

Jumps

Making use of recent asymptotic results by Barndorff-Nielsen and Shephard (2004, 2006) that allow for separate (non-parametric) identification of the two components of the quadratic variation process, we can thus define the standardized *Realized Bipower Variation* measure as follows:

$$BV_{t+1}(\Delta) = \mu_1^{-2} \sum_{j=2}^{1/\Delta} |y_{t+j\Delta,\Delta}||y_{t+(j-1)\Delta,\Delta}| = \mu_1^{-2} \sum_{i=2}^{n_t} |y_{t,i}||y_{t,i-1}| \quad (25)$$

where $\mu_1 = \sqrt{2/\pi} = E(|Z|)$ denotes the mean of the absolute value of standard normally distributed random variable Z. Barndorff-Nielsen and Shephard (2004, 2006) show that

$$\operatorname*{plim}_{\Delta \longrightarrow 0} \quad BV_{t+1}(\Delta) = \int_t^{t+1} \sigma^2(s) ds \quad (26)$$

Jumps

Hence, combining the results in equations (26) and (24), the contribution to the quadratic variation process due to the discontinuities (jumps) in the underlying price process may be consistently estimated by

$$\underset{\Delta \longrightarrow 0}{\text{plim}} \quad RV_{t+1}(\Delta) - BV_{t+1}(\Delta) = \sum_{t<s\leq t+1} k^2(s) \qquad (27)$$

As nothing prevents the right hand-side of (27) from becoming negative in a given sample, Barndorff-Nielsen and Shephard (2004) suggest to impose a non-negativity truncation on the actual empirical jump measurements,

$$J_{t+1}(\Delta) = max[RV_{t+1}(\Delta) - BV_{t+1}(\Delta), 0] \qquad (28)$$

Models for Forecasting Realized Volatility: HAR-RV

Corsi (2009) recently proposed a class of volatility models that seem to successfully achieve the purpose of modelling the long memory behavior of volatility in a very simple and parsimoniously way.

In order to describe the HAR-RV model, we have to use the multi-period realized volatilities defined as the normalized sum of the one-period volatilities, as we did in (21),

$$RV_{t,t+h} = h^{-1}[RV_{t+1} + RV_{t+2} + ... + RV_{t+h}]. \tag{29}$$

Andersen et al. (2007) refer to these normalized volatility measures for $h = 5$ and $h = 22$ as the weekly and monthly volatilities, respectively. Moreover, by definition of the daily volatilities, $RV_{t,t+1} = RV_{t+1}$.

The daily *HAR-RV* model of Corsi (2009) may be expressed as

$$RV_{t,t+1} = \beta_0 + \beta_D RV_t + \beta_W RV_{t-5,t} + \beta_M RV_{t-22,t} + \epsilon_{t+1} \tag{30}$$

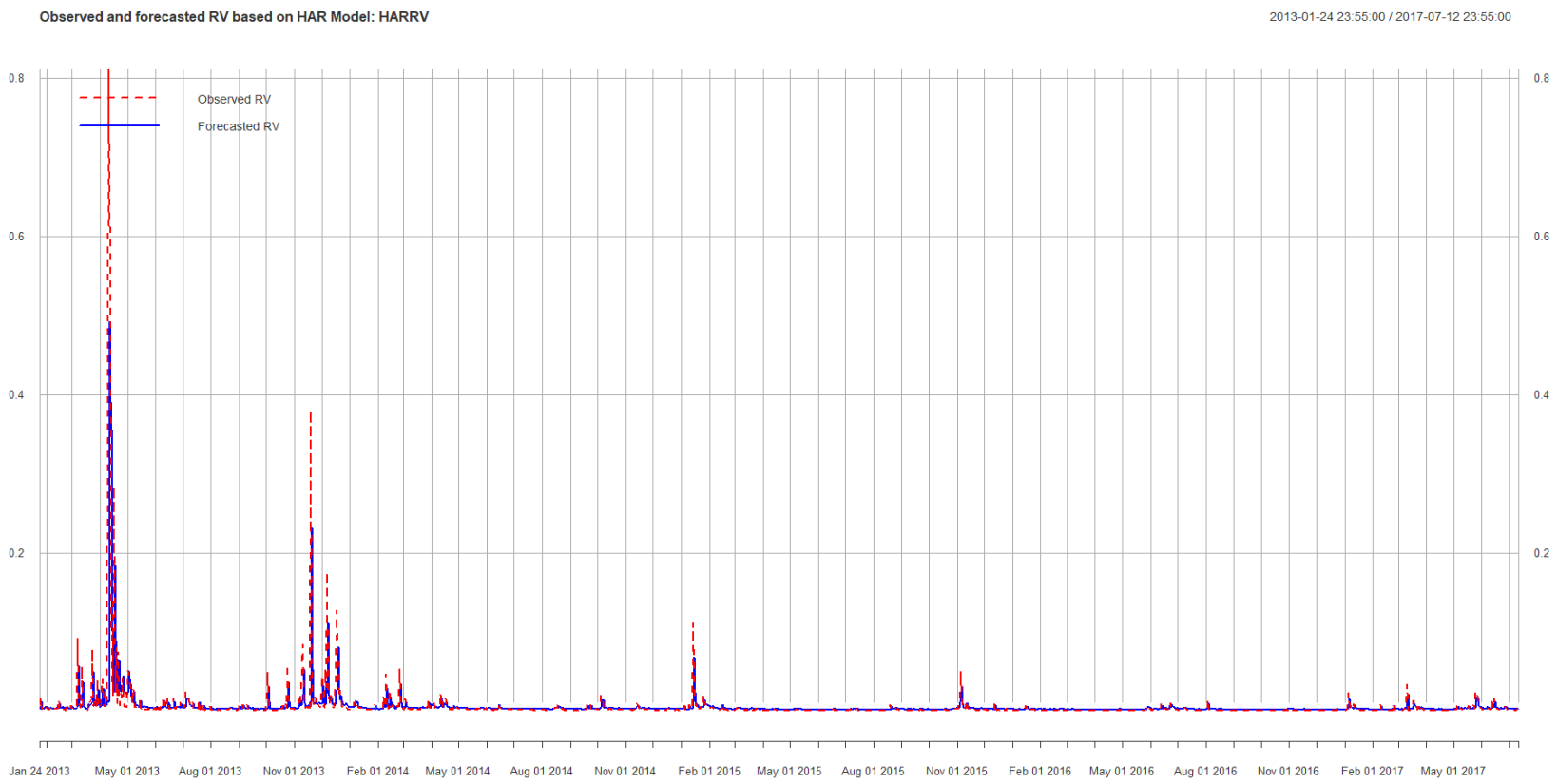
Realized Volatility: an example with R

```
# Download and aggregate data
library(highfrequency)
localbtcMXN <-bitcoincharts_single_download(name = "localbtcMXN.csv.gz")
btcdeEUR <-bitcoincharts_single_download(name = "btcdeEUR.csv.gz")
krakenEUR <-bitcoincharts_single_download(name = "krakenEUR.csv.gz")
localbtcINR <-bitcoincharts_single_download(name = "localbtcINR.csv.gz")
coinbaseUSD <-bitcoincharts_single_download(name = "coinbaseUSD.csv.gz")
data5<-list(localbtcMXN =localbtcMXN, btcdeEUR=btcdeEUR, krakenEUR=krakenEUR,
             localbtcINR=localbtcINR, coinbaseUSD=coinbaseUSD)
data_merged<-aggregate_merge_bictoincharts_data(data_list=data5)


# Realized Variance using 5 minutes sampling
rv <- rCov( rdata = data_clean$price_ts, align.by ="minutes", align.period =5,
            makeReturns=TRUE);
head(rv,2)
                              aa
2011-09-13 23:55:00 4.042090e-03
2011-09-14 23:55:00 5.802934e-03

# Realized Bipower Variation using 5 minutes sampling
rbpv = rBPCov( rdata = data_clean$price_ts, align.by ="minutes", align.period =5,
               makeReturns=TRUE);
head(rbpv,2)
                              aa
2011-09-13 23:55:00 0.0001555618
2011-09-14 23:55:00 0.0004933023
```

Realized Volatility: an example with R

```
library(highfrequency)
setwd("C:/Users/Dean/Downloads")
data_clean=readRDS("data_clean.rds")
dat<-data_clean$price_ts
dat<- dat["2013-01-02/2017-07-12"]
dat_ret <- highfrequency::makeReturns(dat)
btc_harrv<- highfrequency::harModel(data=dat_ret,periods=c(1, 5, 22),
                            RVest = c("rCov"), type="HARRV",h=1,transform=NULL)
summary(btc_harrv)

Call: "RV1 = beta0  +  beta1 * RV1 +  beta2 * RV5 +  beta3 * RV22"
Residuals:
     Min       1Q   Median       3Q      Max
-0.20307 -0.00180 -0.00147 -0.00087  0.79924

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
beta0  0.001584   0.000687   2.305  0.02128 *
beta1  0.598949   0.025207  23.762  < 2e-16 ***
beta2 -0.014474   0.040409  -0.358  0.72026
beta3  0.135749   0.051953   2.613  0.00906 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.02604 on 1627 degrees of freedom
Multiple R-squared:  0.3823,    Adjusted R-squared:  0.3812
F-statistic: 335.7 on 3 and 1627 DF,  p-value: < 2.2e-16

plot(btc_harrv)
```

Realized Volatility: an example with R



Chapter 10: Multivariate volatility modelling

Chapter 10: Multivariate volatility modelling

- Multivariate GARCH (MGARCH) models: VEC, BEKK, DCC
- Copula-GARCH models
- Realized Covariance models

DCC models

Engle (2002) proposes a model which is usually referred to as $DCC_E(S, L)$ and is defined as:

$$\Sigma_t = D_t R_t D_t \tag{31}$$

where D_t is a diagonal matrix with std. deviations, and

$$R_t = (diag Q_t)^{-1/2} Q_t (diag Q_t)^{-1/2} \tag{32}$$

where the $N \times N$ symmetric positive definite matrix Q_t is given by:

$$Q_t = \left(1 - \sum_{l=1}^{L} \alpha_l - \sum_{s=1}^{S} \beta_s\right) \bar{Q} + \sum_{l=1}^{L} \alpha_l u_{t-l} u'_{t-l} + \sum_{s=1}^{S} \beta_s Q_{t-s} \tag{33}$$

where $u_{it} = \varepsilon_{it}/\sqrt{\sigma_{ii,t}}$, $\bar{Q}$ is the $n \times n$ unconditional variance matrix of u_t, α_l (≥ 0) and β_s (≥ 0) are scalar parameters satisfying $\sum_{l=1}^{L} \alpha_l + \sum_{s=1}^{S} \beta_s < 1$, to have $Q_t > 0$ and $R_t > 0$. Q_t is the covariance matrix of u_t , since $q_{ii,t}$ is not equal to 1 by construction. Then, it is transformed into a correlation matrix by (32).

If $\theta_1 = \theta_2 = 0$ and $\bar{q}_{ii} = 1$ the CCC model is obtained.

Copula-GARCH models

It is possible to show that the CCC and DCC models can be represented as special cases within a more general copula framework, see Patton (2006a), Patton (2006b), Fantazzini (2008) and Fantazzini (2009c).

Particularly, the multivariate normal likelihood can be decomposed by considering the joint normal density function as the product of a *normal copula* with correlation matrix $\Sigma = R_t$ together with *normal marginals*:

$$f^{normal}(x_1, \ldots, x_n) = c^{normal}(F_1^{normal}(x_1), \ \ldots, \ F_n^{normal}(x_n); R_t) \cdot \prod_{i=1}^{n} f_i^{normal}(x_i) \tag{34}$$

where F_i^{normal} is the normal cumulative density function.

If we consider a general model for the conditional means and variances, the two models can be restated as follows:

$$\begin{aligned} X_t &= E[X_t|\Im_{t-1}] + D_t z_t \\ z_t &\sim H(z_1, \ldots, z_n) \equiv C^{normal}(F_1^{normal}(z_1), \ldots, F_n^{normal}(z_n); R_t) \end{aligned}$$

where $D_t = diag(\sigma_{11,t}, \ldots \sigma_{nn,t})$, $\sigma_{ii,t}^2$ follows a univariate GARCH model and the Sklar's Theorem was used.

Copula-GARCH models

It is clear from the previous section that the copula approach enables us to consider far more general cases than the normal CCC and DCC.

A multivariate model that allows for marginal skewness, kurtosis and normal dependence can be expressed as follows:

$$\begin{aligned} X_t &= \mathbb{E}\{X_t|\mathfrak{F}_{t-1}\} + D_t z_t \\ z_t &\sim H(z_1,\ldots,z_n) \equiv C^{normal}(F_1^{Skewed-t}(z_1),\ldots,F_n^{Skewed-t}(z_n);R_t) \end{aligned}$$

where $F_i^{Skewed-t}$ is the cumulative distribution function of the marginal Skewed-t, and R_t can be made constant or time-varying, as in the standard CCC and DCC models, respectively.

If the financial assets present symmetric tail dependence, we can use a Student's t copula, instead,

$$\begin{aligned} X_t &= \mathbb{E}\{X_t|\mathfrak{F}_{t-1}\} + D_t z_t \\ z_t &\sim H(z_1,\ldots,z_n) \equiv C^{Student's\,t}(F_1^{Skewed-t}(z_1),\ldots,F_n^{skewed-t}(z_n);R_t,\nu) \end{aligned}$$

where ν are the Student's t copula degrees of freedom

Example with R

In case of large portfolios and dynamic copula models, it is better to employ the **rugarch** and the **rmgarch** packages:

```
## Example 3: rugarch and rmgarch packages
library(Quandl); library(TTR)
# Download cryptocurrencies from Quandl
BTC <- Quandl::Quandl("BITFINEX/BTCUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
LTC <- Quandl::Quandl("BITFINEX/LTCUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
ETH <- Quandl::Quandl("BITFINEX/ETHUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
BTC <- BTC [, "Last", drop=F]
ETH <- ETH [, "Last", drop=F]
LTC <- LTC [, "Last", drop=F]
ETH.BTC.LTC = merge(ETH,BTC,LTC)
ETH.BTC.LTC = na.omit(ETH.BTC.LTC)
colnames(ETH.BTC.LTC) =c("ETH","BTC","LTC")
R <- TTR::ROC(ETH.BTC.LTC, na.pad = FALSE)

### Model specification:VAR(2)-GARCH(1,1) with skewed marginals +
### + t-copula with DCC(1,1) for the correlation matrix
# Marginal specification
uspec <- ugarchspec(variance.model=list(garchOrder=c(1,1),
                    model = "sGARCH"),  distribution.model = "sstd")
# copula specification
mspec <- cgarchspec(uspec = multispec( replicate(ncol(R), uspec) ), VAR = TRUE,
          robust=FALSE,lag=2,lag.max=NULL,lag.criterion=c("AIC","HQ","SC","FPE"),
          external.regressors=NULL,robust.control=list("gamma"=0.25,"delta"=0.01,
          "nc"= 10,"ns" = 500), dccOrder = c(1,1), asymmetric = FALSE,
          distribution.model = list(copula = c("mvnorm", "mvt")[2],
          method = c("Kendall", "ML")[2], time.varying = TRUE,
          transformation = c("parametric", "empirical", "spd")[1]))
fit1 <- cgarchfit(mspec, data = R, fit.control = list(eval.se=TRUE))
```

Example with R

```
fit1

*---------------------------------------------------*
*                  Copula GARCH Fit                 *
*---------------------------------------------------*
Distribution         :  mvt
DCC Order            :  1 1
Asymmetric           :  FALSE
No. of Parameters    :  42
[VAR GARCH DCC UncQ]: [21+15+3+3]
No. of Series        :  3
No. of Observations   :  461
Log-Likelihood       :  2886.76
Av.Log-Likelihood    :  6.262
Optimal Parameters
---------------------------------------------------
                Estimate  Std. Error  t value Pr(>|t|)
[ETH].omega     0.000258    0.000162   1.5897 0.111895
[ETH].alpha1    0.273238    0.107150   2.5500 0.010771
[ETH].beta1     0.725762    0.093477   7.7641 0.000000
[ETH].skew      1.276437    0.060990  20.9286 0.000000
[ETH].shape     3.665482    0.639309   5.7335 0.000000
[BTC].omega     0.000028    0.000017   1.5952 0.110678
[BTC].alpha1    0.177897    0.034455   5.1631 0.000000
[BTC].beta1     0.821103    0.048919  16.7849 0.000000
[BTC].skew      1.038374    0.041898  24.7831 0.000000
[BTC].shape     2.949946    0.253263  11.6478 0.000000
[LTC].omega     0.000066    0.000038   1.7297 0.083691
[LTC].alpha1    0.163358    0.047341   3.4507 0.000559
[LTC].beta1     0.835642    0.089694   9.3165 0.000000
[LTC].skew      1.277271    0.067492  18.9248 0.000000
[LTC].shape     2.680948    0.303606   8.8304 0.000000
[Joint]dcca1    0.061210    0.020727   2.9532 0.003145
[Joint]dccb1    0.896533    0.040142  22.3342 0.000000
[Joint]mshape   7.402293    1.572778   4.7065 0.000003
```

Example with R

```
Information Criteria
---------------------
Akaike       -12.446
Bayes        -12.284
Shibata      -12.449
Hannan-Quinn -12.382

# If you need the parameters of the VAR(2) model for the conditional mean, type:
fit1@model$varcoef
           ETH.l1       BTC.l1       LTC.l1       ETH.l2      BTC.l2       LTC.l2
ETH  0.1259141551  0.086756895  0.018058618 -0.009809315  0.07699290 -0.145510244
BTC -0.0008673604 -0.103929844  0.019590329 -0.018561690 -0.02332346 -0.008141854
LTC  0.0104835675  0.003802743 -0.005066713 -0.041902091 -0.05611200  0.016245733
            const
     0.006019725
     0.004393645
     0.005779316
```

Chapter 11: Market Risk Management

Chapter 11: Market Risk Management

- Risk Measures: VaR and Expected Shortfall
- Backtesting market risk measures
- Examples with univariate and multivariate volatility models

Risk Measures

If we define $\Psi(\Delta P_t)$ the risk measure of ΔP_t, ADEH(1999) affirm that $\Psi(\Delta P_t)$ is a coherent risk measure if it has the following properties:

- **Translation Invariance**: given a random variable ΔP_t, the risk-free title ΔG and a generic constant $\theta_G \in \mathbb{R}$, then

$$\Psi(\Delta P_t + \theta_G \Delta G) = \Psi(\Delta P_t) - \theta_G \tag{35}$$

- **Sub-additivity**: given two price variations (or returns) $\Delta P_{t,1}$ and $\Delta P_{t,2}$, it holds that

$$\Psi(\Delta P_{t,1} + \Delta P_{t,2}) \leq \Psi(\Delta P_{t,1}) + \Psi(\Delta P_{t,2}) \tag{36}$$

- **Positive Homogeneity**: given ΔP_t and a not negative constant λ, then:

$$\Psi(\lambda \Delta P_t) = \lambda \Psi(\Delta P_t) \tag{37}$$

- **Monotonicity**: given two price variations (or returns) $\Delta P_{t,1}$ and $\Delta P_{t,2}$, such that $\Delta P_{t,1} \leq \Delta P_{t,2}$ then

$$\Psi(\Delta P_{t,2}) \leq \Psi(\Delta P_{t,1}) \tag{38}$$

Risk Measures

The Expected Shortfall measures the average of the worst α results that we can get from an investment, where α can be a percentage or, even better, the percentile of the returns distribution.

Formally, *The Expected Shortfall (ES_α) is the simple arithmetic mean of all the losses that we have with probability equal or smaller than* α:

$$ES_\alpha = -\frac{1}{\alpha}\int_0^\alpha F^{-1}(\Delta P_t)d\Delta P_t \tag{39}$$

While the properties of translation invariance, positive homogeneity and monotonicity follow easily from the properties of quantiles and the previous definition, the subadditivity is more complicated to prove and we refer to Acerbi and Tasche (2002) for more details.

$\rightarrow$ The Value-at-Risk is not a coherent risk measure becasue it is not always sub-addittive (except for ellyptical distributions).

Example with R

```
# 2) ======================  COPULA-VAR(2)-GARCH(1,1) ======================
# ========================================================================
# Model specification: VAR-GARCH-SSTD and DCC(1,1)-t-copula
uspec = ugarchspec(mean.model = list(armaOrder = c(0,0)), variance.model =
         list(garchOrder=c(1,1), model="sGARCH"), distribution.model="sstd")
spec1 = cgarchspec(uspec = multispec( replicate(m, uspec) ), VAR = TRUE,
      robust=FALSE, lag=2,lag.max= NULL,lag.criterion=c("AIC","HQ","SC","FPE"),
         external.regressors = NULL, robust.control = list("gamma" = 0.25,
         "delta"=0.01,"nc"=10,"ns"=500), dccOrder=c(1,1), asymmetric = FALSE,
         distribution.model = list(copula = c("mvnorm", "mvt")[2],
         method = c("Kendall", "ML")[2], time.varying = TRUE,
         transformation = c("parametric", "empirical", "spd")[1]))
tic = Sys.time(); set.seed(123)
#Set up the parallel estimation
cl <- parallel::makePSOCKcluster(7)
parallel::clusterEvalQ(cl = cl, library(rmgarch))
parallel::clusterEvalQ(cl = cl, library(xts))
parallel::clusterExport(cl, varlist = c('R', 'w', 'spec1', 'n','backtest.length',
                                        'v_alpha'),   envir = environment())
mod = clusterApply(cl = cl, (n-backtest.length):(n-1), fun = function(i) {
  fit1 = cgarchfit(spec1, data = R[1:i,], fit.control = list(eval.se=FALSE))
  sim1 = cgarchsim(fit1, n.sim = 1, m.sim = 10000, startMethod = "sample",
                   only.density = TRUE)

  # Compute the realized and simulated weighted returns
  realized<- as.numeric(R[i+1,] %*% w)
  simx <- t(sapply(sim1@msim$simX, FUN = function(x) x[1,]))
  simret<- simx %*% w
  VaR.alpha <-quantile(simret, probs = v_alpha)
  ES.alpha = mean(simret[simret <= VaR.alpha[5]])
  ans = list(realized = realized, VaR.alpha = VaR.alpha, ES.alpha=ES.alpha)
  return(ans)
})
stopCluster(cl)
toc = Sys.time()-tic
realized = VaR.alpha = ES.alpha= NULL
for (i in 1:length(mod)) {
  realized = rbind(realized, mod[[i]]$realized)
  VaR.alpha = rbind(VaR.alpha, mod[[i]]$VaR.alpha)
  ES.alpha = rbind(ES.alpha, mod[[i]]$ES.alpha)
}
```

Example with R

```
# ====================== VaR BACKTESTING ====================
# Compute VaR
m_VaR <-VaR.alpha
# Test each VaR using UC and CC VaR tests
test_VaR_mat = NULL
for (i in 1: length(v_alpha)){
  test_Var <- VaRTest(alpha=v_alpha[i], actual=realized, VaR=m_VaR[,i])
  test_VaR_mat <-  rbind(test_VaR_mat, cbind(test_Var$uc.LRp,  test_Var$cc.LRp,
                                             test_Var$actual.exceed))
}
colnames(test_VaR_mat)= c("UC pvalue", "CC pvalue", "Actual exceed.")
test_VaR_mat
            UC pvalue  CC pvalue Actual exceed.
      [1,] 0.38190642 0.66060336              4
      [2,] 0.21487449 0.40677691              8
      [3,] 0.00654602 0.01459809             16
      [4,] 0.01044855 0.01776247             19
      [5,] 0.01382659 0.04828762             22
# Compute the number of violations in each cell,
n_cell<-c(test_VaR_mat[,3], backtest.length) - c(0, test_VaR_mat[,3])
#and test all VaR jointly using the multinomial VaR backtest by Kratz et al.(2018)
theo_cell <- c(v_alpha, 1) - c(0, v_alpha)
 XNomial::xmonte(n_cell, theo_cell, detail=2)
   P value (LLR) = 0.12576 ± 0.001049
   1e+05  random trials
   Observed:  4 4 8 3 3 478
   Expected Ratio:  0.005 0.005 0.005 0.005 0.005 0.975
# ====================== ES BACKTESTING ====================
es.multi = ES.alpha
# Compute the Z2 test of eq. (11.15)
real <- realized
VaR2.5 <- m_VaR[,5]
# 2nd test by Acerbi and Szekely (2014)
Z2 <- sum( (real/(-es.multi*length(es.multi)*es.alpha))
           *(real < VaR2.5) ) + 1
Z2
   [1] -0.7027364
# ES tests by Bayer and Dimitriadis (2018)
esback::esr_backtest(r = real, e = es.multi, alpha = 0.025, B=199)
Asymptotic  Bootstrap
 0.0769853  0.1356784
```

Chapter 12: Portfolio Management

Chapter 12: Portfolio Management

- A review of the classics: Markowitz mean-variance analysis
- Tail-based risk optimal portfolios: mean-VaR, mean-CVaR, mean-CDaR
- Other risk-optimal portfolios: MAD, Minimax, LPM, Omega
- A simple portfolio diversification rule using online data

Mean-CDaR portfolios

The Conditional Drawdown-at-Risk (CDaR) was proposed by Pardalos et al. (2004) and Chekhlov et al. (2005).

A portfolio's drawdown at time t is the difference between the maximum uncompounded portfolio value before time t and its current value at t: for example, if the latest value of our portfolio is 20 million and the maximum value of our portfolio in the past was 40 million, the absolute drawdown would be 20 million, while the relative drawdown 50%.

This measure is of great importance in the portfolio management industry: a large drawdown can force a client to withdraw his mandate and the portfolio manager would lose his management fees. Formally speaking, the drawdown function for a portfolio is given by:

$$D(\mathbf{w}, t) = \max_{0 \leq \tau \leq t} \{v(\mathbf{w}, \tau)\} - v(\mathbf{w}, t)$$

where $v(\mathbf{w}, t)$ is the uncompounded portfolio value at time t. As Krokhmal et al. (2002) highlighted, the drawdown accounts not only for the number of losses over a time interval but also for their sequence so that the drawdown is a loss measure with memory.

Mean-CDaR portfolios

Using the drawdown function, we have three functional risk measures:

- *Maximum drawdown*: $\text{MaxDD}(\mathbf{w}) = \max_{0 \leq t \leq T}\{D(\mathbf{w}, t)\}$
- *Average drawdown*: $\text{AveDD}(\mathbf{w}) = \frac{1}{T}\int_0^T D(\mathbf{w}, t)dt$
- *Conditional draw-down at risk at confidence level* α: let ζ_α be the threshold that is exceeded by $(1-\alpha)T$ drawdowns, and if $(1-\alpha)T$ is an integer number, then

$$CDaR(\mathbf{w})_\alpha = \frac{1}{(1-\alpha)T}\int_\Omega D(\mathbf{w}, t)dt$$

where $\Omega = \{t \in [0, T] : D(\mathbf{w}, t) \geq \zeta_\alpha\}$. Instead, if $(1-\alpha)T$ is not an integer number, then CDaR is a linear combination of the threshold and the drawdowns strictly exceeding this threshold, similarly to what we saw with the CVaR for general distributions not necessarily continuous:

$$CDaR(\mathbf{w})_\alpha = \min_\zeta \left\{\zeta + \frac{1}{(1-\alpha)T}\int_0^T [D(\mathbf{w}, t) - \zeta]^+ dt\right\}$$

where $[a]^+ = \max(0, a)$. For example, the $\text{CDaR}(\mathbf{w})_{0.95}$ can be interpreted as the average of the 5% largest drawdowns.

Mean-CDaR portfolios

Note that the maximum drawdown is based on one worst case event that took place in the examined time sample, which may not reflect the future sample path: particularly, a very large maximum drawdown may force risk managers to be far more conservative than needed.

Instead, the average drawdown considers all drawdowns, thus potentially masking some large drawdowns. The CDaR measure solves most of these problems but still assumes that the past financial history will be similar to the future financial path.

Chekhlov et al. (2005) proposed a portfolio optimization which maximizes the expected value of the uncompounded cumulative portfolio rate of return at the final time moment T subject to a constraint on a drawdown measure (AveDD, or MaxDD, or CDaR_α) which should not be larger than a proportion γ of the initial capital.

Chekhlov et al. (2005) also proved that drawdown risk measures satisfy the properties of *deviation measures*, that is (1) nonnegativity, (2) insensitivity to constant shift, (3) positive homogeneity, and (4) convexity.

An example with R:

The package **FRAPO** acconpanies the textbook by Pfaff (2016) and contains several functions to compute constrained (long-only) maximum draw-down, average draw-down, and conditional draw-down at risk portfolios, which are described in details in section 12.5.2 in Pfaff (2016)

```
library(Quandl); library(TTR); library(FRAPO);
# Download cryptocurrencies from Quandl
BTC <- Quandl::Quandl("BITFINEX/BTCUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
LTC <- Quandl::Quandl("BITFINEX/LTCUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
ETH <- Quandl::Quandl("BITFINEX/ETHUSD", type="xts",start_date="2016-03-14",
                      end_date="2017-07-02")
BTC <- BTC [, "Last", drop=F]
ETH <- ETH [, "Last", drop=F]
LTC <- LTC [, "Last", drop=F]
ETH.BTC.LTC = merge(ETH,BTC,LTC)
ETH.BTC.LTC = na.omit(ETH.BTC.LTC)
colnames(ETH.BTC.LTC) =c("ETH","BTC","LTC")

# 1) Portfolio optimization with maximum drawdown constraint:
# the argument MaxDD sets the upper bound of the maximum drawdown(in % of the capital)
# the argument softBudget allows the budget constraint to be a soft constraint,
# i.e. the sum of the weights can be less than one
port_MaxDD<-PMaxDD(PriceData=coredata(ETH.BTC.LTC),MaxDD=0.3,softBudget=TRUE)
port_MaxDD
Optimal weights for portfolio of type:
maximum draw-down
   ETH    BTC    LTC
0.0036 0.1910 0.0270
# Interestingly, to satisfy the DD constraint we should not invest our full capital,
# and this is true for all possible values of MaxDD
```

An example with R:

```
# 2) Portfolio optimization with average draw down constraint
# the argument AveDD sets the upper bound of the average portfolio
# drawdown (in %)
port_AveDD <- PAveDD(PriceData=coredata(ETH.BTC.LTC),AveDD=0.3,softBudget=F)
port_AveDD
Optimal weights for porfolio of type:
average draw-down
   ETH    BTC    LTC
0.3097 0.6087 0.0816
# No need of a softBudget here, unless the AveDD constraint is < 0.24

# 3) Portfolio optimization with CDaR constraint(95%). The argument alpha
# specifies the confidence level, while 'bound' sets the upper bound of
# the CDaR function
port_CDaR <- PCDaR(PriceData = coredata(ETH.BTC.LTC), alpha = 0.95,
                bound = 0.3,  softBudget = TRUE);  port_CDaR;
Optimal weights for porfolio of type:
conditional draw-down at Risk
  ETH   BTC   LTC
0.042 0.180 0.027

# Also here, to satisfy the CDaR contraint, we should not invest our full
# capital,  and this is true for all possible values of `bound`
```

Chapter 13: Credit Risk Management

An introduction to classical credit risk management

The components of credit risk are defined as follows:

1. **Probability of Default (PD)**, which can be examined considering:

- the simple two events: (1) not insolvency and (2) insolvency of the debtor. This is sometimes called *pure default risk*;
- the deterioration of the credit rating, that implies an increase of the probability of default. This is also known as *migration risk*, and the default risk is the last "absorbing" state

2. **The Loss Given Default (LDG)**: when a default takes place not the whole credit is necessarily lost, due to collateral or guarantees that allow to recover at least part of the credit.

$\rightarrow$ This brings us to the so-called *Recovery Rate* (RR) of the obligor, which is the percentage which can be recovered in case of insolvency, given by $RR = 1 - LGD$.

An introduction to classical credit risk management

3. **Exposure At Default (EAD)**: it represents the total amount of the payment obligations of the obligor which would enter the bankruptcy proceedings if a credit event occurred (default or migration). It is evident that if we want to determine the EAD we have to consider the the whole position subject to credit risk including:

- *Loans*
- *Bonds*
- *Guarantees released to clients*
- *OTC Derivatives*
- *Credit Derivatives*

4. **Default dependence and/or migration dependence**. The measurement of such dependence is very complex: historical data are few and even if large datasets were available, it would emerge that simultaneous default simultaneous are very rare. Financial research is still needed with this regard.

Credit risk management FOR cryptocurrencies

The MtGox bankruptcy showed that one of the main aspects of credit risk management with cryptocurrencies: the *default probability* of the online-exchange used to trade cryptocurrencies. Why?

If we employ the classical framework used for measuring credit risk, the Exposure At Default (EAD) is easy to compute and is represented by the amount deposited in the exchange (both FIAT and crypto-currencies) which is fixed and certain.

$\Rightarrow$ The Loss Given Default (LGD) for crypto-exchanges is extremely high: Moore et al. (2018) examined 80 Bitcoin exchanges established between 2010 and 2015, and found that *38 have since closed*; of these 38,

- five fully refunded customers,
- five refunded customers only partially,
- six exchanges did not reimburse anything,
- while there is no information for the remaining 22 exchanges.

Credit risk management FOR cryptocurrencies

These numbers are rather staggering and show that closed crypto-exchanges imply LGDs comparable to subordinated bonds if not public shares, see Shimko (2004) for more details about classical LGDs estimated using the Moody's Default Risk Service Database

$\Rightarrow$ A risk-averse investor (and credit risk manager) would definitely not exaggerate if he/she set the LGD for closed crypto-exchanges to 100%, that is a recovery rate of zero.

Given this discussion, the computation of the probability of default/closure clearly becomes the key issue when measuring credit risk with crypto-exchanges.

Only two papers who developed models for the default probability of cryptoexchanges - Moore et al. (2013) and Moore et al. (2018): the main reason is the difficulty in finding data of closed exchanges, without which any proper empirical analysis is not viable.

Credit risk management FOR cryptocurrencies

Summarizing Moore et al. (2013), they find that

- a high transaction volume decreases the exchange probability of default,
- while a security breach increases this probability but this latter effect is not statistically significant.
- The anti-money laundering indicator shows no correlation with the hazard rate.

Instead, a separate logistic regression shows that

- transaction volume increases the probability of a security break,
- whereas the number of months the exchange was open has no significant effect.

However, be careful of these results because,

- the number of regressors is small and a large part of the model randomness cannot be explained.
- the dataset is rather small and some regressors like the long-term average of the transaction volume may need to be changed

Credit risk management FOR cryptocurrencies

Moore et al. (2018) extended the work by Moore et al. (2013) considering transactions between 2010 and March 2015 and up to 80 exchanges.

They built quarterly indicators and estimated a panel logit model with an expanded set of explanatory variables.

Some previous results confirmed + some interesting new findings:

- a security breach increase 13.5 times the odds that the exchange will close that same quarter,
- while doubling of the daily transaction volume determines a 12% decrease in the odds that the exchange will shut down that quarter.
- New finding: *exchanges who get most of their transaction volume from fiat currencies which are traded by few other exchanges (mono- or duopoly currencies) are 91% less likely to close* than other exchanges who trade fiat currencies with higher competition.
- A time trend is significant and decreases the probability of closure,
- The anti-money laundering indicator and 2-factor authentication not significant

Given this evidence, where do we go from here?

The amazing growth of crypto-exchanges in the last years cannot hide the fact that these busi nesses mostly belong to the large family known as *Small and Medium-sized Enterprises (SMEs)*, and which represents the vast majority of businesses in most countries.

There is no a unique definition of SME worldwide, and each country has its own legal definition which depends on the number of employees, annual sales, assets, business sector, or any combination of these, see *en.wikipedia.org/wiki/Small_and_medium-sized_enterprises*

for a quick review.

As anybody who has dealt with credit risk management for SMEs, modelling and forecasting the PD of SMEs is very difficult. There are two main reasons:

- lack of data
- poor financial reporting.

Given this evidence, where do we go from here?

- Forecasting the PD of exchanges: Expert and credit rating systems
- Forecasting the PD of exchanges: Credit Scoring Systems
- Forecasting the PD of exchanges: Classical and Bayesian Panel Models
- Forecasting the PD of exchanges: Machine learning
- Forecasting the PD of quoted exchanges: Merton's Model
- Forecasting the PD of quoted exchanges: the ZPP
- Forecasting the probability of death of coins with the ZPP
- Model Evaluation: ROC, AUC and Loss Functions

Conclusions: challenges ahead

Conclusions: challenges ahead

- Short-term challenges: energy consumption and max of daily number of transactions
- Medium and long-term challenges: quantum computing