Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques
Universitat de Barcelona

FINANCIAL TIME SERIES OBTAINED BY AGENT-BASED SIMULATION

Pol Ferrando Hernández

Director: Josep Fortiana Gregori
Realitzat a: Departament de Probabilitat, Lògica i Estadística. UB
Barcelona, 30 de gener de 2015

Contents

Acknowledgments

First and foremost, I would like to express my sincerest gratitude to my advisor, Dr. Josep Fortiana Gregori for the continuous support of my TFG. His guidance helped me in all the time of research and writing of this work. Special thanks to Elder Silva for sharing many useful information too. Finally, I also dedicate it to my family, that always believe in me and made me who I am today.

Chapter 1

Introduction

Economic news often talks about growths and drops of indexes and individual stocks, but many people (including me before I did this work) do not understand neither how the stock market works nor what causes its price fluctuations.

Stock markets are complex systems. In empirical sciences, a common strategy used to study a real system consists in making simplified models that keep their main features, and analyze them in order to understand further the system dynamics. There is a type of models known as *agent-based models* that are used to simulate complex systems by creating software objects (called *agents*) whose behavior have global consequences for the system. This concept allows modelers to connect the micro-level of individuals with the macroscopic patterns, what is essential to understand systems interactions. One example of agent-based environment and programming language is NetLogo, created by Uri Wilensky in 1999.

Since individual investors' decisions control the dynamics of stock markets, it seems reasonable to treat the stock market as if it is a dynamic system of interacting agents, that will represent investors. The collective behavior of these investors, each of which acts independently, produces prive movements. Based on Silva's (2014) *Collective behavior in the Stock Market* model, we have designed and implemented a model for the evolution of a very simple market, with a single asset price, using the NetLogo environment.

On the other hand, companies' share prices form *time series*. Statistics supplies powerful tools and methods to understand the processes behind time series, make a model of them and, furthermore, forecast future values based on the current data. Hence, financial time series analysis plays an important role in investments strategies and other economical applications. We are going to describe the main methods and models used for this kind of series, but there is so much bibliography on the statistical treatment of financial series; see, for example, Tsay (2005).

This project has been developed with many objectives. First, we want to introduce agent-based modeling, explain in which cases is useful and, furthermore, supply the basis of NetLogo programming language so that when the reader reaches the end of this work, he or she should be able to use and implement simple codes on NetLogo. Second, this work tries to teach step by step how to build an statistical model to fit given data. Also, we want to find out about distinctive features of financial time series and most important models used to fit them: ARMA and ARCH processes. Third, we attempt to understand several economical concepts and some common behaviors exhibited by investors. Finally, the main purpose of this project is to reproduce the evolution of a single asset price by an agent-based model, whose output series of returns will be analyzed viewed as a financial time series and will be shown to fit an ARCH model.

In summary, this final project deals with *agent-based modeling* and *time series analysis* in the framework of finance. In chapter 2, we expound agent-based computational economics and, then, we describe the Netlogo environment and the main features of its programming language in chapter 3. Also, the stock market model that we have created is explained in chapter 4, as well as its hypotheses and weaknesses. In chapter 5, we define and explain all the fundamental concepts, methods and statistical models that we need to analyze the simulated series of returns in chapter 6. In the end, the conclusions of the work are described and possible works that would complement this study are discussed.

To finish this introduction, I would want to explain my personal motivation to choose this subject as a final project. Basically, I sought a work about time series analysis because I wanted to go into detail about this area and apply its concepts to a real analysis. On the other hand, I am interested in the stock market and decision making under uncertainty, so I chose this subject to learn more about statistical methods applied to a real problem: investments.

Chapter 2

Agent-based computational economics (ACE)

According to Tesfatsion (See Ref. [4]), *complex systems* are composed of interacting units whose interactions produce properties that are not properties of the individual units themselves. Furthermore, a complex system is *adaptive* if it includes reactive units, that is, units capable of exhibiting different behaviors in reaction to changed environmental conditions.

Agent-based computational economics (ACE) is the computational study of economic processes modeled as complex adaptive systems. Units are called *agents*, and in this case, each agent is an encapsulated piece of software that includes both data and behavioral methods. For example, agent's data may be its attributes, while agent's behavioral methods may be its strategies or its algorithm for updating strategies. This encapsulation into agents attempts to achieve a more realistic representation of real-world systems.

Once the model is implemented, the modeler must specify the initial state of the economic system by setting each agent's initial data and behavioral methods. This will be the last interference, because an ACE model must be *dynamically complete*, that is, the simulated economic process must be able to develop over time based on agents interactions without further interventions from the modeler.

In summary, an ACE model is essentially a collection of procedures encapsulated into the methods of agents constituting a computationally constructed world of interacting entities.

Chapter 3

NetLogo

3.1 What is NetLogo?

NetLogo is a programmable modeling environment for doing agent-based simulations. It was created by Uri Wilensky in 1999 and it is used for modeling complex systems and studying their behavior over time.

Agents are single beings which act independently from others. Agent-based models consist of multiple agents (that could be from different kinds) interacting with each other if it is necessary. When the model runs, agents behave according to their rules generating a dynamic system. For example, a model for sheep predation by wolves may consist of three kinds of agents: wolves, sheeps and grass. These agents would act according to different rules depending on their types: wolves would try catching sheeps to eat, sheeps would move around the landscape eating grass where it would be, and grass would not do anything but grow again after a period of time. As consequence of agents behavior, maybe wolves or sheeps become extinct, or maybe the number of sheeps and the number of wolves tend to be constant (despite fluctuations). That is the point of agent-based models: simulating systems which involve many agents (even hundreds) and watching results of their interaction.

NetLogo lets us create agent-based models, so we could create hundreds or thousands of agents, give them instructions and after that watch their collective behavior as result of all operating independently at the same time. This allows exploring the connection between individual actions and macro-level patterns emerged from agents interaction.

In addition, its flexibility lets users do simulations with existent models, modify the models code based on their interests or create their own models. These two last actions require knowledge of NetLogo programming language, which we are going to talk about later in this chapter.

In summary, NetLogo allows us to make simulations for modeling phenomena that involve multiple agents operating independently. Because of this, it is used in disciplines as diverse as psychology, computing sciences, biology, physics or economics.

3.2 The NetLogo world

The NetLogo world is the place where agents remain. There are four types of agents in NetLogo:

- **Patches**. The world is a two dimensional grid of patches, so we can say that each patch is a square piece of "ground". Patches cannot move and they have coordinates. The patch at coordinates (0, 0) is called the origin and the coordinates of the other patches are the horizontal and vertical distances from this one. These coordinates are always integers.
- **Turtles** are agents that move around in the world while they follow instructions. They have coordinates too, but in this case they are not necessary integers because each turtle can be positioned at any point within its patch. There are no turtles by default, they have to be made by patches or by the observer. The name *turtles* is due to the fact that NetLogo is a dialect of the Logo language, which uses the nomenclature of "turtles".
- **Links** are agents that connect two turtles. If either turtle dies, the link dies too. They do not have coordinates.
- The **observer** is an external agent. It is not represented in the world but it can interact with the world creating or destroying agents, assigning properties to the agents, etc.

When we open NetLogo, our computer screen looks like this:

As we see at the top of Figure 3.1, there are three tabs labeled *Interface*, *Info* and *Code*. Only one tab at a time can be visible, but you can switch between them by clicking on the pertinent one.

First tab (*Interface*) is where users watch models run. It includes several options to control model and interface settings, a *Command center* to input commands and a customizable interface where users can add elements such as plots and monitors.

The second tab (*Information*) looks like Figure 3.2. It lets modelers add information about the purpose of the model, how to run it and suggestions to extend it, among others. Note that NetLogo is thought to create a community of modelers that share their models, making it easy for others to see, review, create variations and comment on their work[1]. Therefore, this information is useful for other users

[1]All models uploaded by other NetLogo users can be downloaded from Modelling Commons: http://modelingcommons.org/account/login

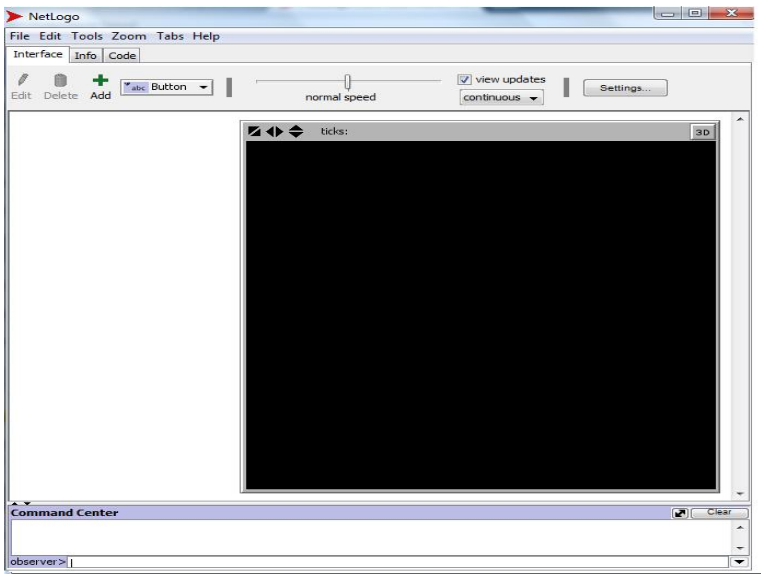


Figure 3.1: NetLogo interface.

more than for the own modeler.

Finally, the Code tab is where the code for the model is stored. It includes commands that will be used repeatedly while models will be running. In Section 3.4 we will talk about NetLogo programming language features in depth.

3.3 Interface tab

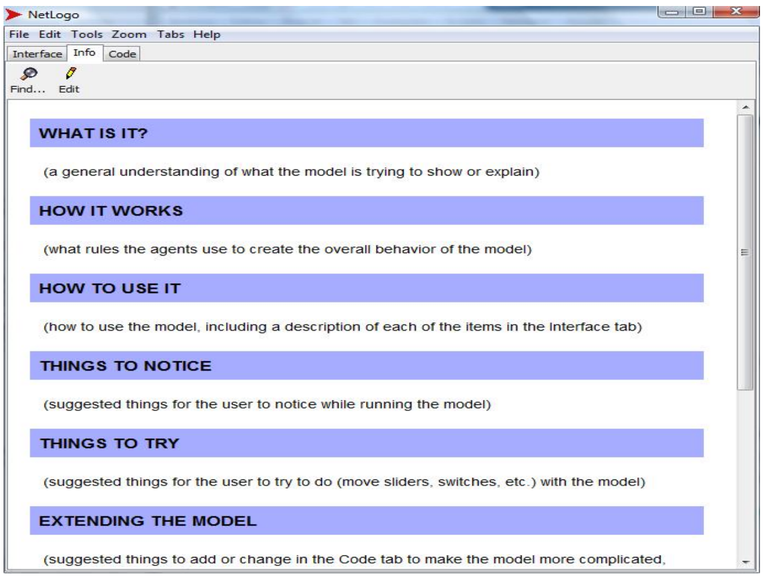
1. The **world** is the place where the agents will be shown.

2. **Model settings**. The *Settings...* button allows users to change the model settings: the size and the boundaries of the world, the patch size and the frame rate, among others.

3. **Control panel**.

   - Simulation speed. The slider lets user control how fast the model runs.
   - View updates (continuous or on ticks). Only if this check box is selected, the view of the world will change during the simulation. There are two update options: "continuous" and "on ticks". The first updates the view many times a second while the second one updates it when the tick counter advances.

4. **Command center**. This is the place where commands can be input directly, without being added into the Code tab. This is useful for inspecting and manipulating agents at a particular time, but it is laborious for commands that are used many times.



WHAT IS IT?

(a general understanding of what the model is trying to show or explain)

HOW IT WORKS

(what rules the agents use to create the overall behavior of the model)

HOW TO USE IT

(how to use the model, including a description of each of the items in the Interface tab)

THINGS TO NOTICE

(suggested things for the user to notice while running the model)

THINGS TO TRY

(suggested things for the user to try to do (move sliders, switches, etc.) with the model)

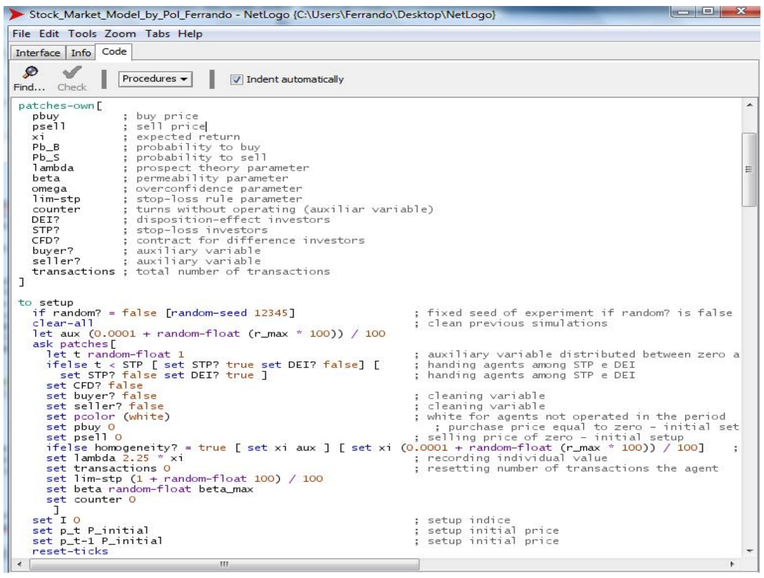
EXTENDING THE MODEL

(suggested things to add or change in the Code tab to make the model more complicated,

Figure 3.2: Info tab of a NetLogo model.

5. **Interface elements**. NetLogo allows the user to modify the interface tab by adding elements which can run commands, change parameters used during the simulation or show the value of variables. In addition, these elements can be edited, moved, resized or deleted whenever. The different kinds of elements are:

   - *Button* executes its instructions. There are two types: *once* or *forever*. Once buttons execute their instructions only one time, while forever buttons execute their code repeatedly until they are clicked again.
     Typically a model has at least a *setup* button, to set up the initial conditions of the world, and a *go* button to make the model runs continuously. Some models have additional buttons that perform other actions.
   - *Sliders* set the value of global variables, which are accessible by all agents.
   - *Switches* are a visual representation for a true/false global variable.
   - *Choosers* let users pick a value for a global variable from a list of options. The choices may be strings, numbers, booleans, or lists.
   - *Input* boxes are places where users input a value of a global variable. Possible values only depends on the type of the input box (strings or numbers, for example).
   - *Monitors* display the value of a reporter.
   - *Plots* show data generated by the model.
   - *Output*. This is a scrolling area of text which can be used to create a log of activity in the model.
   - *Notes* allows modelers to add informative text in the Interface tab.


Stock_Market_Model_by_Pol_Ferrando - NetLogo {C:\Users\Ferrando\Desktop\NetLogo}
File Edit Tools Zoom Tabs Help
Interface Info Code
Find... Check
Procedures
Indent automatically

```
patches-own[
  pbuy          ; buy price
  psell         ; sell price
  xi            ; expected return
  Pb_B          ; probability to buy
  Pb_S          ; probability to sell
  lambda        ; prospect theory parameter
  beta          ; permeability parameter
  omega         ; overconfidence parameter
  lim-stp       ; stop-loss rule parameter
  counter       ; turns without operating (auxiliar variable)
  DEI?          ; disposition-effect investors
  STP?          ; stop-loss investors
  CFD?          ; contract for difference investors
  buyer?        ; auxiliary variable
  seller?       ; auxiliary variable
  transactions ; total number of transactions
]

to setup
  if random? = false [random-seed 12345]                      ; fixed seed of experiment if random? is false
  clear-all                                                   ; clean previous simulations
  let aux (0.0001 + random-float (r_max * 100)) / 100
  ask patches[
    let t random-float 1                                      ; auxiliary variable distributed between zero a
    ifelse t < STP [ set STP? true set DEI? false] [          ; handing agents among STP e DEI
      set STP? false set DEI? true ]                          ; handing agents among STP e DEI
    set CFD? false
    set buyer? false                                          ; cleaning variable
    set seller? false                                         ; cleaning variable
    set pcolor (white)                                        ; white for agents not operated in the period
    set pbuy 0                                                  ; purchase price equal to zero - initial set
    set psell 0                                               ; selling price of zero - initial setup
    ifelse homogeneity? = true [ set xi aux ] [ set xi (0.0001 + random-float (r_max * 100)) / 100]    ;
    set lambda 2.25 * xi                                      ; recording individual value
    set transactions 0                                        ; resetting number of transactions the agent
    set lim-stp (1 + random-float 100) / 100
    set beta random-float beta_max
    set counter 0
     ]
  set I 0                                                     ; setup indice
  set p_t P_initial                                           ; setup initial price
  set p_t-1 P_initial                                         ; setup initial price
  reset-ticks
```

Figure 3.3: Example of a code tab of a NetLogo model.

3.4 NetLogo programming language

3.4.1 Agents

In NetLogo, there are specific commands for each agent. However, other commands can be run by several agent types. All information about pre-built commands, such as their syntax or which agents are able to run them, can be found in the NetLogo Dictionary[2]. If it is not specified which agents have to run each command, the code will be run by the observer if it is permitted.

Examples:

i) `forward` (or `fd`) is a turtle-related command. Its syntax is "`forward` *number*". With this command, the turtle moves forward by *number* steps (if number is negative, the turtle moves backward).

ii) `distance` command can be run only by a turtle or a patch. Its syntax is "`distance` *agent*" and it reports the distance from the *agent* to the given turtle or patch.

3.4.2 *Ask*

The observer is not represented in the world but it can give a command to the patches or turtles using `ask`.

Its syntax is: `ask` *agent/agentset* [*commands*], and it makes that the specified *agent* or *agentset* runs the given commands.

[2]https://ccl.northwestern.edu/netlogo/docs/dictionary.html

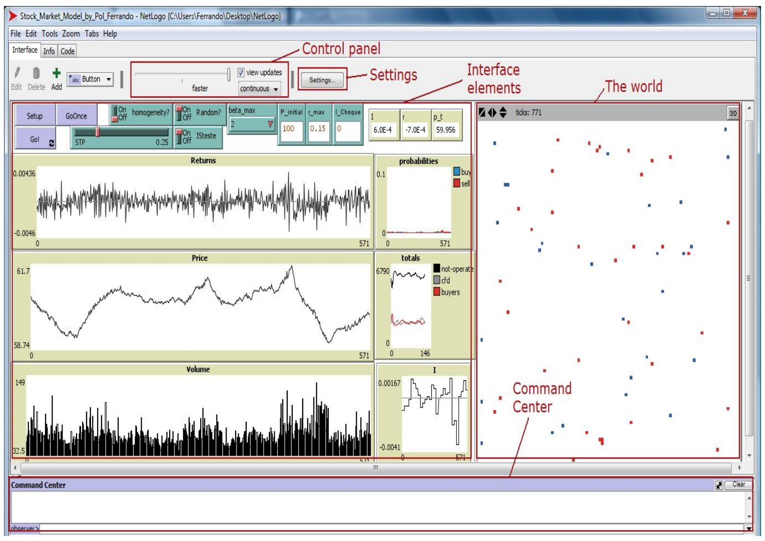


Figure 3.4: Example of interface tab of NetLogo model.

Therefore, the observer uses `ask` to ask all turtles, all patches or all links to run commands. You can also use `ask` to have an individual agent do the same. However, when a command is directly given to a group of agents, you only have to give the command itself.

Examples:

i) `ask turtles [ face patch 0 0 ]`. All turtles will be oriented towards patch (0,0).

ii) `ask patch 0 0 [ set pcolor green ]`. Only the patch with coordinates (0,0) will turn green.

3.4.3 Procedures

In NetLogo, **commands** and **reporters** tell agents what to do.

Commands are actions that have to be performed by an agent or agentset, resulting in some effect. Some examples are `forward` or `face`, used before.

Reporters are instructions for computing a value, which the agent then "reports" to whoever asked it. The instruction `distance` used before in this section is an example. It may be used within a command as a condition.

On one hand, we have commands and reporters pre-built in NetLogo; they are called *primitives*. As we said before, the NetLogo Dictionary has a complete list of primitive commands and reporters. Among them, there are arithmetic operators like `+`, `-`, `*`, `/`, `>`, `>=`... and other common mathematical functions like `sin`, `exp`, `log`, `mean` or `variance`. It is also possible generate random numbers with primitive reporters like `random` (which reports a random integer number), `random-float` (which reports a random floating point number) or `random-normal` (which reports a random number of a normal distribution with mean and standard-deviation specified). Finally, important primitive commands are the ones that control flow like `if`, `ifelse` or `stop`, and all the logic commands like `and` or `or`.

For instance, we are going to analyze in depth the following instruction:

```
ask patches[
    ifelse distance (patch 0 0) > 5 [set pcolor red][set pcolor yellow]
  ]
```

In this case, we have used *three* primitive commands (ask, ifelse, set), *one* primitive reporters (distance) and *one* primitive variable[3](pcolor):

1. `ask`. See Section 3.4.2.
2. `ifelse`. Syntax: `ifelse` *reporter* `[` *commands1* `]` `[` *commands2* `]`. Description: *reporter* must report a true/false value. The agent runs *commands1* or *commands2* depending on whether it reports true or false, respectively. The *reporter* may report a different value for different agents, so some agents may run *commands1* while others run *commands2.*
3. `distance`. Syntax: `distance` *agent.* Description: as we explained in 3.4.1, it reports the distance from the *agent* to the given turtle or patch.
4. `set`. Syntax: `set` *variable value.* Description: sets *variable* to the given *value.*
5. `pcolor`. Description: variable containing the color of a patch.

Therefore, that code would turn the color of patches into red or yellow, depending on if they were farther or closer than 5 units of distance from, respectively.

On the other hand, commands and reporters defined by the modeler are called **procedures**. Their code begins with `to` or `to-report`, depending on whether it is a command procedure or a reporter procedure, followed by a name which will be its label in the program. To put an end to the procedure, `end` is used.

For example, the code for a procedure called *move* that makes turtles walk randomly around the world would look like this:

```
to move
  ask turtles[
    rt random 360
    fd 1
  ]
end
```

[3]See 3.4.4.

Many commands and reporters take inputs, which are values that the command or reporter uses in performing its actions or computing its result. Procedures can take inputs too by putting their names in square brackets after the procedure name.

Finally, when buttons were explained in Section 3.3 we said a model typically has at least a *setup* and a *go* button, but in reality these buttons are just a quick way to run two procedures called *setup* and *go*, respectively. This works for all procedures and commands, so buttons of the Interface tab contain instructions (either the name of the procedure or command, or lines of code directly) that will be run when users press them. So in our example we could add a button called *move* which may contain the code "move". It would call the procedure *move* when a user clicked the button.

3.4.4 Variables

Variables (or agent variables) are places to store values (such as numbers) in an agent. There are **global variables**, **turtle variables**, **patch variables**, or **link variables**. In addition, there are some primitive variables like `color` or `pcolor`, but you can also define your own variables.

Global variables have a single value, and every agent can access them at whatever time. New global variables can be created by adding a switch, slider, chooser or input box in the Interface tab, or by using `globals` at the beginning of the Code tab. For example, `globals [price]` would create a global variable called "price".

Turtle, patch, and link variables are different. Each turtle has its own value for every turtle variable, so there are the same number of values for a turtle variable as turtles in the world. The same goes for patches and links, but it is important to know that a turtle can read and set patch variables of the patch it is standing on. New turtle, patch or link variables can be defined by using `turtles-own` , `patches-own` and `links-own`, respectively. For example, `turtles-own [energy]` would create a turtle variable called energy.

The structure "`[`*variable-name* `] of` *agent/agentset*" used in the correct context would refer to values of the variable called *variable-name* for the specified *agent* or *agentset*. For example, `[xcor] of turtles` used in an observer context would show the current x coordinate of all turtles. As exception, values of global variables would be shown by calling their name without brackets in any context, so in the example above `price` would refer to the value of the global variable *price* and it may be used in other procedures like `if price < 100 [...]`.

In addition, there are **local variables** that are defined and used only within a particular procedure. The `let` command is used to create local variables.

Finally, a distinguished variable-related command is `set`. Its syntax is `set` *variable value*. Thereafter, the given *variable* stores *value*, so it is the way to change values of variables. For example, `ask turtles [ set size 3 ]` would make turtles change their size (which is a pre-built turtle variable) to 3 (by default is 1).

3.4.5 Ticks

In many NetLogo models, time pass in discrete steps called **ticks**. They are important in the *setup* and *go* procedures, which should have the following structure:

```
to setup
   clear-all
    ...
   reset-ticks
end
```

We use `clear-all` and `reset-ticks` commands. First one clears the tick counter along with everything else, and the other one resets the tick counter to zero.

```
to go
   ...
  tick
end
```

The `tick` command advances the tick counter by 1.

3.4.6 Plots

The two basic commands for plotting things are `plot` and `plotxy`.

Plot only requires one reporter which will be the y value plotted. The x value will automatically be 0 for the first point, 1 for the second, and so on. It is especially useful for plotting a new point of a variable at every time step. For example: `plot count turtles` would plot the total number of turtles (using the primitive reporter `count`). This function can use reporters made by the modeler.

On the other hand, plotxy requires two reporters that will be the x and y values of the point plotted. It can use reporters made by the modeler too. For example, `plotxy [xcor] of turtle 1 [ycor] of turtle 1` plots the trajectory of turtle 1.

Chapter 4

Stock Market Model

Taking Silva's (2014) *Collective Behavior in the Stock Market* model as a departure point, we have designed and implemented a simple agent-based stock market simulating the evolution of a single stock price. Agents represent investors buying and selling based on their individual decisions.

The general idea for this market model is that investors try to obtain benefits. Therefore, if investors think that price will rise, they will buy shares of the company expecting this growth and when price reaches their expected return goal, they will sell the shares. This strategy is called *long buying*. In the same way, if investors think that price will fall, they will borrow shares for selling them at a higher price and they will buy them later at a lower price to return them. This other strategy is called *short selling*. Nevertheless, investors' beliefs may be wrong and price can fall (rise) while they are expecting its increase (drop). When this happens, the time until an agent realizes his/her losses and close position will depend on each one.

Investors are divided into two groups: investors affected by *disposition effect* and investors who use *stop-loss* orders, depending on when they close positions. But independently to the group they belong, all investors try to profit using *long buying* and *short selling*.

At each time period, agents have either a long position, or a short position or fetch information from their neighbors and the market sentiment to enter the market. Investors with a long position close it depending on the price. If sale at the current price produces a revenue higher than their individual expected return, they sell. Closing a losing position depends on the type of investor. Investors using stop-loss orders sell if the price decrease results in a loss greater than the stop-loss boundary. Investors affected by the disposition effect sell if losses are greater than 2.25 times their expected return. Finally, if the price has not reached these limits, both types of investors hold their position in this time period. Similarly, investors with a short position close it depending on the price. If buying at the current price produces a revenue higher than their individual expected return, they buy. Closing a losing position also depends on the type of investor. Investors using stop-loss orders buy if the price increase results in a loss greater than the stop-loss boundary. Investors affected by the disposition effect buy when losses are greater than 2.25 times their expected return. Finally, if the price has not reached these limits, both types of investors hold their position in this time period. If investors do not buy or sell, they enter the market with a probability buyprob of long buying and sellprob of short selling. Otherwise, agents do not operate.

Another assumption of the model is that all investors use the same strategy for deciding when they should enter the market. But the same strategy does not mean they act in the same way in a given situation; it will depend on their individual beliefs in the information they have and the market sentiment. With this, we try to capture the idea that there are investors who occasionally exhibit overconfidence resulting in risky operations and also the influence of market sentiment in decision making.

By default, heterogeneous expectations are considered, so every agent has a unique expected return. This captures the idea that every individual has a different behavior in risk situations.

4.1 Model hypotheses

Based on Silva's (2014) model, we consider a market populated by 10.000 agents divided into two groups: investors affected by the *disposition effect* and investors who use *stop-loss* orders. Independently to the group they belong, all investors use *long buying* and *short selling* as strategies.

Agents collect information on whether their neighbors are buying (or selling) and will be willing to buy (or sell) depending on this information. In addition, at any moment an investor j may have additional information recommending to buy or sell. To model this situation, we use an auxiliary variable $O^{j\,t}_{buy(or\,sell)}$ for agent j and time step t. For example, if $O^{j\,t}_{buy} = 1$, the agent possesses extra information advising him to buy; if not, $O^{j\,t}_{buy} = 0$, and analogously for selling. We balance the influence of his preferred choice and his surrounding behavior. We use a weighting term $\omega_{j\,t}$ representing the investor's confidence in his own information: if $\omega_{j\,t} = 0$, the agent pays no attention to the information he owns; if $\omega_{j\,t} = 1$, his own information receives the same weight as that of his neighbors; when $\omega_{j\,t}$ grows, the weight given to his own information increases towards *overconfidence*. Finally, we assume that $\omega_{j\,t}$ follows a uniform distribution on the interval $[0, \omega_{max}]$ where ω_{max} is a maximum value previously established.

On the other hand, investors get information on the global sentiment of the market, so we define a market sentiment index I by:

$$I = D_t + I_{shock}$$

where

$$D_t = \frac{b_t - s_t}{n}, \tag{4.1.1}$$

$n = 10000$, b_t and s_t are the total number of buyers and sellers in the time period t, respectively. Also, I_{shock} is a constant related to additional information ($I_{shock} > 0$ is related to good news about the company, while $I_{shock} < 0$ to bad news and $I_{shock} = 0$ means that there is no important information).

Nevertheless, each agent uses I differently, and the random parameter $\beta_j \in [0, 1)$ captures this fact. $\beta_j = 0$ means that agent j is not influenced by market sentiment, whereas $\beta_j = 1$ means that agent j keeps in mind this information to make decisions.

In summary, the probability that agent j buys stocks in a given time period t is:

$$P_{j\,t}^{buy} = \frac{b_{j\,t} + \omega_{j\,t}\,O_{j\,t}^{buy}}{8 + \omega_{j\,t}\,O_{j\,t}^{buy}} + \beta_j\,I \ \text{, if } I > 0 \tag{4.1.2}$$

where $b_{j\,t}$ is the number of buyers around agent j (maximum 8) and the last term $(\beta_j\,I)$ is not considered if $I \leq 0$.

The probability that agent j sells stocks in a given time period t is:

$$P_{j\,t}^{sell} = \frac{s_{j\,t} + \omega_{j\,t}\,O_{j\,t}^{sell}}{8 + \omega_{j\,t}\,O_{j\,t}^{sell}} + \beta_j\,|I| \ \text{, if } I < 0 \tag{4.1.3}$$

where $s_{j\,t}$ is the number of sellers around agent j (maximum 8) and the last term $(\beta_j\,|I|)$ is not considered if $I \geq 0$.

By default, heterogeneous expectations are considered, so every agent has a unique expected return. In addition, expected returns follow a uniform distribution on the interval $[0, r_{max}]$, where r_{max} is a maximum value previously established.

Finally, we consider a hyperbolic tangent functional form for the excess demand, resulting in the following stock price in the time period t:

$$p_t = p_{t-1}\,(1 + \tanh D_t) \tag{4.1.4}$$

Note that $D_t \in (-1, 1)$ and it is both a measure of the market sentiment and what rules the *law of supply and demand*: $D_t > 0$ iff $b_t > s_t$, that is there are more buyers than sellers and price increases due to price formula 4.1.4 (as the law of supply and demand states); $D_t < 0$ iff $s_t > b_t$, that is there are more sellers than buyers and price decreases (as the law of supply and demand states too).

4.2 Investors affected by the disposition effect

On their analyses of decision under uncertainty, Kahneman and Tversky (1979) realized that people behavior is different in the domains of gains and losses: *risk aversion* prevail when there are gains, whereas *risk seeking* is exhibited in the domain of losses. That is, people tend to choose prospects with less risk (even if their expected value is lower) when their choices involve profits, however they prefer choices with a substantial probability of a loss to a sure loss which is smaller. Furthermore, Kahneman and Tversky also realized that losses produce more dissatisfaction than joy is produced by an equivalent amount of gains . All these ideas result in the Prospect theory.

The disposition effect is an irrational behavior observed in financial decisions that consists in the investors' tendency to hold losing positions for too long and sell winning positions too soon.

It can be explained by prospect theory. About selling winning positions early, prospect theory would say that investors prefer riskless profits to a riskier option that either produces greater or lower profits. Whereas holding losing positions too long could be explained by investors' aversion to losses, which make them choose risky options that could produce greater losses instead of admit their current losses.

To model this behavior, agents affected by disposition effect only will close a loser position if there is a variation in price greater than 2.25 times their individual expected return. Since agents of the model close winning positions when outcome exceeds expected return independently of price tendency, these agents hold losing positions too long, sell winning ones too soon, and furthermore, show a greater loss aversion; so they exhibit the disposition effect. The 2.25 value is the estimation (based on experiments) made by Tversky and Kahneman (1992) to a parameter of the value function used in prospect theory to explain the ideas of previous paragraphs.

4.3 Investors using stop-loss orders

In the stock market, stop-loss orders are one effective way to avoid the disposition effect. Stop-loss orders protect positions by triggering a market order if the price exceeds a certain level. There are two types: sell-stop orders, which protect long positions, and buy-stop orders, which protect short positions. These orders are based on the assumption that if the price falls (or increases) at a certain value, it may continue to fall (or to increase) much further, so the loss is capped by closing the position.

In our model, we suppose that agents who use stop-loss orders always use the same strategy, so each one has a unique fixed value as limit. This limit is the percentage of the investment this agent will accept as losses. If the price at a given moment means losses greater than this limit, a order to close position will be requested. For example, an investor who has 5% as limit and has a open long position whose initial price was 100$ will sell if the price falls below 95$.

4.4 Market entry strategy

If an agent has no stocks, it will enter the market when a chance appears. We suppose investors keep in mind three information sources for deciding when they should invest: individual beliefs, neighbors' actions and the global market sentiment.

Couzin (2008) modeled how information is transferred within animal groups when group members do not know which individuals, if any, have information. In that paper, he also considered the preferences of individuals who has more information than others, a common situation in the stock market.

We adapted Couzin's idea to the stock market in order to compute the probabilities to make a purchase or a sale, adding a term quantifying the global market opinion. The results are Eq. 4.1.2 and Eq. 4.1.3.

4.5 Weaknesses of the model

Making a model of the stock market is not easy due to the quantity of factors that are involved. Even so, we created a model based on the hypotheses discussed before in this chapter. However, we might assume things that are not always true or we might simplify some elements too much. Now we are going to discuss these aspects that could be improved and these real life situations that have been excluded.

First of all, there are intrinsic limitations in the model like the approximation of expressing the price as a formula or the hyperbolic tangent form of the excess demand function.

Each investor may have his own market entry strategy, but we supposed only one strategy for all. Besides that, we use the idea of collective behavior in this strategy but there is no evidence that it works in financial markets.

Although we introduced irrational behaviors like holding loser positions too long due to the disposition effect, these are restricted to the moment when agents realize gains or losses. Thus, spontaneous irrational behaviors are not considered. Because of this, there are common situations in the real stock market that are not considered in our model; for example, buying more assets, even if the investor is losing money and price is still falling, because he supposes that price will recover part of its past value.

In addition, we assume that all agents use both long buying and short selling equally and this is not always true. Of course there are investors using these two strategies equally, but many investors only buy long in the real stock market.

Finally, the value of model's parameters could be discussed too but it is a less important matter because they affect model dynamics less than other hypothesis. For example, it may be reasonable that expected return would not be constant for each agent but different for each investment of this agent. This would capture the idea that investors has different expected returns depending on several factors like the operation risk or the entry price.

Chapter 5

Analysis of financial time series

When a variable is measured sequentially in time at a fixed interval, the data form a *time series*. There are time series in many areas, for example: history of average global temperature in meteorology, birth rates in demography or the daily closing stock prices in economics. Time series analysis is the group of statistical methods used to understand the processes behind time series, make a model of them and, furthermore, forecast future values based on current data.

The main features of many time series are *trends* and *seasonal effects*. If these patterns are deterministic, they can be modeled with functions of time by *linear models*. However, sometimes the trends of a time series cannot be explained by a plausible reason or their changes seem unpredictable; for example, in economic and financial time series. Both random (*stochastic*) and deterministic trends can be removed by *differencing*.[1]

Another important feature of time series is that observations near in time tend to be correlated (or *serially dependent*). A key question in time series analysis is how to include this dependency in the models to fit better the data. There are several methods to explain and estimate this correlation.

Finally, time series can show periods of different variances. This happens when the variance is serially dependent, technically called *conditional heteroskedasticity*, and it is common in financial and climatological time series. A logarithmic transformation may remove this effect, otherwise there are models that try to incorporate it.

In this chapter, we are going to focus on financial time series analysis. First, a common strategy for modeling time series is expounded, followed by the main features of financial time series. Then, basic concepts of time series analysis are defined and the most popular stationary models are discussed. Finally, we explain the concept of volatility in economics and, after that, we describe the ARCH models

[1]Given a time series $\{x_t : t = 1, ..., n \text{ and } n \in \mathbb{N}\}$, note that the differences $\nabla x_t = x_t - x_{t-1}$, for $t \geq 2$, form a new time series.

typically used to fit financial time series.

5.1 Model-building strategy

Box and Jenkins (1976) used a methodology in order to build ARIMA models[2] that has become the basis of time series analysis. This strategy has three main steps:

1. Identification
2. Fitting
3. Diagnostics

Identification (or model specification) consists in selecting a model that may fit an observed time series properly. This step includes plotting the series, calculating several statistics, determining if the observations are correlated and other procedures in order to identify the main features of the data. In addition, when we are selecting the model, we should keep in mind the *principle of parsimony*, which states that the model with the fewest number of parameters should be selected. Nevertheless, the selected model is provisional and subject to revision later.

Since models have parameters that must be estimated from the data, the next step for building a model is calculating the best possible estimate of the parameters included in the selected model. That is the purpose of the fitting step, and it can be reached with either least squares or maximum likelihood estimation.

Finally, model diagnostics evaluates the quality of the model we have specified and estimated. If the model is appropriate, the observed values can be explained by it and, furthermore, these will satisfy the model assumptions. When this happens, we have a model which fits the data properly, so it can be used to forecast future values. Otherwise, we must return to the identification step and repeat the process until we find a model suitable for the observations.

Nowadays, there are statistical software aggregates like R including procedures needed for identification and model fitting, so these programs allow analysts to focus on model diagnostics.

In addition, it is possible that more than one model fits the data. In this case, we should choose one as implemented in the Akaike Information Criterion (AIC; Akaike, 1974). The AIC is a measure of the quality of a statistical model for a given set of data. It rewards a good fitting for the observations but it penalizes a large number of parameters, so it incorporates the principle of parsimony.

[2]These models will be discussed later in this chapter.

Definition 5.1.1. Akaike information criterion (AIC)

$$\text{AIC} = -2 \times \text{log-likelihood} + 2 \times \text{number of parameters}$$

The first term refers to the maximum probability to obtain the data given the model, whereas the second term is the penalization for models with too many parameters. Hence, given a group of possible models for the data, the preferred one is the one with the minimum AIC.

Note that AIC only provides a rule for deciding between a group of options. However, if all candidates fit the data poorly, AIC does not warn us about this fact, so we will choose one model as the better fit erroneously. That is why we must evaluate if the model candidates are a good fit for the data before using AIC.

5.2 Financial time series and their features

Financial time series analysis includes all the procedures used to study the evolution of asset valuations over time. Although it is a field of study very empirical due to the complexity of the subject, there are statistical tools useful for analyzing these series.

5.2.1 Returns

Definition 5.2.1. Simple return
Let P_t the price of an asset at time index t. The *simple (net) return* R_t is defined by

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

That is, the relative price variation.

Therefore, the *simple gross return* is: $1 + R_t = \dfrac{P_t}{P_{t-1}}$.

Most financial studies use returns instead of prices. This mainly happens because returns have no dimensions, so they are very useful to measure investment opportunities or performance. In addition, returns present better statistical properties than prices. However, there are several definitions of an asset return.

Holding the assets for k periods between $t-k$ and t gives a simple gross return

$$1 + R_t[k] = \frac{P_t}{P_{t-k}} = \frac{P_t}{P_{t-k}} = \frac{P_t}{P_{t-1}} \times \frac{P_{t-1}}{P_{t-2}} \times \cdots \times \frac{P_{t-k+1}}{P_{t-k}} =$$

$$= (1 + R_t)(1 + R_{t-1}) \cdots (1 + R_{t-k+1}) = \prod_{j=0}^{k-1}(1 + R_{t-j}) \tag{5.2.1}$$

Definition 5.2.2. Log return
Let P_t the price of an asset at time index t. The *log return (or continuously compounded return)* r_t is defined by

$$r_t = \ln(1 + R_t) = \ln\left(\frac{P_t}{P_{t-1}}\right) = p_t - p_{t-1}$$

where $p_t = \ln P_t$. Note that the log return is the natural logarithm of the simple gross return.

In this case, the k-period gross log return is

$$\begin{aligned} r_t[k] = \ln(1 + R_t[k]) &= \ln[(1 + R_t)(1 + R_{t-1}) \cdots (1 + R_{t-k+1})] = \\ &= \ln(1 + R_t) + \ln(1 + R_{t-1}) + \cdots + \ln(1 + R_{t-k+1}) = \\ &= r_t + r_{t-1} + \cdots + r_{t-k+1} \end{aligned} \tag{5.2.2}$$

Hence, the multi period log return is simply the sum of log returns involved.

5.2.2 Distributions of returns

Stock returns are usually treated as continuous random variables. Therefore, if we have n log returns $\{r_1, ..., r_n\}$, the joint density function is:

$$f(r_1, ..., r_n; \theta) = f(r_1)f(r_2|r_1) \cdots f(r_n|r_{t-1}, ..., r_1) \tag{5.2.3}$$

where θ is a vector of parameters and it is omitted for simplicity in the right part of the identity. To obtain eq. 5.2.3 we used successively the relation between joint, marginal and conditional distributions

$$f_{x,y}(x, y; \theta) = f_{x|y}(x; \theta) \times f_y(y; \theta)$$

Equation 5.2.3 implies that finding the conditional distributions of returns is essential in financial time series. However, marginal distributions are easier to be estimated than conditional ones and, in addition, if observed returns have small correlations, both marginal and conditional distributions tend to be equal. That is why we are going to discuss the two most popular distributions proposed as marginal distributions of returns.

1. **Normal distribution**
   A traditional hypothesis is that the simple returns $\{R_t : t = 1, ..., n\}$ are independent and identically distributed (iid) as normal variables with constant mean and variance. That is, $R_t \sim N(\mu, \sigma^2)$, $\forall t$. However, this assumption implies several problems:

   i) Normal distributions take values in the real line, whereas simple returns are always greater than -1 as a consequence of definition 5.2.1.

ii) The multiperiod simple gross return (Eq. 5.2.1) does not follow a normal distribution because the product of normal distributions is not normally distributed.

iii) Observed returns series do not seem to be normal; they tend to have heavier tails than normal distribution (see Fig. 5.1).

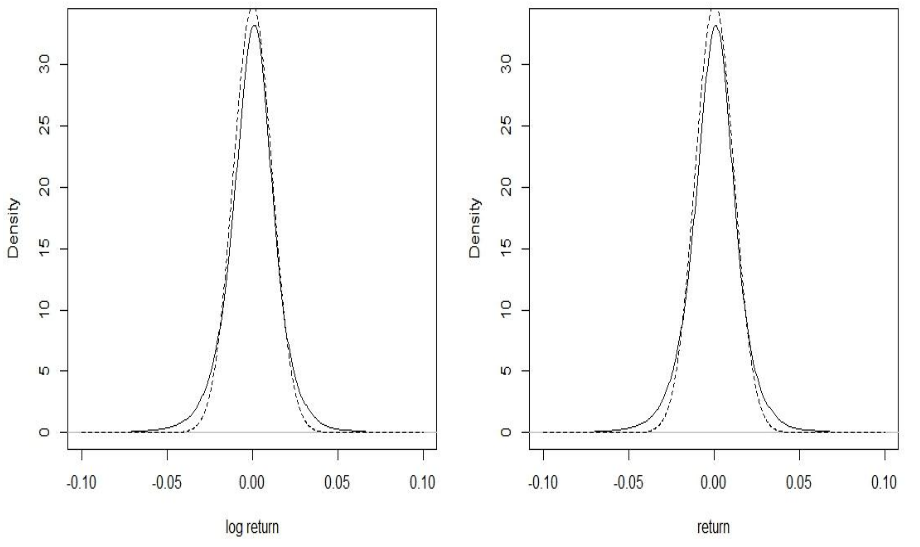


Figure 5.1: *Comparison of empirical (solid lines) and normal (dashed line) densities for the daily simple and log returns of Ibex-35 Index. The sample period is from October 19, 1990 to January 13, 2015. The left plot is for simple returns and the right plot for log returns. The normal density uses the sample mean while the standard deviation is estimated by a sample quantile. We do not use the sample standard deviation due to the high volatility of log returns (See Fig. 5.2).*

2. **Lognormal distribution**

Another common assumption is that the log returns $\{r_t : t = 1, ..., n\}$ are iid and they follow a normal distribution with constant mean and variance. That is, $r_t \sim N(\mu, \sigma^2)$, $\forall t$. Therefore, the simple returns R_t are iid random variables log normally distributed. First, note that

$$1 + R_t = \frac{p_t}{p_{t-1}} = \exp(r_t) \Longrightarrow R_t = \exp(r_t) - 1 > -1.$$

Hence, the previous problem with the lower bound of R_t is solved with the lognormality hypothesis. Furthermore, since the multiperiod log return is just a sum of log returns (Eq. 5.2.2), which are iid and normally distributed, the multiperiod log return also follows a normal distribution. So the second problem of the normality assumption is solved too. However, log-normality is not consistent with observed log returns, again because they tend to have heavier tails than the normal distribution (see Fig. 5.1).

5.2.3 Volatility

As we said in the introduction to this chapter, many financial time series show periods of low and high *volatility.* Later on, in section 5.6 we will show a procedure to model this variability (as *conditional heteroskedasticity*). Volatility is an important factor because it quantifies the investment risk. Also, modeling the volatility of a time series can improve the efficiency in parameter estimation and the accuracy in interval forecast. However, volatility is not directly observable because we usually only have one observation for each time period.

We are going to discuss other relevant features of volatility with one example. Figure 5.2 shows the daily log returns of Ibex-35 Index from October 19, 1990 to January 13, 2015. First, we can see there was a high variance period around 2009, while years 2004 and 2005 had a low variance, so there are volatility clusters. Second, volatility seems to evolve continuously in time, with no sudden changes. Third, variances are within a range of values. For example, in Fig. 5.2 limits are approximately -0.1 and 0.1. Finally, although in this example is not clear, volatility is also characterized by the *leverage effect*, which is the tendency of volatility to increase when the price falls, but not with upwards movements.

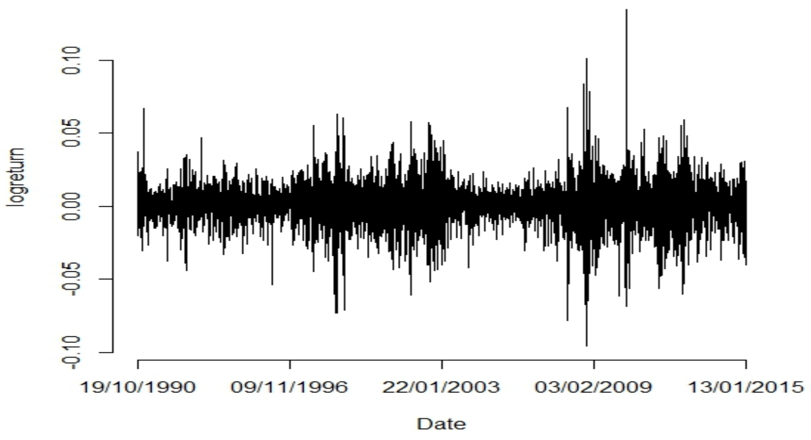


Figure 5.2: *Time plot of daily log returns of Ibex-35 Index from October 19, 1990 to January 13, 2015. (Source: www.infomercados.com)*

Definition 5.2.3. Volatility
The *volatility* at time t, $\sigma^2_{t|}$, is the conditional variance of a given return:

$$\sigma^2_{t|} = Var(r_t|F_{t-1})$$

where F_{t-1} is the available information at time $t-1$.

Hence, *conditional heteroskedastic models*, discussed later in this chapter, focus on modeling this conditional variance so that it satisfies the properties explained above.

5.3 Fundamental concepts

5.3.1 Time series model

Definition 5.3.1. Stochastic process
A stochastic process is the sequence of random variables $\{X_t : t \in \mathbb{Z}\}$ used as a model for an observed time series $\{x_t : t = 1, ..., n\}$. [3] It is usually called *time series model.*

5.3.2 Mean and variance

Definition 5.3.2. Mean function
The *mean function* of a time series model $\{X_t\}$ is

$$\mu_t = E(X_t)$$

That is, the mean function is the expected value of the process at time t. In general, μ_t is a function of t. However, if the mean function is constant, we say that the time series model is *stationary in the mean.*

Since we usually only have a realization of the time series model, we can, without assuming a particular trend structure, estimate the mean at each time period by the corresponding observed value; that is, $\mu_t = x_t$. In practice, we first estimate trend and seasonal effects, remove them and, after that, we treat the residual time series (the random component) as a constant in mean series. In this case, the sample estimate of the population mean μ is the sample mean $\bar{x} = \dfrac{1}{n}\displaystyle\sum_{t=1}^{n} x_t$ where n is the number of observations.

Definition 5.3.3. Variance function
The *variance function* of a time series model $\{X_t\}$ that is *stationary in the mean* is

$$\sigma_t^2 = E\left[(X_t - \mu)^2\right]$$

which may depend on time period t.

If we assume the model is *stationary in the variance*, we can estimate the constant population variance σ by the sample variance $Var(x) = \dfrac{1}{n-1}\displaystyle\sum_{t=1}^{n}(x_t - \bar{x})^2$ where n is the number of observations.

[3]We will distinguish the model from the data by using uppercase and lowercase, respectively.

5.3.3 Stationarity

Definition 5.3.4. Strict stationarity
A time series model $\{X_t\}$ is *strictly stationary* if the joint distribution of $X_{t_1}, ..., X_{t_n}$ is the same as the joint distribution $X_{t_1+k}, ..., X_{t_n+k}$, $\forall t_1, ..., t_n, k \in \mathbb{Z}$. That is, the distribution is the same after an arbitrary time shift.

Definition 5.3.5. Second-order stationarity
A time series model $\{X_t\}$ is *second-order stationary* if it is stationary in both the mean and the variance and, furthermore, the autocovariance $Cov(X_t, X_s)$ depends only on the number of time steps k between them (that is, $k = |t-s|$). This number k is known as the *lag*.

Note that strict stationarity implies second-order stationarity.

5.3.4 Autocorrelation and the correlogram

Definition 5.3.6. Autocovariance function
The *autovariance function* (acvf) of a time series model $\{X_t\}$ is defined as

$$\gamma_{t\,s} = Cov(X_t, X_s)$$

where $Cov(X_t, X_s) = E\left[(X_t - \mu_t)(X_s - \mu_s)\right] = E(X_t X_s) - \mu_t \mu_s$. Note that $\gamma_{t\,t} = Var(X_t)$.[4]

It is a measure of the linear association between whatever X_t and X_s. Positive values mean that X_s tends to increase when X_t increases, whereas negative ones mean that X_s tends to decrease when X_t increases. Finally, $\gamma_{t\,s} = 0$ indicates no linear association between X_t and X_s.

Definition 5.3.7. Autocorrelation function
The *autocorrelation function* (acf) of a time series model $\{X_t\}$ is given by

$$\rho_{t\,s} = Corr(X_t, X_s) = \frac{\gamma_{t\,s}}{\sqrt{\gamma_{t\,t}\gamma_{s\,s}}}$$

It is a dimensionless measure of the linear association between X_t and X_s. Its interpretation is equal to autocovariance's one, with the exception of autocorrelation only takes values between -1 and 1.

For second-order stationary time series models, which have an autocovariance $Cov(X_t, X_s)$ that depends only on *lag* k and a mean constant in time, the acvf is given by

$$\gamma_k = E\left[(X_t - \mu)(X_{t+k} - \mu)\right]$$

[4] $Var(X_t) = E\left[(X_t - \mu_t)^2\right] = E\left[(X_t - \mu_t)(X_t - \mu_t)\right] = \gamma_{t\,t}$.

and does not depend on the time period t. This follows from definition 5.5.9 by replacing s with $t+k$ and using $\mu_t = \mu_{t+k} = \mu$, $\forall k$.

In addition, the acf is defined by

$$\rho_k = \frac{\gamma_k}{\sigma^2}$$

where $\sigma^2 = Var(X_t) = \gamma_{t\,t} = \gamma_0$, $\forall t$. Finally, note that $\rho_0 = 1$.

The acvf and acf can be estimated by their sample equivalents. The sample acvf is $c_k = \dfrac{1}{n}\displaystyle\sum_{t=1}^{n-k}(x_t - \bar{x})(x_{t+k} - \bar{x})$ and the sample acf is $r_k = \dfrac{c_k}{c_0}$. Note that c_0 is the variance calculated with denominator n. And c_k is also used calculated with denominator n, although we are summing $n-k$ terms, because then $c_k \in [-1, 1]$, $\forall k \in \mathbb{N}$.

Therefore, both the acvf and the acf are useful in order to evaluate the dependency between observations near in time. That is why we use the correlogram to do this.

Definition 5.3.8. The correlogram
The *correlogram* is the plot of r_k against the lag k.

If $\rho_k = 0$, r_k is approximately distributed like a normal with mean $-1/n$ and variance $1/n$. Thus, the limits

$$-\frac{1}{n} \pm \frac{1.96}{\sqrt{n}}$$

are usually plotted in the correlogram to identify statistically significant values. If r_k falls outside this confidence interval, we will have evidence against the null hypothesis of ρ_k=0 at the 5% level. If $\rho_k = 0$ $\forall k$, we expect that 5% of the r_k fall outside the limits.

5.4 White noise

A very important example of a stationary model is the *white noise*.

Definition 5.4.1. White noise
A sequence $\{W_t : t = 1, 2, ..., n\}$ of random variables is a *white noise* if the variables W_1, W_2,...,W_n are independent and identically distributed with mean zero.

Note that by this definition all the variables W_t have the same variance, which we will denote σ_w^2.

W_t is strictly stationary:

$$\begin{aligned}P(W_{t_1} \leq x_1, W_{t_2} \leq x_2, ..., W_{t_n} \leq x_n) = &\quad \text{(by independence)}\\ = P(W_{t_1} \leq x_1)P(W_{t_2} \leq x_2)....P(W_{t_n} \leq x_n) = &\quad \text{(identical distributions)}\\ = P(W_{t_1+k} \leq x_1)P(W_{t_2+k} \leq x_2)....P(W_{t_n+k} \leq x_n) = &\quad \text{(by independence)}\\ = P(W_{t_1+k} \leq x_1, W_{t_2+k} \leq x_2, ..., W_{t_n+k} \leq x_n) &\quad \square\end{aligned}$$

Therefore, the autocovariance function is

$$\gamma_k = \begin{cases} \sigma_w^2 & \text{for} \quad k = 0 \\ 0 & \text{for} \quad k \neq 0 \end{cases}$$

because

$$\gamma_k = Cov(W_t, W_{t+k}) = E[(W_t - \mu_t)(W_{t+k} - \mu_{t+k})] = E[(W_t)(W_{t+k})] = Var(W_t)\,\delta_{k\,0} = \sigma_w^2\,\delta_{k\,0}$$

And the autocorrelation functin is given by

$$\rho_k = \begin{cases} 1 & \text{for} \quad k = 0 \\ 0 & \text{for} \quad k \neq 0 \end{cases}$$

One interesting example of white noise is the *Gaussian* white noise, which is the special case when the variables follow a normal distribution (i.e., $W_t \sim \mathrm{N}(0,\, \sigma_w^2)$).

5.5 ARMA models

Removing trends and seasonal effects is the first step of time series analysis. However, sometimes there are correlations between adjacent observations, which can be detected using the correlogram described before in this chapter. In this section we are going to discuss the most popular family of stationary models: the ARMA models.

5.5.1 Moving Average models

Definition 5.5.1. MA(q) process
A*moving average* (MA) process of order q is a linear combination of the current white noise term and the q most recent past white noise terms, so it is given by

$$X_t = W_t + \beta_1 W_{t-1} + \beta_2 W_{t-2} + ... + \beta_q W_{t-q} \tag{5.5.1}$$

where $\{W_t\}$ is white noise with zero mean and variance σ_w^2 and $\beta_q \neq 0$.

If we define the *backward shift operator* (or lag operator) $\mathbf{B}$ by

$$\mathbf{B}X_t = X_{t-1}$$

we can rewrite equation 5.5.1 as

$$X_t = (1 + \beta_1\,\mathbf{B} + \beta_2\,\mathbf{B}^2 + ... + \beta_q\,\mathbf{B}^q)\,W_t = \phi_q(\mathbf{B})\,W_t \tag{5.5.2}$$

where $\phi_q(\mathbf{B})$ is a polynomial of order q.

Properties

1. **MA processes are stationary**

   Since MA processes consist of a sum of white noise terms, and white noise is stationary, they are stationary. Therefore, MA models have constant mean and variance and the autocovariance depends only on the lag.

2. **Mean**

$$\begin{aligned} &\mu = E[X_t] = && \text{(by properties of} \\ &= E[W_t + \beta_1\,W_{t-1} + \beta_2\,W_{t-2} + ... + \beta_q\,W_{t-q}] = && \text{expected value)} \\ &= E[W_t] + \beta_1\,E[W_{t-1}] + \beta_2\,E[W_{t-2}] + ... + \beta_q\,E[W_{t-q}] = && \text{(white noise)} \\ &= 0 \end{aligned}$$

3. **Variance**

$$\begin{aligned} &\sigma^2 = Var[X_t] = && \text{(by properties of} \\ &= Var[W_t + \beta_1\,W_{t-1} + \beta_2\,W_{t-2} + ... + \beta_q\,W_{t-q}] = && \text{the variance)} \\ &= Var[W_t] + \beta_1^2\,Var[W_{t-1}] + ... + \beta_q^2\,Var[W_{t-q}] + \sum_{i\neq j}\beta_i\,\beta_j\,Cov(W_i, W_j) = \\ &= \sigma_w^2\,(1 + \beta_1^2 + ... + \beta_q^2) \end{aligned} \tag{5.5.3}$$

   since $\{W_t\}$ is white noise.

4. **Acvf and acf**

   The autocovariance function is given by

$$\begin{aligned} &\gamma_k = Cov(X_t, X_{t+k}) = \\ &= Cov\left(\sum_{i=0}^{q}\beta_i\,W_{t-i}, \sum_{j=0}^{q}\beta_j\,W_{t+k-j}\right) = && \text{(properties of covariance)} \end{aligned}$$

$$= \sum_{i=0}^{q}\sum_{j=0}^{q} \beta_i\,\beta_j Cov\left(W_{t-i}, W_{t+k-j}\right) = \sum_{i=0}^{q}\sum_{j=0}^{q} \beta_i\,\beta_j \hat{\gamma}_{k+i-j} = \qquad \text{(white noise)}$$

$$= \begin{cases} \sigma_w^2\left(1+\beta_1^2+...+\beta_q^2\right) & \text{for} \quad k=0 \\ \sigma_w^2 \sum_{i=0}^{q} \beta_i\,\beta_{i+k} & \text{for} \quad 1 \le k \le q \\ 0 & \text{for} \quad k>q \end{cases} \tag{5.5.4}$$

where $\hat{\gamma}$ refers to the acvf of the white noise sequence $\{W_t\}$ and $\beta_0 = 1$.

Therefore, the autocorrelation function is given by

$$\rho_k = \begin{cases} 1 & \text{for} \quad k=0 \\ \sum_{i=0}^{q-k} \beta_i\,\beta_{i+k} / \sum_{i=0}^{q} \beta_i^2 & \text{for} \quad 1 \le k \le q \\ 0 & \text{for} \quad k>q \end{cases} \tag{5.5.5}$$

5. **Invertibility**

   An MA(q) process is invertible if it can be expressed as a stationary autoregressive process of infinite order without an error term, which we are going to define later. It can be demonstrated that this is equivalent to the fact that all the roots of $\phi_q(\mathbf{B})$ exceed unity in absolute value.

5.5.2 Auto Regressive models

Definition 5.5.2. AR(p) process
An *autoregressive* (AR) process of order p is given by

$$X_t = \alpha_1\,X_{t-1} + \alpha_2\,X_{t-2} + ... + \alpha_p\,X_{t-p} + W_t \tag{5.5.6}$$

where $\{W_t\}$ is white noise with zero mean and variance σ_w^2 and $\alpha_p \neq 0$.

Using the backward shift operator, it also can be expressed as

$$\theta_p(\mathbf{B})\,X_t = \left(1 - \alpha_1\,\mathbf{B} - \alpha_2\,\mathbf{B}^2 - ... - \alpha_p\,\mathbf{B}^p\right) X_t = W_t \tag{5.5.7}$$

where $\theta_p(\mathbf{B})$ is a polynomial of order p.

The term ‘autoregressive’ of the name is due to the model is a regression of X_t on the p most recent past terms from the same series.

Properties

1. **Stationarity**

   An AR(p) process is stationary if, and only if, all the roots of $\theta_p(\mathbf{B})$ exceed unity in absolute value. In this case, the AR process can be expressed as a moving average process of infinite order:

$$(1 - \alpha_1 \mathbf{B} - \alpha_2 \mathbf{B}^2 - ... - \alpha_p \mathbf{B}^p) X_t = W_t \Rightarrow$$

$$\Rightarrow X_t = (1 - \alpha_1 \mathbf{B} - \alpha_2 \mathbf{B}^2 - ... - \alpha_p \mathbf{B}^p)^{-1} W_t = (1 + \beta_1 \mathbf{B} + \beta_2 \mathbf{B}^2 + ...) W_t$$

   where the parameters β_i are functions of α_1, α_2, etc. and their powers, so roots' absolute value must be greater than 1 for convergence.

2. **Mean**

   Assuming stationarity:

$$\begin{aligned}
\mu &= E[X_t] = && \text{(stationarity)}\\
&= E\left[\sum_{i=0}^{\infty} \beta_i W_{t-i}\right] = && \text{(properties of the expected value)}\\
&= \sum_{i=0}^{\infty} \beta_i E[W_{t-i}] = && \text{(white noise)}\\
&= 0
\end{aligned}$$

3. **Variance**

   Assuming stationarity:

$$\begin{aligned}
\sigma^2 &= Var[X_t] = && \text{(stationarity)}\\
&= Var\left[\sum_{i=0}^{\infty} \beta_i W_{t-i}\right] = && \text{(properties of the variance)}\\
&= \sum_{i=0}^{\infty} \beta_i^2 Var[W_{t-i}] + \sum_{i \neq j} \beta_i \beta_j Cov(W_i, W_j) = && \text{(white noise)}\\
&= \sigma_w^2 \sum_{i=0}^{\infty} \beta_i^2 && (5.5.8)
\end{aligned}$$

   where $\beta_0 = 1$.

4. **Acvf and acf**

   The autocovariance function is given by

$$\gamma_k = Cov(X_t, X_{t+k}) =$$

$$= Cov\left(\sum_{i=0}^{\infty} \beta_i W_{t-i}, \sum_{j=0}^{\infty} \beta_j W_{t+k-j}\right) = \qquad \text{(properties of covariance)}$$

$$= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \beta_i \beta_j Cov\left(W_{t-i}, W_{t+k-j}\right) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \beta_i \beta_j \hat{\gamma}_{k+i-j} = \qquad \text{(white noise)}$$

$$= \begin{cases} \sigma_w^2 \sum_{i=0}^{\infty} \beta_i^2 & \text{for} \quad k = 0 \\ \sigma_w^2 \sum_{i=0}^{\infty} \beta_i \beta_{i+k} & \text{for} \quad k \geq 1 \end{cases} \tag{5.5.9}$$

where $\hat{\gamma}$ refers to the acvf of the white noise sequence $\{W_t\}$.

Therefore, the autocorrelation function is given by

$$\rho_k = \begin{cases} 1 & \text{for} \quad k = 0 \\ \sum_{i=0}^{\infty} \beta_i \beta_{i+k} / \sum_{i=0}^{\infty} \beta_i^2 & \text{for} \quad k \geq 1 \end{cases} \tag{5.5.10}$$

For instance, we are going to discuss the main features of AR(1) models.

$$X_t = \alpha X_{t-1} + W_t \Longrightarrow (1 - \alpha \mathbf{B}) X_t = W_t \Longrightarrow X_t = (1 - \alpha \mathbf{B})^{-1} W_t$$

We have:

$$(1 - \alpha \mathbf{B})^{-1} = 1 + \alpha \mathbf{B} + \alpha^2 \mathbf{B}^2 + ... = \sum_{i=0}^{\infty} \alpha^i \mathbf{B}^i$$

Hence

$$X_t = \left(\sum_{i=0}^{\infty} \alpha^i \mathbf{B}^i\right) W_t = \sum_{i=0}^{\infty} \alpha^i W_{t-i}$$

Its mean is zero and its variance is (see Eq. 5.5.8)

$$\sigma^2 = \sigma_w^2 \sum_{i=0}^{\infty} (\alpha^i)^2 = \sigma_w^2 \sum_{i=0}^{\infty} \alpha^{2i} = \frac{\sigma_w^2}{1 - \alpha^2}$$

where the series is convergent if, and only if, $|\alpha| < 1$. Since $\theta_p(\mathbf{B}) = 1 - \alpha \mathbf{B}$, which root is $1/\alpha$, the condition $|\alpha| < 1$ is equivalent to $|1/\alpha| > 1$ as we said as stationarity condition.

The autocovariance function is given by (see Eq. 5.5.9)

$$\gamma_k = \left\{ \begin{array}{lll} \dfrac{\sigma_w^2}{1-\alpha^2} & \text{for} & k=0 \\ & & \\ \dfrac{\sigma_w^2\,\alpha^k}{1-\alpha^2} & \text{for} & k \geq 1 \end{array} \right\} = \frac{\sigma_w^2\,\alpha^k}{1-\alpha^2} \quad \forall\, k$$

because for $k \geq 1$: $\gamma_k = \sigma_w^2 \sum_{i=0}^{\infty} \alpha^i\,\alpha^{i+k} = \sigma_w^2\,\alpha^k \sum_{i=0}^{\infty} \alpha^{2i}$

Therefore, the autocorrelation function is

$$\rho_k = \frac{\gamma_k}{\sigma^2} = \frac{\sigma_w^2\,\alpha^k/(1-\alpha^2)}{\sigma_w^2/(1-\alpha^2)} = \alpha^k$$

Therefore, since $|\alpha| < 1$, the acf decays exponentially to zero.

5.5.3 ARMA models

The autoregressive moving average (ARMA) models that we are going to describe in this section are a compact form that combines AR and MA models so that the number of parameters used is smaller.

Definition 5.5.3. ARMA(p,q) process
An *autoregressive moving average* (ARMA) process of order (p, q) is given by

$$X_t = \alpha_1\,X_{t-1}+\alpha_2\,X_{t-2}+...+\alpha_p\,X_{t-p}+W_t+\beta_1\,W_{t-1}+\beta_2\,W_{t-2}+...+\beta_q\,W_{t-q} \quad (5.5.11)$$

where $\{W_t\}$ is white noise with zero mean and variance σ_w^2, $\alpha_p \neq 0$ and $\beta_q \neq 0$.

Using the backward shift operator, it also can be expressed as

$$\theta_p(\mathbf{B})\,X_t = \phi_q(\mathbf{B})\,W_t \quad (5.5.12)$$

where

$$\theta_p(\mathbf{B})\,X_t = 1 - \alpha_1\,\mathbf{B} - \alpha_2\,\mathbf{B}^2 - ... - \alpha_p\,\mathbf{B}^p$$

and

$$\phi_q(\mathbf{B}) = 1 + \beta_1\,\mathbf{B} + \beta_2\,\mathbf{B}^2 + ... + \beta_q\,\mathbf{B}^q$$

Properties

1. **Stationarity**

   An ARMA(p,q) process is stationary if, and only if, all the roots of the AR characteristic equation $\theta_p(\mathbf{B}) = 0$ exceed unity in absolute value. In this case, the ARMA process can be expressed as a moving average process of infinite order:

$$(1 - \alpha_1 \mathbf{B} - \alpha_2 \mathbf{B}^2 - ... - \alpha_p \mathbf{B}^p)\, X_t = (1 + \beta_1 \mathbf{B} + \beta_2 \mathbf{B}^2 + ... + \beta_q \mathbf{B}^q)\, W_t \Rightarrow$$

$$\Rightarrow X_t = (1 - \alpha_1 \mathbf{B} - \alpha_2 \mathbf{B}^2 - ... - \alpha_p \mathbf{B}^p)^{-1} (1 + \beta_1 \mathbf{B} + \beta_2 \mathbf{B}^2 + ... + \beta_q \mathbf{B}^q)\, W_t =$$
$$= (1 + \hat{\beta}_1 \mathbf{B} + \hat{\beta}_2 \mathbf{B}^2 + ...)\,(1 + \beta_1 \mathbf{B} + \beta_2 \mathbf{B}^2 + ... + \beta_q \mathbf{B}^q)\, W_t =$$
$$= [1 + (\beta_1 + \hat{\beta}_1)\, \mathbf{B} + (\beta_2 + \hat{\beta}_2 + \hat{\beta}_1 \beta_1)\, \mathbf{B}^2 + ...]\, W_t =$$

   where the parameters $\hat{\beta}_i$ are functions of α_1, α_2, etc. and their powers, so roots' absolute value must be greater than 1 for convergence.

   Note that the other properties are the same as for AR models.

5.6 ARCH models

In order to explain volatility, we need a model that allows for conditional changes in the variance. One option is to use an autoregressive model for the variance process, which results in the autoregressive conditional heteroskedastic (ARCH) models.

The correlogram of the squared series (after removing linear dependences if it is necessary) is used to detect ARCH effects because it will show statistically significant values of the acf if there is volatility. We also can use tests like the Box-Ljung test for autocorrelation.

Definition 5.6.1. ARCH(p) process
An *ARCH process* of order p is given by

$$X_t = W_t \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i\, X_{t-i}^2} \qquad (5.6.1)$$

where $\{W_t\}$ is white noise with zero mean and unit variance. Also, $\alpha_0 > 0$ and $\alpha_i \geq 0$, for $i > 0$.

In a compact form:

$$X_t = \sigma_t \, W_t, \qquad \sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \, X_{t-i}^2$$

Note that, with this structure, large past squared shocks imply a large conditional variance for the new observation X_t. Consequently, it is more likely that X_t will have a large value (but not necessarily, because a large variance only increases the probability of obtaining a large value). This behavior is similar to the volatility clusters observed in financial time series.

Properties

1. **Mean**

$\mu = E[X_t] =$ (conditional expectation)

$$= E[E[X_t | X_{t-j},\ j=1,2,...]] = E\left[E\left[W_t \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \, X_{t-i}^2} \, | \, X_{t-j},\ j=1,2,...\right]\right] =$$

$$= E\left[E[W_t] \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \, X_{t-i}^2}\right] = \qquad \text{(white noise)}$$

$= 0$

2. **Variance**

$\sigma_t^2 = Var[X_t] = E[X_t^2]$ (conditional expectation)

$$= E[E[X_t^2 | X_{t-j},\ j=1,2,...]] = E\left[E\left[W_t^2 \left(\alpha_0 + \sum_{i=1}^{p} \alpha_i \, X_{t-i}^2\right) | \, X_{t-j},\ j=1,2,...\right]\right] =$$

$$= E\left[E[W_t^2] \left(\alpha_0 + \sum_{i=1}^{p} \alpha_i \, X_{t-i}^2\right)\right] = \qquad \text{(properties of the expected value)}$$

$$= \sigma_w^2 \left(\alpha_0 + \sum_{i=1}^{p} \alpha_i \, E\left[X_{t-i}^2\right]\right) = \alpha_0 + \sum_{i=1}^{p} \alpha_i \, \sigma_{t-i}^2$$

since $\sigma_w^2 = 1$.

If we compare this equation with Eq. 5.5.2, we see that the variance of an ARCH(p) process behaves like an AR(p) model.

3. **Positive excess kurtosis**.

   The excess kurtosis κ is defined by

   $$\kappa = \frac{E[X_t^4]}{Var[X_t]^2} - 3$$

   and it is a measure of the heaviness of tails. A normal distribution has $\kappa = 0$, whereas heavy-tailed distributions have $\kappa > 0$ and light-tailed distributions have $\kappa < 0$.

   It can be demonstrated that ARCH models have positive excess kurtosis ($\kappa > 0$), so the distribution of X_t has heavier tails than normal distributions, which is consistent with observed financial series.

4. **Weaknesses of ARCH models**

   ARCH models assume that positive and negative observations of time series have the same effects on volatility because it depends on the square of the previous values, when in reality the reaction of asset prices is different in each case. Furthermore, ARCH processes tend to overpredict the volatility because they respond slowly to isolated large realizations.

   Volatility models like ARCH, which are commonly used in financial time series analysis, must be applied to time series with no linear dependencies, so first we must fit an AR, MA or ARMA model if there are correlations in the data to remove these dependencies and then we must fit an ARCH to the residuals of the ARMA.

Chapter 6

Simulated stock market series

We used the stock market model described in chapter 4 in order to obtain a simulated time series of log returns. In the current chapter, we are going to apply all the methodologies, the procedures and the tools learned previously in this work to the observed data and discuss if they have the main features of real financial time series.

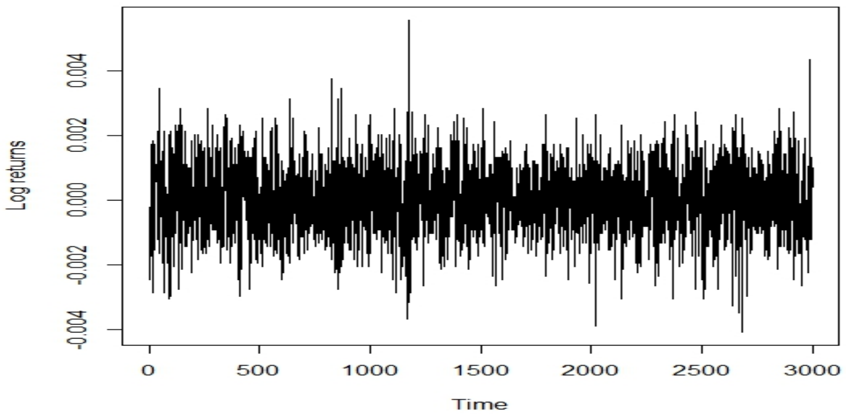


Figure 6.1: *Time plot of 3000 simulated log returns of an asset. The simulation was made with the stock market model described in chapter 4.*

Fig. 6.1 shows a representative run of the simulation process, with n=10000 agents, expected return limit $r_{max} = 0.15$, $\omega_{max} = 1$ and $I_{shock} = 0$. Also, the ratio of investors affected by the disposition effect to investors using stop-loss orders is 3. Conventionally, the initial price is set to 100, although it does not affect the model dynamics. At first sight, the series seems to be stationary, so we will analyze its correlogram to determine if an ARMA model could fit the data. However, first we are going to analyze the marginal distribution of log returns and check if the null hypothesis of normality is rejected or not using the *Shapiro-Wilk test.*

Looking at Fig. 6.2 we may think that the simulated log returns follow a normal distribution because the normal density fits properly to the empirical one. However, if we look carefully the Q-Q plot (right), we note that both tails are heavier than the normal ones. To dissipate any doubts, we use the Shapiro-Wilk test, whose null hypothesis is that the data are normally distributed.

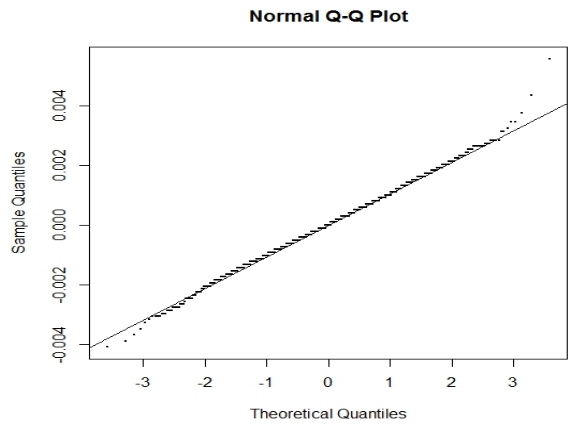


Figure 6.2: *Q-Q plot of simulated log returns versus a normal distribution.*

```
        Shapiro-Wilk normality test

data:  r
W = 0.9974, p-value = 5.469e-05
```

Since the p-value is 5.469×10^{-5}, we reject the null hypothesis of normality at the level 0.05. Hence, despite the appearance, the simulated log returns are not normal.

In Fig. 6.1 the series seems to be stationary, so the next step in our analysis is trying to fit an ARMA model to the data. But first we should verify that the observations are correlated. According to the correlogram (Fig. 6.3), there are correlations statistically significant. We also verify it with the Box-Ljung test for autocorrelation, whose null hypothesis is that observations are not correlated.

```
        Box-Ljung test

data:  r.ts
X-squared = 219.5913, df = 9, p-value < 2.2e-16
```

Since the p-value is less than 2.2×10^{-16}, we reject the null hypothesis of uncorrelated data at the level 0.05. Therefore, an ARMA model seems to be a good fit.

Based on the Akaike Information Criterion (AIC), see Table 6.1, we obtain that the ARMA model which fits better the data is an ARMA(1,2). Using the `arima` function in R, we obtain:

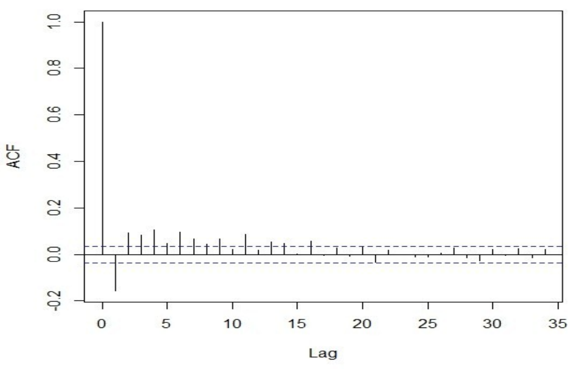


Figure 6.3: *Correlogram of the simulated log returns series.*

```
Call:
arima(x = r.ts, order = c(1, 0, 2))

Coefficients:
         ar1      ma1     ma2  intercept
      0.9152  -1.1171  0.2707          0
s.e.  0.0159   0.0228  0.0179          0

sigma^2 estimated as 1.036e-06:  log likelihood = 16413.03,  aic = -32816.05
```

Therefore, the fitted model is

$$r_t = 0.92\, r_{t-1} + z_t - 1.12\, z_{t-1} + 0.27\, z_{t-2} \tag{6.0.1}$$

where $\{z_t\}$ is assumed to be *white noise* with zero mean and variance $\sigma_z^2 = 1.044 \times 10^{-6}$.

	ARMA(1,0)	ARMA(0,1)	ARMA(1,1)	ARMA(2,0)	ARMA(0,2)
AIC	-32655.47	-32643.47	-32660.09	-32667.34	-32679.73
Log Likelihood	16330.74	16324.73	16334.05	16337.67	16343.86

	ARMA(2,1)	ARMA(1,2)	ARMA(2,2)	ARMA(3,0)	ARMA(0,3)
AIC	-32794.82	-32816.05	-32814.99	-32700.98	-32707.69
Log Likelihood	16402.41	16413.03	16413.50	16355.49	16358.85

Table 6.1: AIC and Log Likelihood of fitted ARMA processes.

The model-building strategy described in section 5.1 states that the next step must be the diagnostics of the fitted model, so we are going to check if the assumptions of the model are verified by the data. Therefore, the residual series of the fitted model should be white noise, which is the same as both the residual series and its square should be uncorrelated. However, as we see in Fig. 6.4, the ARMA fit solved the problem with the correlated observations but cannot explain the correlation in the variance. The Box-Ljung test confirms our suppositions:

```
        Box-Ljung test

data:  res
X-squared = 4.9993, df = 9, p-value = 0.8344

        Box-Ljung test

data:  res^2
X-squared = 181.4821, df = 9, p-value < 2.2e-16
```

The p-value of the test for the residuals is 0.8344 and, thus, we can accept the null hypothesis of no autocorrelations at the level 0.05, while the p-value for the squared residuals is less than 2.2×10^{-6}, so we reject the null hypothesis at the level 0.05. Hence, we are in front of a *conditionally heteroskedastic* variance, so we may need to fit an ARCH model.

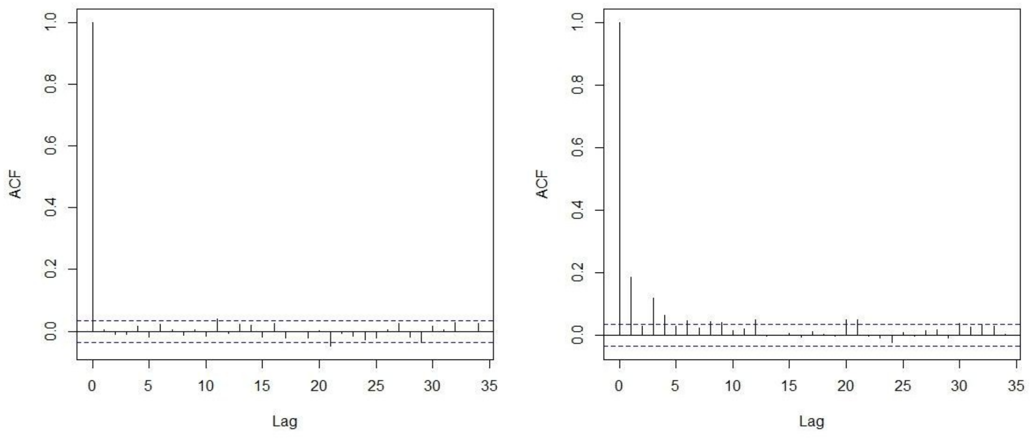


Figure 6.4: *Correlogram of the residuals of the fitted ARMA(1,2) for the simulated log returns (on the left). Correlogram of the squared residuals of the fitted ARMA(1,2) for the simulated log returns(on the right).*

To verify that an ARCH model could be a good fit, we do the Lagrange Multiplier test for autoregressive conditional heteroskedasticity (ARCH LM-test), whose null hypothesis is that there are no ARCH effects in the time series.

```
        ARCH LM-test; Null hypothesis: no ARCH effects

data:  res
Chi-squared = 155.5402, df = 12, p-value < 2.2e-16
```

Since the p-value is less than 2.2×10^{-6}, we reject the null hypothesis and an ARCH model is required to fit the residuals.

Based on the AIC, we obtain an ARCH(3) (see Table 6.2). Using the `garch` function in R:

```
Model:
GARCH(0,3)

Residuals:
      Min        1Q    Median        3Q       Max
-3.447758 -0.660397 -0.006432  0.665198  3.563245

Coefficient(s):
    Estimate  Std. Error  t value Pr(>|t|)
a0 7.375e-07   3.556e-08   20.741  < 2e-16 ***
a1 1.929e-01   2.534e-02    7.613 2.69e-14 ***
a2 4.589e-13   1.839e-02    0.000 1.000000
a3 8.982e-02   2.391e-02    3.757 0.000172 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Therefore, the fitted model is

$$z_t = a_t \sqrt{7.38 \times 10^{-7} + 0.19\, z_{t-1}^2 + 0.09\, z_{t-3}^2} \tag{6.0.2}$$

where $\{a_t\}$ is assumed to be *white noise* with 0 mean and unit variance. Note that coefficient `a2` is not considered because it is not statistically significant.

	ARCH(1)	ARCH(2)	ARCH(3)	ARCH(4)
AIC	-32910.17	-32897.33	-32914.47	-32896.64
Log Likelihood	19212.98	19206.64	19215.29	19206.46

	ARCH(5)	ARCH(6)	ARCH(7)	ARCH(8)
AIC	-32885.96	-32867.46	-32857.13	-32852.36
Log Likelihood	19201.2	19192.03	19186.95	19184.64

Table 6.2: AIC and Log Likelihood of fitted ARCH processes.

As we can see in Fig. 6.5, the ARCH effects seem to be eliminated from the residuals of the ARCH model, and we are going to verify it by the Ljung-Box tests:

```
        Box-Ljung test
```

```
data:  res.garch
X-squared = 4.7878, df = 9, p-value = 0.8524

        Box-Ljung test

data:  res.garch^2
X-squared = 9.259, df = 9, p-value = 0.4137
```

Since the p-values are 0.8524 and 0.4137 for the new residual series and its square, we can accept the null hypothesis of no autocorrelations. Furthermore, if we do the ARCH LM-test for the residuals of the ARCH:

```
        ARCH LM-test; Null hypothesis: no ARCH effects

data:  res.garch
Chi-squared = 21.3472, df = 12, p-value = 0.04552
```

Although we obtain a p-value less than 0.05, it is very close to that level and we will accept the hypothesis of no ARCH effects because both correlograms and Box-Ljung tests suggested it. Therefore, the residuals of the ARCH can be considered white noise and we finally have found a model which fit the simulated log returns.

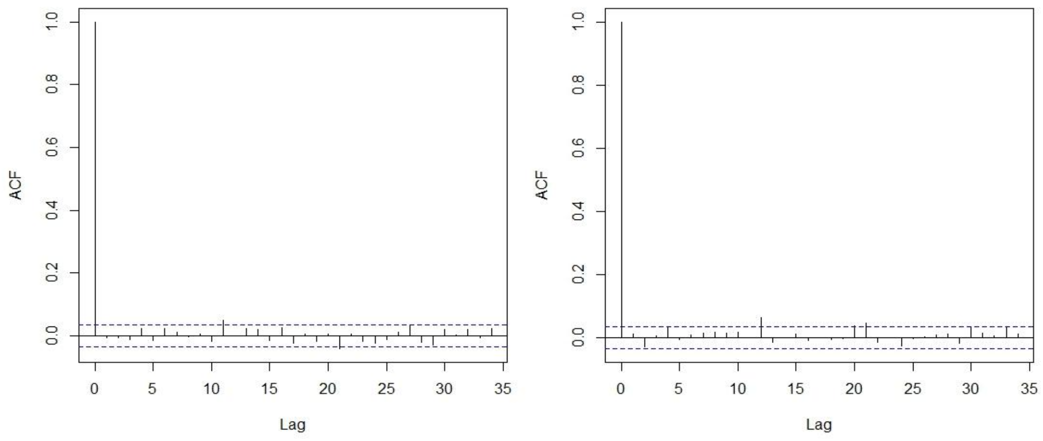


Figure 6.5: *Correlogram of both the residuals (on the left) and the squared residuals (on the right) of the ARCH(3) model fit for the residuals of the ARMA(1,2).*

In summary, the complete model for the log returns is one ARMA(1,2)-ARCH(3) given by:

$$r_t = 0.92\, r_{t-1} + z_t - 1.12\, z_{t-1} + 0.27\, z_{t-2} \tag{6.0.3}$$

where

$$z_t = a_t \sqrt{7.38 \times 10^{-7} + 0.19\, z_{t-1}^2 + 0.09\, z_{t-3}^2}$$

and $\{a_t\}$ is *white noise* with 0 mean and unit variance.

Chapter 7

Conclusions

Throughout this work, we have learned that agent-based simulations are useful to make models of complex systems such as the stock market, where the collective behavior of investors, each of which acts independently, produces price movements. Therefore, ACE models can be used for better understanding of such systems. In that sense, NetLogo is an agent-based programming language whose simplicity have allowed us to redesign Silva's model in order to reproduce a very simple stock market.

In addition, time series analysis attempts to understand the processes behind observations, extract insights from data and make forecasts. It considers time series as a sequence of random variables, thus powerful tools of the probability theory can be used. The main purpose of time series analysis is to model trends, seasonal effects, dependence between adjacent observations and volatile variance. If trends and seasonal effects are deterministic, they may be solved using linear models or logarithmic transformations. Also, correlation between adjacent observations is removed by autoregressive (AR) and moving average (MA) models. However, the compact form of ARMA processes improve the fitted model due to less parameters are required. On the other hand, ARCH models are used to remove the volatility effects. The latter is based on the same idea as autoregressive models but applied to the squared series; that is, variance at each time period is determined by the most recent past variances. To identify when an ARMA or an ARCH model is required, we use the correlogram. Finally, AIC is the standard criterion used to choose between several fitted models so that the model which has the minimum AIC is the best fit.

Returns (or log returns) are frequently used instead of prices due to they provide the same information as well as they have no dimensions and present better statistical properties. However, returns usually do not follow neither a normal nor a log-normal distribution, so they are considered time series and, furthermore, usually verify the stationarity assumption. Therefore, returns usually can be fit to an ARMA or/and an ARCH model.

The simple agent-based stock market described in chapter 4 has a remarkable realistic feature: typically the simulated returns series can be fit to an ARMA-ARCH model. Therefore, the output series of returns exhibit several properties of real financial time series: their distribution is heavy-tailed and there are both correlation and volatility.

Nevertheless, this work is just an initial step in a study of the relation between agent properties, as described by their functionality and parameters, and the statistical behavior of the resulting series of returns.

Appendix: NetLogo code

```
globals[
  r         ; realized return
  D         ; demand
  I         ; indice
  p_t-1     ; price t - 1
  p_t       ; price
  buyer     ; total buyer
  seller    ; total seller
  hold      ; total hold
]

patches-own[
  pbuy          ; purchase price
  psell         ; sale price
  xi            ; expected return
  Pb_B          ; probability to buy
  Pb_S          ; probability to sell
  lambda        ; prospect theory parameter
  beta          ; permeability parameter
  omega         ; overconfidence parameter
  lim-stp       ; stop-loss rule parameter
  counter       ; turns without operating (auxiliar variable)
  DEI?          ; disposition-effect investors
  STP?          ; stop-loss investors
  shortselling? ; short selling strategy
  buyer?        ; auxiliary variable
  seller?       ; auxiliary variable
  transactions  ; total number of transactions
]

to setup
  if random? = false [random-seed 12345]                    ; fixed seed of experiment if random? is false
  clear-all                                                 ; clean previous simulations
  let aux (0.0001 + random-float (r_max * 100)) / 100
  ask patches[
    let t random-float 1                                    ; auxiliary variable distributed between zero and one
    ifelse t < STP [ set STP? true set DEI? false] [        ; handing agents among STP e DEI
      set STP? false set DEI? true ]                        ; handing agents among STP e DEI
    set shortselling? false
    set buyer? false                                        ; cleaning variable
    set seller? false                                       ; cleaning variable
    set pcolor (white)                                      ; white for agents not operated in the period
    set pbuy 0                                              ; purchase price equal to zero - initial setup
    set psell 0                                             ; selling price of zero - initial setup
    ifelse homogeneity? = true [ set xi aux ]
       [ set xi 0.01 + (random-float (r_max * 100) / 100)]  ; create heterogeneous expectations
    set lambda 2.25 * xi                                    ; recording individual value
    set transactions 0                                      ; resetting number of transactions the agent
    set lim-stp 0.01 + random-float xi
    set beta random-float 1
    set counter 0
     ]
  set I 0                                                   ; setup indice
  set p_t P_initial                                         ; setup initial price
  set p_t-1 P_initial                                       ; setup initial price
  reset-ticks
end

to go                                              ; Run
  ask patches[
    if ticks > 150 [if counter > 75[set pcolor white set pbuy 0 set psell p_t set shortselling? false]]
    ifelse STP? = true[ agent-STP ] [ agent-DEI ]
  ]
  set buyer count patches with [buyer?]
  set seller count patches with [seller?]
  set hold (max-pycor * max-pxcor) - buyer - seller
  ifelse (buyer + seller) != 0 [ set D ((buyer - seller) / (buyer + seller + hold)) ] [ set D 0 ]
  set p_t (((exp(D) - exp(- D)) / (exp(D) + exp(- D)) + 1 ) * p_t-1)
  set r (ln(p_t) - ln(p_t-1))
  set p_t-1 p_t
  I-indice
  tick

end

to fetch-information
  let info? random 2
  let aux random 2
  set Pb_B 0
  set Pb_S 0
  set omega 0.001 + (random-float omega_max)
  ifelse info? = 1 [
    ifelse aux = 0[set Pb_B ((count neighbors with [pcolor = blue]) + omega) / (8 + omega)]
                  [set Pb_S ((count neighbors with [pcolor = red]) + omega) / (8 + omega)]
  ][
    set Pb_B (count neighbors with [pcolor = blue]) / 8
    set Pb_S (count neighbors with [pcolor = red]) / 8
  ]
  ifelse I > 0 [ set Pb_B Pb_B + beta * I][ set Pb_S Pb_S - beta * I]
end

to agent-DEI              ; investors affected by the disposition effect
  let t random-float 1
  ifelse ticks < 100 [ ifelse t < (1 / 3) [ buy ] [ ifelse t > ( 2 / 3) [ sell ] [ not-operate ] ] ] [
    ifelse pbuy = 0 [
      ifelse shortselling? = false[
        fetch-information
        ifelse t < (Pb_B) [ buy ][ ifelse t > (1 - Pb_S) [ set shortselling? true sell] [ not-operate ] ]
      ][
        ifelse p_t > psell [ closing-operation-high-shortselling ] [ closing-operation-falling-shortselling ]
      ]
    ][
      ifelse p_t > pbuy[ closing-operation-high ] [ closing-operation-falling ]
    ]
  ]
end

to agent-STP              ; investors using stop-loss orders
  let t random-float 1
  ifelse ticks < 100 [ ifelse t < (1 / 3) [ buy ] [ ifelse t > ( 2 / 3) [ sell ] [ not-operate ] ] ] [
  ifelse pbuy = 0 [
    ifelse shortselling? = false[
      fetch-information
      ifelse t < (Pb_B) [ STP-buy ][ ifelse t > (1 - Pb_S) [ set shortselling? true STP-sell] [ not-operate ] ]
    ][
        ifelse p_t > psell [
          ifelse (p_t - psell) > (lim-stp * psell) [ STP-buy-shortselling ] [ not-operate ]
        ] [
          ifelse (psell - p_t) > (xi * psell) [ STP-buy-shortselling ] [ not-operate ]
        ]
    ]
  ] [
      ifelse p_t > pbuy [
        ifelse (p_t - pbuy) > (xi * pbuy) [ STP-sell ] [ not-operate ]
      ] [
        ifelse (pbuy - p_t) > (lim-stp * pbuy) [ STP-sell ] [ not-operate ] ]
    ]
  ]
end

to buy
  set buyer? true
  set seller? false
  set pcolor (blue)
  set pbuy p_t
  set psell 0
  set counter 0
```

```
end

to sell
  set buyer? false
  set seller? true
  set pcolor (red)
  set pbuy 0
  set psell p_t
  set counter 0
end

to not-operate
  set buyer? false
  set seller? false
  set pcolor (white)
  set counter counter + 1
end

to STP-buy
  set buyer? true
  set seller? false
  set pcolor (blue)
  set pbuy p_t
  set psell 0
  set counter 0
end

to STP-buy-shortselling
  set shortselling? false
  set buyer? true
  set seller? false
  set pcolor (blue)
  set pbuy 0
  set psell p_t
  set counter 0
end

to STP-sell
  set transactions transactions + 1
  set buyer? false
  set seller? true
  set pcolor (red)
  set pbuy 0
  set psell p_t
  set counter 0
end

to closing-operation-high
  ifelse (p_t - pbuy) > (xi * pbuy) [
      set transactions transactions + 1
      set buyer? false
      set seller? true
      set pcolor (red)
      set pbuy 0
      set psell p_t
      set counter 0
      ] [ not-operate ]
end

to closing-operation-high-shortselling
  ifelse (p_t - psell) > (lambda * psell) [
      set transactions transactions + 1
      set shortselling? false
      set buyer? true
      set seller? false
      set pcolor (blue)
      set pbuy 0
      set psell p_t
      set counter 0
      ] [ not-operate ]
```

```
end

to closing-operation-falling
  ifelse (pbuy - p_t) > (lambda * pbuy) [
      set transactions transactions + 1
      set buyer? false
      set seller? true
      set pcolor (red)
      set pbuy 0
      set psell p_t
      set counter 0
      ] [ not-operate ]
end

to closing-operation-falling-shortselling
  ifelse (psell - p_t) > (xi * psell) [
      set transactions transactions + 1
      set shortselling? false
      set buyer? true
      set seller? false
      set pcolor (blue)
      set pbuy 0
      set psell p_t
      set counter 0
      ] [ not-operate ]
end

to I-indice
  ifelse ISTeste = true [ if ticks > 1 [ set I (D + I_shock)] ] [ set I 0 ]
end
```

Bibliography

[1] Wilensky, U. (1999). NetLogo. `http://ccl.northwestern.edu/netlogo/`. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[2] Poza, David J. *Un manual de NetLogo en español*

`<https://sites.google.com/site/manualnetlogo/home>`

[3] Silva, E. (2014). *Collective Behavior in the Stock Market*, NetLogo Modeling Commons. `http://modelingcommons.org`.

[4] Tesfatsion, L., Judd, K. L. (eds.) (2006), *Handbook of Computational Economics. Vol. 2: Agent-Based Computational Economics.* North-Holland/Elsevier, Amsterdam, the Netherlands.

[5] Cowpertwait, Paul S. P., Metcalfe, Andrew V. (2009). *Introductory Time Series with R.* New York: Springer Science+Bussiness Media, LLC.

[6] Cryer, Jonathan D., Chan, Kung-Sik (2008) *Time Series Analysis: With Applications in R.* New York: Springer Science+Bussiness Media, LLC.

[7] Tsay, Ruey S. (eds.) (2005) *Analysis of financial time series* (2nd edition). Hoboken: John Wiley & Sons, Inc.

[8] Phung, Albert. *Behavioral finance: Key concepts* in *Investopedia.*

`<http://www.investopedia.com/university/behavioral_finance/behavioral11.asp>`

`<http://www.investopedia.com/university/behavioral_finance/behavioral9.asp>`

[9] Couzin, Iain D., Krause, Jens, Franks, Nigel R., Levin, Simon A. (2005). Nature, vol. 433, p.513-516