Evolution of an Artificial Market and its use to Predict Future Stock Prices

Louis Charbonneau

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Science (Computer Science) at
Concordia University
Montreal, Quebec, Canada

October 2008

Your file Votre référence
ISBN: 978-0-494-45455-8
Our file Notre référence
ISBN: 978-0-494-45455-8

ABSTRACT

Evolution of an Artificial Market and
its use to Predict Future Stock Prices

Louis Charbonneau

We propose a model of a deterministic artificial stock market driven by a chromosome that encodes the different trading rules of its agents as individual genes. We first define a stylized version of a price-adjustment mechanism that is calibrated to real market data to interpret any random chromosome. Once the gene is activated, we use a steady-state genetic algorithm to invert the market, namely to infer which chromosome is activated in order to generate a given financial time-series without any a priori knowledge of its agent structure. This reconstructed active chromosome is then used to generate price forecasts. These forecasts are analyzed and compared to the standard ARIMA time-series forecasting method.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Nawwaf Kharma, for superb assistance and encouragement, without which this dissertation could not have been written. Special thanks to Joel Krajden, Manager of core technologies at the Dean of Engineering in Concordia University, for assistance in programming the ENCS cluster.

The model described in this work was written on top of the "GAlib" genetic algorithm C++ package, from Matthew Wall at the MIT. This library is available at http://lancet.mit.edu/.

ARIMA parameter estimation was conducted with the Matlab™ System Identification Toolbox™, version 7.2, available at www.mathworks.com.

Lognormal random walk parameter estimation was conducted with the R language `sde` package written by Stefano M. Iacus. The R language is available at http://www.r-project.org/, and the `sde` package can be downloaded from the Comprehensive R Archive Network (CRAN), linked at http://www.r-project.org/.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Investors in the stock markets know that a good forecast of the future evolution of stock prices would be very useful information to know in order to decide on which investment strategy to follow. However, the canonical model of stock prices in finance is the *lognormal random walk,* which offers no appreciable help in forecasting. The lognormal random walk for a stock typically includes two terms: a drift component, which accounts for the central tendency of the price in the future and a volatility term, which is disproportionately high in comparison to drift in actual markets.(Lévy 1965 [47], Black and Scholes 1973 [8], Merton 1973 [48]). Denoting by S_0 the current stock price, μ the annual rate of drift and σ the annual stock volatility stock, the stock price at time T years S_T is given by:

$$S_T = S_0 \cdot e^{((\mu - \frac{\sigma^2}{2})T + \widetilde{\varepsilon}\sigma\sqrt{T})}$$

, where $\widetilde{\varepsilon} \sim N(0,1)$. The expected value of S_T is $S_0 \cdot e^{\mu T}$, which is linear in the logarithm, is not useful knowledge for investors.

In this work, our main goal is to describe how *evolutionary computation* can help us making financial forecasts that improve upon those derived from this canonical stock model, as well as from the standard statistical model used for general system identification, the autoregressive integrated moving average (ARIMA) model.

Evolutionary computation refers to a family of algorithms whose purpose is to imitate (and stylize) the process of biological evolution in developing solutions for a problem. The following pseudocode provides the general algorithm:

```
Generate population of random solutions
While (no satisfactory solution exists)
{
   Measure quality of each solution in population
   Select solutions to reproduce based on quality measure
   Produce offspring solutions from selected parents
   Generate a new generation of solutions by merging parents and offspring
}
```

The genealogy of the idea of reproducing biological evolution in computers is hard to pinpoint: similar techniques have received many names and have had many founders (*cf.* Bäck *et al.* 1997 for a summary [6]). Since the 1950s, computer scientists have tried to solve problems with their machines, and it became clear at that time that designing correct and efficient algorithms is possible in certain cases, and difficult (or impossible) in other cases. So difficult that some wondered whether the computer could be setup to find solutions of problems *by itself*, without human intervention. The appeal of this idea is obvious, but soon it became clear that setting up a problem to make it suitable for evolution is a rather delicate matter that does require human intervention after all.

In this light, evolutionary computation can be defined as one of the two broad approaches to solving a problem with a computer: either the programmer designs the

algorithm that solves it, or the programmer meta-programs an evolving system that is left to itself to ultimately generate a (satisfactory) solution. This seductive idea is fraught with difficulties. For instance, evolution is never guaranteed to converge to an optimal solution. A fascinating biological example is given in Richard Dawkins' "Blind Watchmaker" (Dawkins, 1996): the eyes of mammals (including that of *homo sapiens*), are constructed contrary to common sense, showcasing the worst possible design decisions. Examination of the retina under microscope shows that the photocells are facing away from the light, whereas the cells' wiring is hanging outwards, obstructing the incoming rays. Worse yet, all the wiring needs to return to the brain somehow to carry the signal from the photocells, but the way it does so is by regrouping all the axons together in one bundle. This bundle re-enters the retina at one central area, causing the eye to have a "blind spot". An engineer faced with that problem would have certainly made all the wiring return to the brain either in a uniformly distributed way, or at least at some area far off-center. Ironically enough, lower animals such as squids have eyes constructed the right way, with photocells facing outwards and wiring going inwards. It is very clear that in this case, evolution has generated various random solutions for eyes in parallel and since badly designed eyes did not represent any significant reproductive disadvantage to the animals that carried them, mammals have continued to exist. This is an example of a local optimum that persists in the population forever. It also illustrates an important fact about evolutionary techniques in general, namely that solutions that are reached are strictly consistent with the nature of the quality measure that is used to evaluate each solution: nature uses reproductive ability as a quality measure, not good eyesight. This can have unintended consequences in an artificial setting, as we will show

later.

How is evolutionary computing used to forecast financial markets? Several attempts have been made; the most popular method uses genetic programming (*cf.* for instance Kaboudan 2000 [36]). Genetic programming (GP) evolves trees who are interpreted as functions, whose inputs are past prices and outputs are forecasts (Koza, 1992 [40]). The end result of evolution is a complex-looking function of past prices which has been found successful at making forecasts over a given time period. The generic programming approach could be criticized as being totally *ad hoc*, since there is neither financial logic nor structure behind the function that is found to be the best. Moreover, a GP specification search on the space of best forecasts with no restriction on the functional form is what is known in the econometrics literature as a form of *data snooping*, and should not be adopted in practice unless tested for statistical significance using a data-snooping aware statistical procedure such as the White bootstrap data snooper (White 2000 [58]).

Data snooping refers to the fact that when one tries a large number of functional relationships in data that may have an important random component, then one usually succeeds in eventually "finding it". But the illusory "finding" in this case is due to chance instead of a real relationship present in the data. After all, in standard statistical tests of H_0 against H_a, the probability of an error of type 1 can be fixed for instance at $\alpha = 1\%$, but that probability holds only when one tests one model on one sample. If we try a large number of models on the same sample, then the probability of rejecting H_0 eventually tends to 1, no matter how small α is. To prevent data-snooping bias, the statistical significance of the finding should be evaluated, taking into account all the unsuccessful formulations that were

tried before the one that worked. The *Bonferroni correction* used to adjust the t-values of multiple comparison tests may be familiar to the reader; it is the ancestor of data-snooping tests, since it raises the significance bar if a test is performed on many samples.

To avoid the data-snooping bias, we impose a lot of microstructure on the problem to ensure that the forces of evolution do not generate functions that do not make financial sense. Formally, our approach is an instance of what is known as an "inverse problem" in the geophysics literature (*cf.* Aster *et al.* 2005 [5]). Adopting the formulation in Aster *et al.* 2005, let m be a vector of physical parameters of a model, and d as a vector of observations (that may depend on time), with the function $G(m)$ describing the dynamics of a physical system.

$$d = G(m)$$

In practical situations, this relationship will be contaminated with noise η:

$$d = G(m_{true}) + \eta$$

The *forward problem* consists in finding d knowing G and m_{true}. The *inverse problem* consists in finding m_{true} knowing G and d. The *system identification problem* consists in finding G knowing m_{true} and d (presumably under some restrictions as for the functional form).

A familiar example of an inverse problem is *tomography.* Aster 2005 [5] defines tomography as:

"the general technique of determining a model from path-integrated properties

> such as attenuation (e.g. X-ray, radar, seismic), travel time (e.g. electromagnetic, seismic, acoustic), or source intensity (e.g. positron emission)." (p. 10.)

Another example is the (common) practice of vertical seismic profiling, which consists in inferring the composition of geological strata from seismograph readings of explosions from the bottom of a borehole. It should be pointed out that inverse problems are hard, and oftentimes the vector m is not uniquely determined. To take a simple example, imagine that G is a matrix with more columns than rows. Then trivially the system $Gm = 0$ has an infinite number of solutions, so m_{true} is not uniquely determined. Moreover, inverse problems can be ill-posed in the sense that the solution is sensitive to small changes in η.

In our case, the function $G(m)$ will be defined as an *artificial stock market*, and the model m that will be found will represent the behavioural rules of the traders of that artificial stock market. The data d will be the market orbit, namely the time-series of stock prices generated endogenously by the behaviour of the traders.

Since our target audience is assumed to have a background in computer science, Chapter 2 provides a detailed background on genetic algorithms and on the mechanics of markets, which are crucial to understanding our model. Chapter 3 states the problem, Chapter 4 describes the evolutionary model, Chapter 5 presents the results, and Chapter 6 concludes.

Chapter 2

Background

In this chapter, we review genetic algorithms. We next explain basic market mechanisms that are relevant to our computer model, and conclude by discussing artificial stock market models.

2.1 Genetic algorithms

It is best to present genetic algorithms by means of an example. Consider the following game, a variation on the theme of the Mastermind^{TM1} game. Imagine that there are two players, an encoder and a decoder. The encoder selects a $n \cdot m$ code matrix of symbols taken from the alphabet (called "alleles"), which the decoder tries to guess, using the smallest possible number of runs. Each run proceeds as follows:

- Step 1: the decoder submits a trial solution, which is a $n \cdot m$ code matrix of his choice.

[1]According to Wikipedia, "Mastermind or Master Mind is a simple code-breaking board game for two players, invented in 1970 by Mordecai Meirowitz, an Israeli postmaster and telecommunications expert. It was played as a pencil and paper game called bulls and cows at least as far back as the early 1960s". (http://en.wikipedia.org/wiki/Mastermind_(board_game))

- Step 2: the encoder returns a single number to the decoder, namely how many symbols are in the right place. Let us call this single number the "fitness" function f.

Steps 1 and 2 are repeated until the decoder has found the correct code matrix, namely when the fitness returned is equal to $n \cdot m$. The crucial detail here is that knowledge of fitness provides very limited information, since it is insufficient to determine which symbols are right and where they are located in the code matrix. The decoder must therefore infer the location and the nature of the symbols by submitting trial solutions intelligently, and taking into consideration all the previous trials and their respective fitness. Obviously, this process of guessing is made hardest if the encoder selects a random code to begin with.

While an optimal algorithmic solution could probably be found with some ingenuity, let us instead illustrate how a genetic algorithm would proceed on that problem. We need to modify step 1:

- Step 1': the decoder submits *a population* of trial solutions, which are $n \cdot m$ code matrices of his choice.

Suppose for instance that the encoder selected the following target 3 x 3 code matrix:

$$\begin{bmatrix} f & o & o \\ b & a & r \\ b & a & z \end{bmatrix}$$

Target

And that the decoder submits the four following guesses, generated randomly. We write in bold the symbols that happen to be at the right place:

$$\underset{\text{guess 1}}{\begin{bmatrix} y & \mathbf{o} & e \\ d & h & d \\ g & d & d \end{bmatrix}}, \underset{\text{guess 2}}{\begin{bmatrix} y & u & i \\ y & t & o \\ e & \mathbf{a} & p \end{bmatrix}}, \underset{\text{guess 3}}{\begin{bmatrix} m & x & x \\ n & c & z \\ b & v & q \end{bmatrix}}, \underset{\text{guess 4}}{\begin{bmatrix} k & f & d \\ l & g & s \\ j & h & a \end{bmatrix}}.$$

Then the encoder will compare each of these trial solutions to his original code matrix and return to the decoder their respective fitness. In this case, they are $f_1 = 1$, $f_2 = 1$, $f_3 = 0$, $f_4 = 0$, using obvious notation.

The next decision of the decoder consists in selecting the next generation of guesses. In our example, since each of guesses 1 and 2 contains one symbol at the right place, it may be a good idea to keep them into the next generation. By doing so, we create a so-called "steady-state genetic algorithm". The process of keeping the best individuals unaltered from a generation to the next is called "elitism". Suppose that we decide that the overlap between generations will be 50%, and we retain the solutions with the highest fitness scores. In case of ties, we select the guesses that stay to the next generation randomly. Since we need two more trial guesses to reach a population of 4, then we need two more matrices to submit. There are three ways to proceed:

- choose entirely new random matrices. This is not a standard procedure.
- alter in a random way a matrix from the previous generation. This is called "mutation".
- "crossover" two parents, thereby producing two siblings. This is done by merging the codes of the parents according to a certain pattern. Each sibling will inherit alleles

from the mother and from the father, and can be likened to sexual reproduction in diploid organisms.

Let us illustrate crossover, easier seen on an example where the parents are constituted of all the same symbols. For two-dimensional matrices, many crossover possibilities exist, either stochastic or deterministic. "Uniform crossover" for instance is a stochastic scheme in which each allele in the child is selected independently with a probability from either the mother or the father. To illustrate deterministic crossover, we can select alleles in a regular pattern from mother and father:

$$\underset{\text{mother}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}, \underset{\text{father}}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}} \xrightarrow[\text{crossover}]{} \underset{\text{child 1}}{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}, \underset{\text{child 2}}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}}.$$

We can also do a "single-point crossover". First, a random location is chosen within the matrix. Then we interchange the sub-matrices from mother and father starting from the chosen point up to the last row and column. The following example assumes point (1, 1) was selected:

$$\underset{\text{mother}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}, \underset{\text{father}}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}} \xrightarrow[\text{crossover}]{} \underset{\text{child 1}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}}, \underset{\text{child 2}}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}.$$

We can also interchange selected rows or columns that can also be chosen randomly. The following example assumes row 2 was selected:

$$\underset{\text{mother}}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}}, \underset{\text{father}}{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}} \xrightarrow[\text{crossover}]{} \underset{\text{child 1}}{\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}}, \underset{\text{child 2}}{\begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}}.$$

Other variants are possible. Returning to our game, imagine that we use single-point crossover with point (1, 1) on guesses 1 and 2 to produce two siblings that will be part of the next generation:

$$\underset{\text{guess 1}}{\begin{bmatrix} y & \mathbf{o} & e \\ d & h & d \\ g & d & d \end{bmatrix}}, \underset{\text{guess 2}}{\begin{bmatrix} y & u & i \\ y & t & o \\ e & \mathbf{a} & p \end{bmatrix}} \xrightarrow[\text{crossover}]{} \underset{\text{child 1}}{\begin{bmatrix} y & \mathbf{o} & e \\ d & t & o \\ g & \mathbf{a} & p \end{bmatrix}}, \underset{\text{child 2}}{\begin{bmatrix} y & u & i \\ y & h & d \\ e & d & d \end{bmatrix}}.$$

The first child from this crossover is favoured by nature since it has a fitness of 2, which is higher than either of its parents. Child 2 has a fitness of zero, which will result in its elimination on the next generation.

Usually, both the crossover as well as the mutation operator is applied at each run. To illustrate mutation, imagine that both child 1 and 2 are mutated before passed on to the next generation. One convenient way to implement mutation is to simulate a Bernoulli trial for each position, flipping the symbol to another value (each possible value having equal probability of being selected) if the Bernoulli trial results in a success. In the following example, suppose that positions (1, 2) and (3, 1) are chosen to be mutated in child 1:

$$\underset{\text{original child 1}}{\begin{bmatrix} y & \mathbf{o} & e \\ d & t & o \\ g & \mathbf{a} & p \end{bmatrix}} \underset{\text{mutation}}{\longrightarrow} \underset{\text{mutated child 1}}{\begin{bmatrix} y & k^* & e \\ d & t & o \\ w^* & \mathbf{a} & p \end{bmatrix}}.$$

We see that mutation actually damaged child 1, as it lowered its fitness from 2 to 1. As in living organisms, mutation is a blind force that may either damage or improve a guess. Selecting a probability of mutation represents a trade-off between stability and diversity: if too frequent, mutations may result in the destruction of the solutions already obtained (unless we use a steady-state algorithm as illustrated here, which guarantees the survival of the best individuals into the next generation). On the positive side, mutations cause the algorithm to visit more efficiently the solution space by creating diversity. Many implementations of genetic algorithms start by setting a high probability of mutation in the first generations, which is gradually let to decrease according to a certain schedule.

There are many other mutation schemes that can be implemented. For instance:

- "creep mutation" adds a small random value to each gene with a certain probability. For instance, one may change a "d" to a "e" or a "c", or do nothing at all.
- "swap mutation" swaps symbols from one matrix location to the other.
- "scramble mutation" scrambles symbols within a matrix.
- "inversion mutation" inverts the orders of all symbols or of a specific group of symbols within a matrix.

One can see easily that repeating the process of crossover and mutation for several generations should result eventually in a perfect guess in the Mastermind game. Of course, it is not guaranteed that progress will be made at every generation, but since the algorithm explores a large space of possibilities and retains the best guesses, then it eventually should converge to the full solution.

Selecting parents was easy in our example, since there were two guesses with high fitness and we need two parents, so we took them both. In larger populations, we can select parents in several standard ways:

- *Fitness proportional selection*: we select each parent i with a probability $p_i = \frac{f_i}{\sum_{\text{all j}} f_j}$. This scheme is known to have a few important drawbacks: it may lead to *premature convergence*, which is the situation in which good guesses generated early in the process tend to dominate the population and prevent further variability. This may cause a genetic algorithm to get stuck into a local maximum. The scheme may also paradoxically stall the algorithm in the situation where the fitnesses of most individuals are clustered. If an individual is only slightly better than the rest, the difference in the probability of selecting that individual will not be high enough to favor the transmission of desirable traits into the next round.

- *Rank selection*, which is designed to mitigate the problem of stalling just mentioned. We simply sort individuals by fitness scores, but choose the selection probability based on the rank, not on fitness. In practice, various scalings can be used. Eiben *et al.* [20] discuss various scaling functions.

- *Tournament selection*: the two previous schemes required knowledge of the fitness of

all individuals before parents could be selected. In some situations, fitness may be costly to evaluate. This may happen for example when it is evaluated subjectively by human observers: genetic algorithms producing artworks commonly use this method. Tournament selection consists in selecting (with or without replacement) a subset of k guesses from a population containing N guesses, where $k \ll N$.Fitness is evaluated only for the k guesses, and the best individual is taken from that group. Several variations on this idea are possible, including a probabilistic selection of guesses within a group, which gives a chance of even the worst to survive until the next generation.

Before finishing this quick overview of genetic algorithms, it may be useful to stress the crucial distinction between genotype and phenotype. In living organisms, "genotype" refers to the genetic code. This code, subjected to crossover and mutation, is not a living thing but a series of molecules containing information. The living organism themselves (the phenotypes) are the end result of a long development process. This process is determined in a large measure by the genotype but is affected by various environment conditions, which may trigger or hinder gene expression. The important observation to make is that fitness in nature is evaluated on the basis of the phenotype, not on the genotype. Fitness is therefore necessarily only an *indirect measure* of the quality of the genotype. In the Mastermind problem, the genotypes (namely the code matrices) are as a matter of fact identical to the phenotypes, a feature that renders the problem straightforward to solve. This is not the case for all problems to which genetic algorithms are applied to.

2.2 Real market organizations

In this section, we briefly describe the market algorithms that are relevant to our model, and we explain how prices are generated. A detailed reference on market organization that is helpful for computer modelling is Harris 2002 [30].

2.2.1 Order-driven markets

The simplest kind of market algorithm is the "order-driven market". It includes aural auctions, continuous electronic auctions ("e-Bay") and crossing networks. Many of the important exchanges in the world are order-driven: ECNs (Electronic Communication Networks) such as Archipelago, Island, and Instinet, the Paris Euronext Bourse and the London SETS are all order-driven. They are based on *order precedence rules* to match buyers and sellers, and on *trade pricing rules* to price the resulting trades. In an order-driven market, trading agents submit orders – buy or sell – to the exchange, at the time, price, and quantity of their choice. The exchange sorts these orders into two stacks, one for buy orders, and one for sell orders. These two stacks constitute what is known as the *book*, or the *order book*. For instance, we can define orders by the following struct:

```
enum orderType {buy, sell};
struct order
{
   orderType type;
   int price;
   double quantity;
   double timeSubmitted;
}
```

The price specified for a buy order is known as the *bid*, and the price of a sell is the *ask*. Trading agents can submit multiple orders if they wish so; the market usually specifies a rule concerning the management of past orders that remain in the stack – for instance, orders older than 30 minutes may automatically expire.

In aural auctions, the primary order precedence rule is *price priority*: buy orders are sorted in descending order of prices, and the highest ("best bid") has priority over all following orders. The trading agent that wants to pay the most for the good sold will get its order filled first. If a seller is found for the buyer that submitted the highest bid, and that this bid price is higher or equal to the ask price of the seller, then a transaction will take place. Likewise, sell orders are sorted in ascending ask price order, and the first (lowest ask) has priority over all following orders:

```
while (market is open)
{
   buyStack[].push(buy orders)
   sellStack[].push(sell orders)
   sort(buyStack[].price, descending)
   sort(sellStack[].price, ascending)
   bestBid  <- buyStack[0].price
   bestAsk  <- sellStack[0].price
}
```

After the buy stack is sorted, the cumulative sum of the quantities demanded defines an *economic demand curve*, depending on price. It is so because if one considers the sequence of prices from the highest down to the lowest, then it is apparent that individuals who submit buy orders at high prices would still be willing to buy at a lower offered price, and therefore the quantities they demand must be part of the aggregated demand for any given offered price. A simple example would be as follows. Suppose we have 100 trading

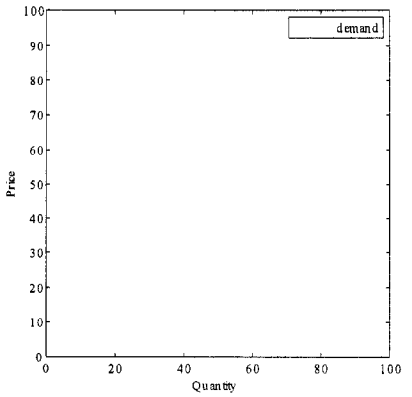


Figure 2.1: Demand curve when each of 100 trading agents bid one unit at prices \$1, \$2, ...\$100.

agents willing to buy one unit of good each, but each posting a different bid price, namely \$1 for agent 1, \$2 for agent 2, etc. Then if a seller willing to sell at a ask price of \$100 shows up, then the total demand of the 100 buying agents will be only one unit of good. If the seller is willing to sell at \$99, then the total demand of the 100 agents will be 2 units of good. This is the case since an ask price of \$99 will satisfy the agent that want to buy at that price, but will also satisfy the agent that wanted to buy at \$100. If someone is willing to buy at a given price, then the transaction would also take place at any lower price as well. Therefore, the total quantity demanded at a given price has to take into quantities demanded at that price as well as quantities demanded at all higher prices. If the price is plotted against quantity demanded, this results in a negatively sloped demand curve (Figure 2.1), familiar to readers of economic texts.

Formally, noting that in the general case each trading agent can demand a different quantity of the good, we define the demand curve as follows:

$$Demand(p) = \sum_{p_i \geq p} \text{buyStack}[i].\text{quantity}$$

Likewise, the cumulative sum of the quantities offered of the sell stack defines a positively sloped *economic supply curve.* Following our previous example, suppose we have 100 trading agents willing to sell one unit of good each, but each posting a different ask price, namely \$1 for agent 1, \$2 for agent 2, etc. Then if a buyer willing to buy at a bid price of \$1 shows up, then the total demand of the 100 selling agents will be one unit of good. If the buyer is willing to buy at \$2, then the total demand of the 100 selling agents will be 2. This is the case since an bid price of \$2 will satisfy the agent that want to sell at that price, but will also satisfy the agent that wanted to buy at \$1. If someone is willing to sell at a given price, then the sale would also take place at any higher price as well. Therefore, the total quantity supplied at a given price has to take into quantities supplied at that price as well as quantities supplied at all lower prices. If the price is plotted against quantity supplied, this results in a positively sloped demand curve (Figure 2.2).

Formally, we have:

$$Supply(p) = \sum_{p_i \leq p} \text{sellStack}[i].\text{quantity}$$

Next, the exchange algorithm checks whether the highest bid is higher or equal to the lowest ask, namely whether there is a buyer that is willing to pay the asking price of a seller or better; if a match is found, a transaction is made and the book is cleared of the orders that were filled. The process continues until the best bid is strictly lower than the best ask, when the trading agent willing to buy at the highest price is not willing to pay the

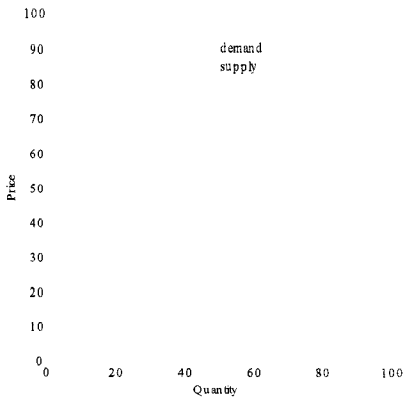


Figure 2.2: Supply curve when each of 100 trading agents ask one unit at prices $1, $2, ...$100.

price of the lowest seller. At that point, the set of buyers consider the good too expensive, and nobody in the set of sellers wants to lower its price. The algorithm then halts and waits for new orders to come into the book until it can be cleared again:

```
// book clearing process
bestAsk <- sellStack[0].price
bestBid <- buyStack[0].price
While (bestBid >= bestAsk)
{
   // check whether best buyer wants more than what best seller offers
   if (buyStack[0].quantity >= sellStack[0].quantity)
   {
      buyStack[0].quantity =- sellStack[0].quantity // residual demand
      sellStack[0].pop                              // remove first sell order
      bestAsk <- sellStack[0].price                 // reset bestAsk
   }

   // check whether best seller wants more than what best buyer offers
   if (buyStack[0].quantity <= sellStack[0].quantity)
   {
      sellStack[0].quantity =- buyStack[0].quantity // residual offer
      buyStack[0].pop                               // remove first buy order
      bestBid <- buyStack[0].price                  // reset bestBid
   }
}
```

In terms of demand and supply curves, a transaction takes place when the curves intersect, but the "equilibrium" price at which trading occurs is *transient*, as orders are added and removed dynamically from the stacks. It is therefore important to note that when economists refer to markets to be "in equilibrium", what they actually have in mind is a series of equilibriums that are constantly changing. Orders may be removed because they become stale, or too far from the current price action, or because they are filled, and orders may be added because traders have new information, or because they want to be closer to the current price action.

The main drawback of order-driven markets is that they can stay idle for extended periods of time; this corresponds to the situation in which all buyers consider prices offered by all sellers to be too high. Consequently, the "market price"[2] does not fluctuate during these waiting periods: it is stalled since the moment the last transaction took place. This is inconvenient in many ways, for instance because it prevents outsiders to use the current market price as current value indicator; a stalled market will offer no information on the recent evolution of stock values. In the era of real-time Internet stock quotes, it should be obvious to everyone that the prototypical modern stock exchange does not operate in this way: it never stops fluctuating. This is characteristic of *quote-driven markets*, which we discuss next.

[2]the question as to whether a "market price" exists when there are no transactions is in a sense similar to the question whether a tree falling in a forest makes a noise when nobody is there to listen. It is just a matter of convention: one could say that a market price can only exist if there is a transaction, but one could posit a theoretical model of a "fair price" that is independent of the existence of transactions. Heisenberg would have disagreed; in order to have a price, one must have an observation, namely a transaction.

2.2.2 Quote-driven markets

Investors know that when the exchange is open, one can trade stocks in a matter of seconds, regardless of whether the opposite order is present in the book. The reason is that in most modern stock exchanges, trading agents do not need to find an opposite order to trade; they trade with a professional (hidden) market *mediator* called a *market-maker.* Without knowing so, trading agents actually sell their stocks to that mediator, and buy their stock from it. The presence of this mediator guarantees immediate execution of all orders, unless the orders explicitly call for a condition to be realized before execution[3].

Examples abound: for instance, the New York Stock Exchange (NYSE), the American Stock Exchange (AMEX) and the NASDAQ (National Association of Securities Dealers Automated Quotation system) are all quote-driven. The NYSE and AMEX have unique market-makers for each stock, and the NASDAQ has multiple competing market-makers per stock.

Market-makers provide *immediacy* and *liquidity*, which are desirable market properties. Immediacy comes at a cost to the public, and at a risk to the market-makers themselves. They must indeed have inventories of stock, and be prepared to sell stock to the public from their inventory, in the event that nobody else wants to sell. To illustrate what may happen, suppose a bad news becomes suddenly known and causes most trading agents to want to sell at the same time. Market-makers are still obligated to "make a market", that is to play the counterpart to these sellers. In so doing their inventories will increase to dangerously large levels — which is a liability, as nothing guarantees that the

[3]for instance in "limit orders" and "stop orders". A limit order opens a new position when a rule involving prices fires; a stop order closes a position when another rule fires. Rules are specified by the trading agents.

newly acquired stock could be profitably sold at a higher price any time later. If the general market mood is "bearish"[4], prices are unlikely to rise, and a large inventory spells trouble.

In order to control their inventories, market-makers are allowed to *quote prices.* This means that the current going price of a stock *is totally up to their decision.* That does not mean that they choose prices randomly, quite to the contrary. If for instance there happens to be an overwhelming desire on the part of market participants to sell stocks, the market-makers will respond by lowering their quotes as a defence mechanism, forcing the public to get less and less income when they sell. This decision will therefore discourage selling impulses, since people sell things that they feel are overvalued. Plummeting quotes not only discourage selling, but encourage opportunistic buying, which lowers inventory misbalance. This depends of course on how overvalued the stock was in the first place. The precise amount by which quotes are going to be changed depends on how elastic market demand with respect to price changes is. Knowing how to ascertain this elasticity and adjust quotes is an important skill that successful market-makers must acquire. We call this the *price-adjustment* mechanism.

Excessive stock supply is therefore countered by lowering quotes. However, controlling inventory and maintaining profitability are two contradictory objectives that must be reconciled with *doigté*: a risky large inventory, acquired in an environment of high prices, must be liquidated by selling stock to the public; however, the public can be persuaded to buy only if prices are low. So it seems at first view that market-makers would logically lose money on average: inventories accumulate at high prices, and are liquidated at low

[4]Bears are the most feared beasts of the Wall Street bestiary, and symbolize downward trending markets. This is supposed to come from their favourite attack posture, which is to stand on their hind legs, and come down onto their prey. Bulls, symbolizing upward trending markets are supposed to attack by raising their heads.

prices, which is a case of "buy high, sell low". Market-makers must therefore be vigilant and change their quotes as soon as they notice a misbalance between stock demand and stock supply; good market-makers should also anticipate price movements.

Since market-makers are expected to incur inventory losses, they are allowed to take a small profit on each round-trip transaction. This is done through the *bid-ask spread*: market-makers have the right to quote their buying price (bid) lower than their selling price (ask). That way, if the public submits buy and sell orders at the same time, the market-maker fills these orders and pockets the difference between the bid and ask prices, without any adverse change in his inventory. This bid-ask spread can be lowered or widened according to market conditions: if transaction volume is low or if uncertainty is high, the spreads tend to be wider. Generally speaking, the goal of the market-maker is to sandwich the "real equilibrium price" between his bid and his ask, thereby getting about the same frequency of buy and sell orders. This is the best guarantee that his inventory will remain under control.

The market-making activity can also be understood in terms of demand smoothing. Market-makers absorb the short-term price jumps and therefore lower the probability of crashes.

Market-makers pay dearly for their seats at the exchange[5]; owning one is generally a profitable business, even though some market-makers bankrupt from time to time. Good market-makers have to perform a balancing act. First and foremost, they have to survive by keeping profitability in the comfort zone, and their inventory relatively low. This depends on

[5] a market-maker seat at the NYSE is currently worth in excess of $1M per year. That price is an indication of the potential profits one can make in that position.

their quick judgment of current market moods (through publicly available information such as newsfeeds and other media, and a private network of close friends in strategic places), and quick price adjustment after new information arrives. Market-makers know that informed investors such as insider traders will hurt their profitability if allowed to trade large blocks of shares at the quoted price, as they actually know something about a company that is not in the public domain yet; therefore market-makers develop heuristics to guess who is genuinely informed, and who is not. Prudence dictates not to accept large orders at any quoted price. Profitability is not the only thing that is important: from the point of view of the market regulator, price efficiency also is. It would be easy for a lazy market-maker to post outrageously unfair prices to ensure profitability; however, trading volume would be curtailed. It is therefore important for a market-maker to post new prices that are not too far away from where the current market demand and supply is. This property is called *price efficiency* and is closely monitored by market watchdogs.

2.3 Artificial markets

The computer being a universal simulation machine, it is not surprising that some researchers have sought to reproduce *in computero* complex economic systems. The economy is after all composed of various specialized agents having reasonably well-defined goals and interacting within a framework that has clearly defined rules, which maps admirably to the multiple agent paradigm. However, the possible angles of attack are infinite, as well as the choice of which particular area one wants to emphasize.

As a consequence, the artificial markets literature offers a wide variety of simula-

tions, with usually very little common ground. Due to space limitations, we only present here a few examples. A nice way to order the different markets is to classify them with respect to their *motivations*.

2.3.1 Motivations of artificial market modellers

Any complex system can be simulated at various levels of detail, and model-builders need to focus on specific questions they want to shed light on. The two most popular motivations artificial markets are used for are:

- as a pseudo-empirical benchmark to either refine, confirm or falsify existing economic theories, or
- to determine how best to trade.

Artificial markets as pseudo-empirical benchmarks

Since Adam Smith, classical economic theory has posited that the freer the markets, the most efficient the price mechanism is in allocating goods. This is the proverbial "invisible hand". Neoclassical economics, founded by Walras in 1874 [57], suggested a formalism borrowed from the energy equations of physics to modelize value, which confirmed and extended the classical view on free markets. However, unlike physics who deals with object having no free will, economic systems are composed of agents whose behaviour may not be perfectly rational. Even if agents want to be rational, they may have limited processing power and little time to make decisions, which may bound their rationality. Moreover, in reality, markets are affected by so-called "frictions", such as taxes and transaction costs, and

may deviate from equilibrium because frictions are not taken into account in the neoclassical equations. The question is therefore to determine the extent to which these equations describe the economic reality (Mirowski, 1989 [49]), or are simply a case of "physics envy".

Artificial markets built for that purpose are for instance that of Gode and Sunder 1993 [25]. The main result of their simulation was to show that market constraints (for instance budget constraints and restrictions on short sales) are sufficient to create rational allocation, even if traders have "zero-intelligence". This is quite interesting, since even if agents do random trading (which is of course not the case in real markets), markets can still reach the point where the going price is going to reflect the demand and supply conditions.

Levy et al., 2000 [44] explored the impact of abolishing the classical assumptions of trader homogeneity, perfect rationality and perfect information. Classical theory is indeed unable to explain empirical puzzles that are posed by ubiquitous properties of real stock time-series, such as fat-tailed return distributions and excess volatility. They showed that all these unexplained characteristics automatically emerge as a default feature of simulated prices as soon as any of the classical assumptions are removed. Their book actually describes a variety of different artificial markets structures that were designed to remove each of the classical assumptions at the time.

Determining optimal trading strategies

Some studies are interested in determining whether optimal trading strategies exist, whether they are stable and what their effect is on market equilibrium. For instance, the simulation in Joshi et al., 1999 [34], and 2002 [35] concludes that under realistic trading, markets never reach an equilibrium, and could never be efficient. This is interesting and

apparently contradicts the results of Gode and Sunder 1993 [25] we just mentioned about.

A research group at the Santa Fe Institute (Arthur et al. 1994 [2], 1997 [3] and LeBaron et al. 1999 [43]) has proposed an open-source artificial market code on the SWARM platform, the "Santa Fe Artificial Stock Market", that has attracted much interest. In a series of papers describing experiments on the platform, they explored the consequences of evolving group learning dynamics (using genetic algorithms) on the profitability of trading strategies. Their conclusion is that no trading strategy is "best", since any successful strategy is imitated by other trading agents and eventually loses profitability. The proportion of different kinds of trading rules in the market perpetually oscillates in a dynamic reminiscent of the Lotka-Volterra predator-prey system. Their finding is in agreement with what traders usually experience in real markets, although it is not clear whether the decrease in profits of trading strategies might as a matter of fact be due to data-snooping.

In general, it is important to bear in mind that while these simulations are interesting, any result on optimal trading strategies must be taken *cum grano salis*. The freedom that artificial markets modellers enjoy is actually not an advantage: many design decisions need to be taken before running the simulation. Choices of design may have a critical impact on the results, but cannot be controlled for easily, unless one runs and compares the results of a very large number of different market specifications, which becomes tedious. Since there are many models, it is also important to bear in mind that any result obtained in the context of one specific simulation may not necessarily hold in the context of another.

2.3.2 The question of calibration

Finding a way to calibrate a simulation to a real phenomenon is an important and hard problem — it may as a matter of fact be impossible. For instance, the current controversy about the reality of global warming is a case in point. Global weather simulations must by nature neglect or simplify some aspects of reality; this is imposed by computational constraints and by the increasing complexity of modelling fine levels of detail. So any conclusion taken from such a simulation, especially if it is projected for a long period in the future, is suspect. We cannot be sure whether it reflects the real system or an impostor.

There are few examples in the literature of modellers that have attempted to calibrate artificial markets to real ones. LeBaron 2003 [42] shows how a SWARM-based[6] market simulation can be constrained to numerically reproduce a given level of market volatility[7], which may be of use for risk management simulations. Fagiolo et al. 2006 [21] discusses the validation problem, and Ecemis et al. 2005 [19] invert a parsimoniously parameterized financial market. In our knowledge, this last paper is the only one in the literature (except from this dissertation) that endeavours to invert a market, and it directly inspired our approach. There are important differences between their work and ours, though: Ecemis *et al.* 2005 [19] do not invert a full artificial market, but only a very roughly parameterized version. For instance, trading agents derive their decision rules from a beta distribution, and a market is described by the two parameters of that beta distribution. Inversion only requires the reconstruction of these two parameters. This considerably restricts the range of behav-

[6]SWARM is a Java and Objective-C platform originally developed at the Santa Fe Institute for multiple agent simulation. A Wiki can be found at www.swarm.org

[7]Volatility is defined as the annualized standard deviation of stock returns computed according to the continuous compounding convention.

iours that traders can exhibit. Another regrettable consequence of having a distribution of trading agents is that no usable financial forecast can be made from the reconstructed market.

Chapter 3

Statement of the problem

Our objective is to use historical prices of a given stock to reconstruct an artificial market *that would have generated these prices deterministically in the past.* Once the market is reconstructed, it is simulated forward to append more prices to the historical sample: these new prices are considered as a forecast (figure 3.1). Since this reconstruction is stock and period-specific, it has to be re-done individually for each stock and each period we wish to forecast.

We therefore need to define a model of a deterministic artificial market *that is simple enough as to allow reconstruction from data, but complex enough as to reflect some financial structure, namely the basic agents of a quote-driven market.* We have seen from chapter 2 that these agents are the traders and the market-maker. By imposing this financial structure on the reconstruction (as opposed as an *ad hoc* genetic programming-derived function), we hope to capture (to a degree that we can test) the decision rules of the various trader blocks that are influencing a given stock price. Figure 3.2 shows the basic price

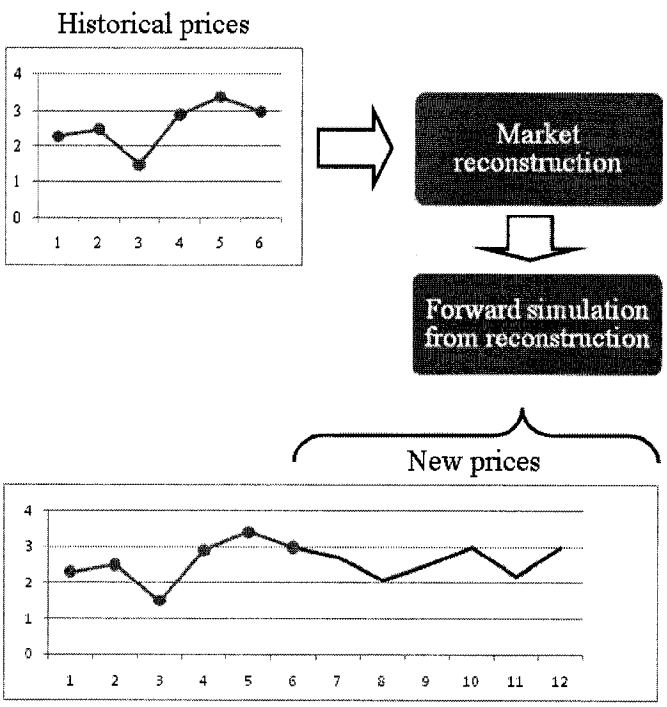


Figure 3.1: General steps for forecasting

generation mechanism of the artificial quote-driven market.

We consider a market as a *population of trading rules* interacting through a mediator, called *market-maker*. We explain now what is meant by a "trading rule".

3.1 Trading rules

A trading rule is a IF–THEN construct based on a certain number of past prices relative to the current moment in time. When the logical expression in the IF part evaluate to TRUE, a *market position* is established for a specific amount. There are two kinds of market positions: a "*long*" (buy) or a "*short*" (sale). Since there is a whole population of trading rules, only a certain subset of rules will fire under a given series of past prices, generating demand that varies through time.

To illustrate a trading rule that is used in reality, consider so-called *moving averages* (MA) from the *technical trading* literature. Technical trading refers to the practice of looking at past prices to infer a normative market position, a practice that is not sanctioned by standard finance theory. Indeed, having already mentioned that the canonical model of stock prices being the lognormal random walk, any pattern of past prices emerging from that stochastic process should not be the basis upon which any investment decision should be taken. Despite this anathema from academia, technical trading is widely spread among finance practitioners and its prevalence is believed to have some effect on prices by shifting demand, as a manner of self-fulfilling prophecy. The reader should note the phrase "moving average" has two meanings in this paper, one referring to the technical trading literature, which we employ when speaking about trading rules, and the other to the ARIMA statistical

model described in chapter 6.

Suppose we take averages on a window of past prices that is computed in a rolling fashion: the value of that indicator on any given day is for instance the average of the 26 preceding daily prices. A moving average is therefore a low-pass filter; it reveals the long term tendency and wipes out small details. Technical traders define a trading rule by comparing the relationships between technical moving averages of various lengths, for instance the 26-day MA and the 12-day MA, according special significance to the points at which the two graphs cross. If the shorter-term MA crosses the longer-term MA from below, it shows a recent upward trend, which is a buy signal. Inversely, if the shorter-term MA crosses the longer-term MA from over, a sell signal is generated. An illustration of this rule is on Figure 3.2, where two buy and two sell signals were generated over the course of two years.

Note that while the rule seems to make sense judging from the graph, following the signals resulted in a negative return of -1.5% over the period: this was due to the excessive lag in the signal generation. So trading rules may well fail to be profitable over specific periods of time. In comparison, a simple buy-and-hold resulted in a positive return of 6.8%. Naturally, looking at only one example is not sufficient to make a judgement about the value of a trading rule: sophisticated traders usually conduct statistical analysis of trading rules profits before adopting them. Putting the issue of performance aside, an advantage of trading rules is that their signals are generated purely by means of a mechanical calculation, and therefore take human subjectivity out of the equation; moreover, any strategy can be back-tested. A trading rule can generate false positives or fail to take profitable positions;

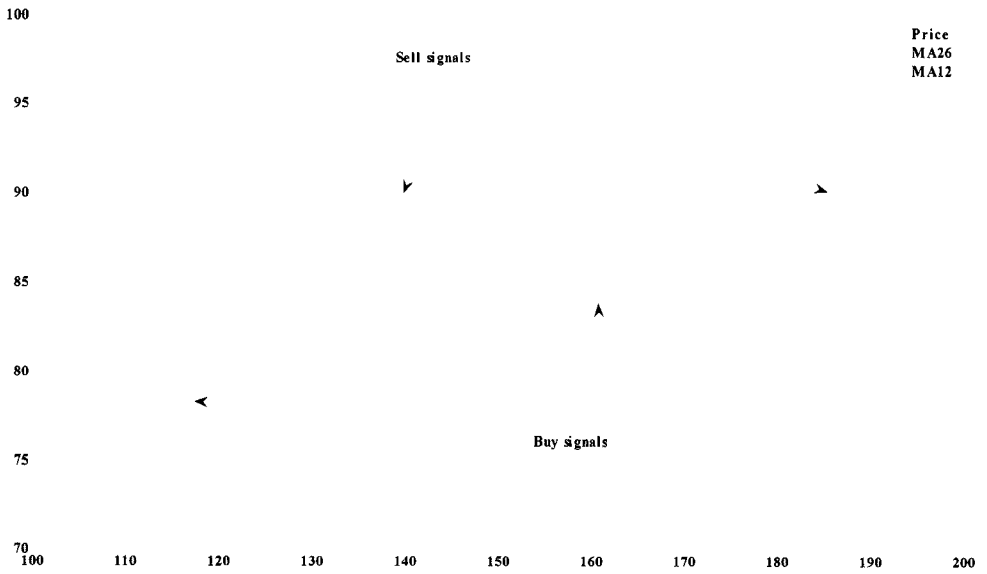


Figure 3.2: Buy and Sell signals generated by the MA26-12 trading rule on IBM, 2005-2007

this particular instance (MA26/12) also requires optimization of the two length parameters, which is sensitive to the data-snooping bias. The choice of 26 versus 12-day moving averages is popular in the industry (Achelis 2003 [1]).

Moving average-based rules are only one example of the hundreds of rules that are used by traders in the financial markets. There are rules based on the difference between the maximum and minimum of prices over a past window, rules the look at regressive trends through these maxima and minima, rules based on Fourier analysis, etc.

3.2 Markets and forward simulation

Suppose we have a population of rules. Since rules are conditional constructs, only a given subset of the population will fire at any point in time, based on the past prices at

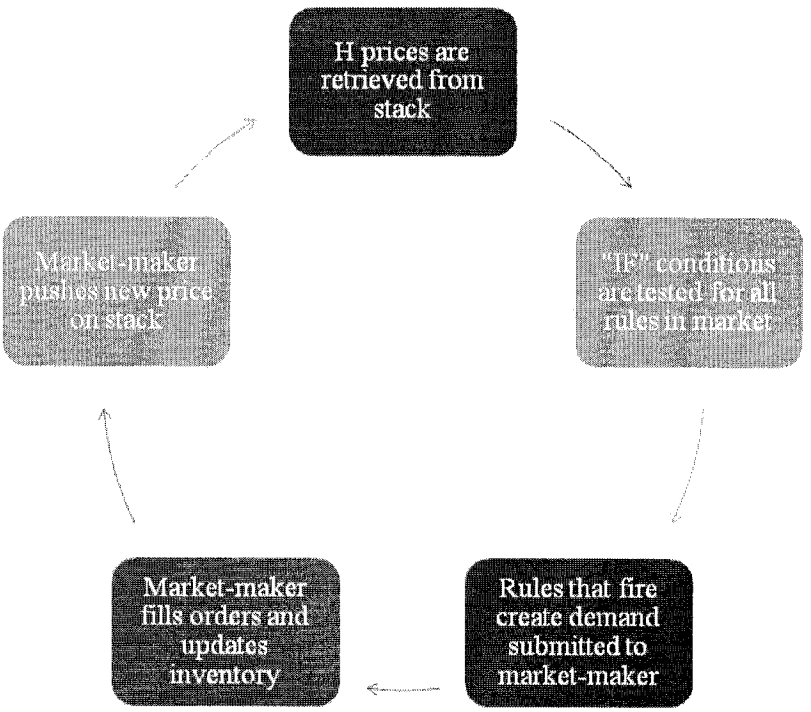


Figure 3.3: Forward simulation of prices generated by a market as a population of rules

that moment. This subset of firing rules will constantly change as the series of past prices evolves. We will make the following simplifying assumptions:

1. time flows in discrete steps, and

2. all the rules from the population need at most H preceding prices to evaluate their "IF" condition.

We can put the past prices in a stack: the older prices are at the bottom of the stack and the earlier on top. The basic forward market simulation is a loop that generates new prices as follows (figure 3.3):

- the market copies the H most recent prices from the price stack.
- all the "IF" conditions are evaluated on the H recent past prices, causing a subset of rules from the population to fire.
- firing rules create *orders* that are put in the book and immediately submitted to the market-maker. These orders constitute a market demand for the current moment in time.
- the market-maker fills the orders at the current price and computes his new stock inventory.
- the time-counter is increased; based on his inventory level, the market-maker pushes a new price on the stack.

This process is repeated until as many prices as desired are generated. To initialize the stack, one may take a series of historical prices from a real market.

Prices generated in this manner are deterministic, and depend on the initial stack and on the exact nature of the population of rules. Changing the initial conditions will completely change the appearance of the time-series of generated prices, since it will cause a different subset of rules to fire, which will generate a different inventory and cause the market-maker to push a different price on the stack, changing the orbit forever. We can also see that the presence or absence of a single rule in the population may also affect the time-series in a nonlinear fashion.

This artificial market setup is therefore a dynamical system: we have sensitivity to initial conditions and nonlinearity, which are the hallmark of chaos. Chaos is not always

guaranteed, though: oscillatory behaviour is also possible. It may indeed happen that a certain configuration of rules will generate a series of prices that will exactly reproduce the initial conditions, thereby causing a repetition of the same firing pattern.

Let's finally turn our attention to the market-maker, which is the last element of the artificial quote-driven market.

3.3 The market-maker

The market-maker is the agent that is responsible for posting new prices based on the state of his inventory. This process can be very complex and require sophisticated artificial intelligence, as we mentioned in chapter 2. We want to stylize this behaviour as much as possible here. We will therefore only require to define is a function that can be called "market-maker response function", or "price adjustment function" $f : \mathbb{R} \longmapsto \mathbb{R}$, with the following minimal *desiderata*:

- if the value of the input inventory is high (an indication that the market-maker has recently bought many stocks), then the current price is too high: too many trading agents want to sell. The function should therefore return a lower price, which will discourage agents from selling and push the inventory to a value closer to zero.

- if the value of the input inventory is low – meaning in this case that it is a large negative number (an indication that the market-maker has recently sold many stocks), it means that the current price is too low: too many trading agents want to buy. The function should therefore return a higher price, which will discourage agents from buying and push the inventory to a value closer to zero.

These properties are obvious; to this we may add that from a statistical point of view, we want to have the following further *desiderata*:

- "low" and "high" values of inventory should not be defined absolutely, but relative to customary inventory levels on that market. It would be nice to find a way to estimate for each stock and market what would constitute a "low" or a "high" value. For instance, some stocks do not have many traders and change in value slowly. Therefore, market-makers acting on these stocks would carry levels of inventory that would be markedly lower from market-makers acting on popular stocks.

- prices posted by the function should not depart drastically from past prices; we should ideally find a way to use the past prices as a statistical benchmark for what constitutes a reasonable change in price from the price-adjustment function.

Finally, from a modelling point of view, we may require that the market-maker function not have too many parameters, since our problem gets harder the more parameters there are to estimate. Ideally, function parameters should be defined automatically from past prices.

3.4 The problem

We now recapitulate. We would like to define *a method* to *infer (reconstruct)* the artificial market that could have generated a given historical time-series of prices. This artificial market should have basic financial structure and be composed of:

- a population of trading rules

- a market-maker

This market, once reconstructed, could then be forward simulated to generate a forecast.

Notice that "reconstruction" implies that *the original historical time-series of prices would be generated if the reconstructed market were forward simulated starting from the historical initial conditions.* Forward simulation is therefore a process that can be not only be used to obtain price forecasts, but also used to verify the accuracy of the reconstruction as well.

The object of the next chapter is to explain how to setup genetic algorithm to reconstruct a market.

Chapter 4

Model definition

4.1 The artificial stock market

4.1.1 Rules based on discretized returns

Instead of working with prices, we work with discretized returns. This has advantages for rule representation in genomes. According to the continuously compounded interest rates convention, returns over the past n days are computed as:

$$r_{t-n,t} = \ln\left(\frac{S_t}{S_{t-n}}\right)$$

, where S_t is the stock price at day t and S_{t-n} the stock price at day $t-n$. Figure 4.1 shows the price and return views of the same time-series. The first panel also shows the two moving average indicators from section 3.1.

The idea behind the moving average rule was that if recent prices grow faster than less recent prices, a buy signal is in order. This suggests that an alternate way of writing

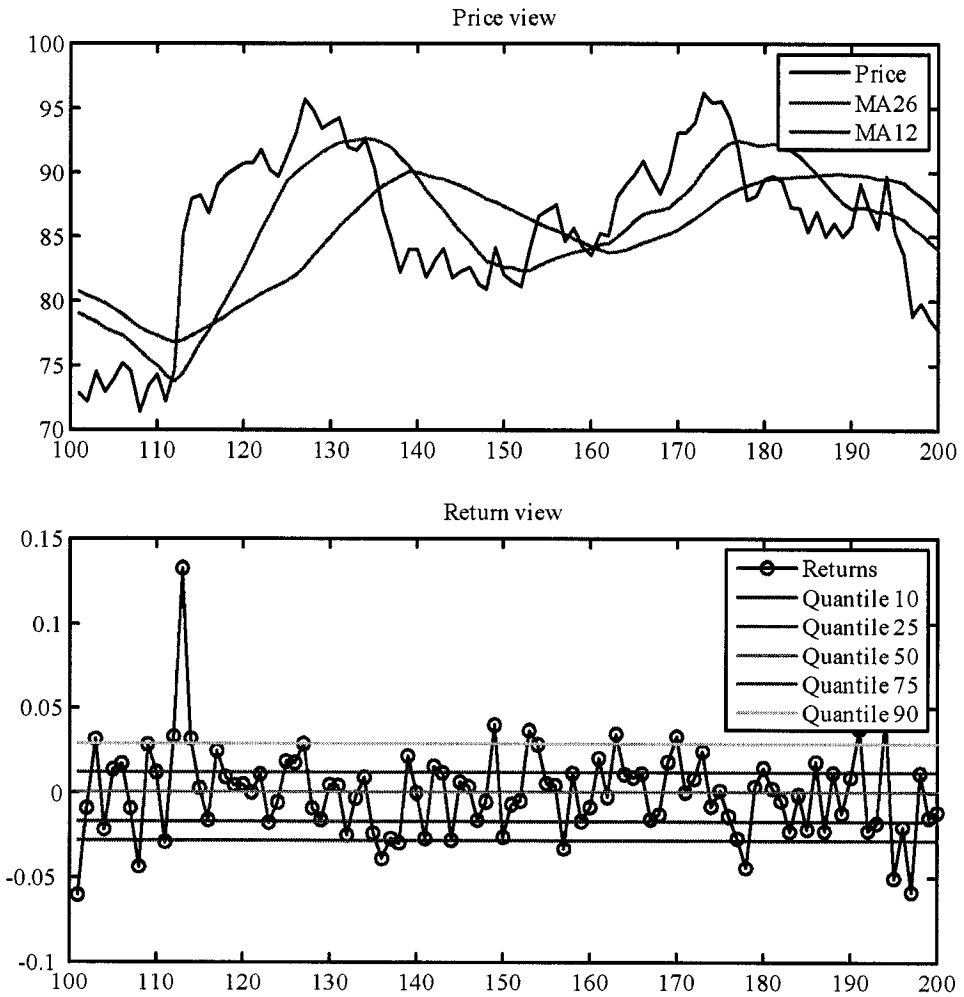


Figure 4.1: Price and return views of IBM 2005-2007. Top: prices and moving average indicators. Bottom: corresponding returns and selected return quantiles.

the rule would be to compare the rates of growth of recent prices to rates of growth over a longer period. Other types of rules, based for instance on regressive trends, would look for persistence in rates of growth, etc. This rule "translation" is possible because patterns in prices must also be present in returns, which are a transformation of prices.

The second panel of figure 4.1 shows the return percentiles superimposed to the returns. The area in which a given return lies shows how average or exceptional it is. Around observation 112 in return view, one notices a return well-beyond the 90^{th} percentile, which indicates that prices are growing fast recently. Not surprisingly, a few days later, a buy signal is generated from the MA 26/12 rule in the price view. This correspondence is not perfect because the price-based MA rule looks at averages 12 and 26 days back in the past and is therefore subject to some hysteresis. To find the returns-based rule that would generate the same signals as the MA price-based rule, one would have to take into consideration the return pattern of the past 26 days. This suggests that *trending patterns in prices can be translated into size patterns in returns.*

Let us now give an example of that principle by translating a variant of the MA rule we just explained to a return size decision rule, with a few simplifying assumptions. Assume for instance that the rule compares the 4-day MA to the 2-day MA, a signal being generated when one exceeds the other. Denote by S_{-4} the price four days ago, $r_{-4,-3}$ the return from day -4 to day -3, etc. Since returns are continuous, we use geometric averages of prices instead of arithmetic averages to provide easy manipulation. Under these assumptions, the long (4-day) moving average indicator is:

$$\begin{aligned} MA(4) &= \sqrt[4]{S_{-4} \cdot S_{-4}e^{r_{-4,-3}} \cdot S_{-4}e^{r_{-4,-3}+r_{-3,-2}} \cdot S_{-4}e^{r_{-4,-3}+r_{-3,-2}+r_{-2,-1}}} \\ &= S_{-4}e^{\frac{3r_{-4,-3}}{4}+\frac{2r_{-3,-2}}{4}+\frac{r_{-2,-1}}{4}} \end{aligned}$$

and the short (2-day) moving average is:

$$\begin{aligned} MA(2) &= \sqrt[2]{S_{-4}e^{r_{-4,-3}+r_{-3,-2}} \cdot S_{-4}e^{r_{-4,-3}+r_{-3,-2}+r_{-2,-1}}} \\ &= S_{-4}e^{\frac{2r_{-4,-3}}{2}+\frac{2r_{-3,-2}}{2}+\frac{r_{-2,-1}}{2}} \end{aligned}$$

so the "buy" signal generated when MA(2) exceeds MA(4) can be written as a condition involving past returns as:

$$r_{-4,-3} + 2r_{-3,-2} + r_{-2,-1} > 0$$

One may justifiably object that this representation accounts for only half of the original MA rule signals, namely the buy side only. This is true but actually immaterial, as we represent markets as *populations* of generalized trading rules, so the second half of the rule can also be present in the population (or not). We can break in this fashion many a complex rule into simpler parts.

Returning to figure 4.1, the reader can see that we plotted return quantiles along returns in the second panel, and that in most cases, one can imagine that not much is lost in substituting return quantiles in lieu of the precise numerical values of the returns. More specifically, we will replace returns by their respective octiles. This is a design decision that has a number of implications, and can be justified by four reasons:

1. It allows for sparsity in rule representation — a characteristic that will become important as we generate large numbers of rule populations in memory.

2. It renders rules less specific, hence more easily activated in case an approximate pattern arises. This is usually what real traders do.

3. If one constantly updates octiles on the fly (using the n most recent observations), rules become dynamic, as opposed to static. Static rules might never get activated is certain market contexts, even when the pattern they are designed to respond to are present, but "shifted".

4. Trading rules are but heuristics; there is (probably) little downside in expressing the data in a granular fashion since there is no precise science of trading. There in fact might we be some advantage in doing away with precise returns, as often happens in the field of machine learning; in some situations, neural nets may have a tougher time (or fail) to generalize when presented with precise data, as opposed to appropriately granularized data.

The choice of octiles as opposed to any other percentile is arbitrary but can be changed if desired; octiles offer an intuitively acceptable compromise between data granularity and rule specificity. Since continuous returns are centered around zero, choosing octiles gives us four different grades of down markets – that we could paraphrase as mildly down, moderately down but less than average, down and more than average, and severely down – and four different grades of up markets, which is probably sufficient for most practical purposes.

We now turn to a general representation for trading rules, that subsumes fundamental as well as technical schemes, and that can be represented with a variable or fixed-length chromosome. We distinguish variable and fixed-length generalized trading rules:

Variable-length generalized trading rules

We first define the *empirical cumulative distribution* of 1-day continuously compounded returns. An empirical cumulative distribution is obtained by sorting returns taken from a sample of historical observations, assuming that each sample return is equiprobable. The Glivenko-Cantelli theorem states that the empirical distribution of a set of independent and identically distributed variables tends to the limit to the real distribution when the sample size tends to infinity (Shirayaev 1989 [55]). Formally, if we have a sample of n return observations x_1, x_2, x_3, ..., x_n , the empirical cumulative distribution $\widehat{F_n[x]}$ is defined as the number of observations that are that are less than or equal to x, divided by the size of the sample n. If the sample is sorted such that $x_1 \leq x_2 \leq x_3 \leq \ldots \leq x_n$, then we have:

$$\widehat{F_n[x]} = \frac{1}{n} \cdot \#\{i;\, 1 \leq i \leq n;\, x_i \leq x\}$$

, where the symbol # denotes the cardinality of the set. This empirical distribution is used to extract empirical return octiles, which are the returns for which $\widehat{F_n[x]} = \frac{1}{8}$, $\widehat{F_n[x]} = \frac{2}{8}$, etc. We refer to these octiles as $r(o_1)$, $r(o_2)$, $\ldots r(o_8)$. Due to the nature of continuously compounded returns, and the fact that in weakly stationary markets the median return is about zero, it follows that return octiles for n days are approximately a simple linear scaling of the 1-day return octiles $n \cdot r(o_1)$, $n \cdot r(o_2)$, $\ldots n \cdot r(o_8)$[1]. Once the

[1]An important caveat here is that this assumes independence of daily returns, which is usually not

octiles are obtained, each return is coded in terms of which octile it belongs to. Denoting the present moment as time t, we write the n-days return from $i-n$ days back in the past to i days back as $r_{t-i-n,t-i}$. We denote the coded return in term of octiles as $\left\langle r_{t-i-n,t-i} \right\rangle_8^n \in \{1,\ 2, \ldots,\ 8\}, \forall n, \forall i$.

We define the *rule lookback* H as the maximum number of past periods a rule considers in order to make its decision. A variable-length generalized trading rule is a quadruple containing the following elements:

- A symbol taken from the set $\{B, S\}$, which determines whether a positive identification of the rule's condition results in a buy (B) or a sell (S) order.
- A variable-length vector $v_{boundary}$ of integers taken between 1 and $H-1$ (inclusively), selected without replacement. These numbers define the boundary points for return calculations. This vector of integers needs not be sorted in the genome (since application of genetic operators are going to scramble their order), but is sorted when interpreted. For instance, if $H = 10$ and the numbers selected are 8, 2 and 5, this means that the rule will look at returns from $t-2$ to present (a 2-days return), $t-5$ to $t-2$ (a 3-days return), $t-8$ to $t-5$ (a 3-days return) and $t-10$ to $t-8$ (a 2-days return). If the vector has k elements, then k+1 periods are implicitly defined.
- A vector $v_{thresholds}$ of $2(k+1)$ integers $t_\alpha, t_\beta \in \{1,\ 2,\ \ldots 8\}$, one pair for each interval defined implicitly by vector $v_{boundary}$. Again, it is not necessary to keep the pairs sorted.

strictly true, but nearly true. Of course, true values can be used instead of the approximation, as a greater computational cost.

- A vector $v_{conditions}$ of $k+1$ symbols taken with replacement from the set $\{a,\ b,\ c,\ d\}$. These define four possible relationships between the returns on the intervals defined in $v_{boundary}$ and the corresponding threshold numbers t_α and t_β from $v_{thresholds}$. For instance, if a given interval is from $t-i-n$ to $t-i$:

1. $\left\langle r_{t-i-n,t-i} \right\rangle_8^n \leq t_\alpha$, (denoted by the symbol "a")
2. $t_\alpha \leq \left\langle r_{t-i-n,t-i} \right\rangle_8^n$, (denoted by the symbol "b")
3. $t_\alpha \leq \left\langle r_{t-i-n,t-i} \right\rangle_8^n \leq t_\beta$, (denoted by the symbol "c")
4. $(\left\langle r_{t-i-n,t-i} \right\rangle_8^n \leq t_\alpha) \vee (t_\beta \leq \left\langle r_{t-i-n,t-i} \right\rangle_8^n)$, (denoted by the symbol "d")

Note that relationships "a" or "b" require the definition of only one threshold number, and in this case, the rule simply ignores the second number t_β from the pair. This second number is however kept in the chromosome and may become expressed in later generations if transmitted to the offspring. Also note that "don't care" conditions can occur in multiple ways in this representation: for instance, with relationship "a", when $t_\alpha = 8$, with relationship "b", when $t_\alpha = 1$, with relationship "c", when $t_\alpha = 1$ and $t_\beta = 8$, and with relationship "d" when $t_\alpha = t_\beta$.

Consider for instance the following simple rule quadruple with H=10:

$$(B,\ (8,\ 2,\ 5),\ (5,\ 1,\ 3,\ 2,\ 6,\ 3,\ 1,\ 8),\ (b,\ b,\ d,\ a))$$

This quadruple identifies a "buy" condition as a run in which we must observe the following return conditions for the rule to fire:

$$5 \leq r_{t-2,t}$$

$$3 \leq r_{t-5,t-3}$$

$$(r_{t-8,t-5} \leq 3) \vee (6 \leq r_{t-8,t-5})$$

$$r_{t-10,t-8} \leq 1$$

For any given simple rule quadruple, we may define its *reciprocal rule* as the rule in which the following symbol substitutions are made:

- $S \leftarrow B$ and $B \leftarrow S$.
- in $v_{conditions}$, $a \leftarrow b$, $b \leftarrow a$, $c \leftarrow d$ and $d \leftarrow c$
- when the original condition for a threshold pair t_α and t_β was "a", then $t_\alpha \leftarrow \min(t_\alpha + 1,\ 8)$.
- when the original condition for a threshold pair t_α and t_β was "b", then $t_\alpha \leftarrow \min(t_\alpha - 1,\ 1)$.
- when the original condition for a threshold pair t_α and t_β was "c", then
  - $\min(t_\alpha,\ t_\beta) \leftarrow \max(1, \min(t_\alpha,\ t_\beta) - 1)$, and
  - $\max(t_\alpha,\ t_\beta) \leftarrow \min(8, \max(t_\alpha,\ t_\beta) + 1)$.
- when the original condition for a threshold pair t_α and t_β was "d", then
  - $\min(t_\alpha,\ t_\beta) \leftarrow \min(8, \min(t_\alpha,\ t_\beta) + 1)$, and
  - $\max(t_\alpha,\ t_\beta) \leftarrow \max(1, \max(t_\alpha,\ t_\beta) - 1)$.

A reciprocal rule is therefore activated under "opposite" conditions than those of the original rule, and does the reverse transaction. The rationale behind this definition is

to offer the possibility to define collections of rules that operate symmetrically, which we can call "oscillators"[2]; oscillators in this sense would be created by adding the reciprocal of each rule to the original collection of rules. We can further define a "symmetrical market" as one composed of oscillators, but this terminology is not standard in finance.

Symmetrical markets offer a certain level of balance between buy and sell activities; we could hypothesize that symmetry implies stationarity in the resulting artificial price time-series when the number of rules is large. However, we do not investigate that hypothesis for reasons of space.

We have not yet mentioned which amount of money rules are betting on a position. There are multiple possibilities in the literature. The most popular mathematical formalism is that of expected utility (used for instance in Levy et al., 2000 [44]), but there are problems with this approach, the first being the fact that expectations — through probabilities of future returns — have to be computed at each point in time, resulting in longer genome processing time. One can argue that in a situation of Knightian uncertainty[3], expectations are themselves uncertain since the probability distributions of future returns are not known; the whole idea of calculating expected utility therefore becomes very dubious.

Another possibility is to have each rule bet a constant amount of money once its condition has been identified, as in Ecemis et al. 2005 [19]. Yet another possibility is to

[2]This term occurs in the technical trading literature; an oscillator is any function (computed from data) that defines a trading condition when it has value zero. If the function crosses zero from below, a buy is identified, and if it crosses from above, we have a sell. See Achelis, 2003 [1] for instance. Our definition is a non-standard extension of this concept.

[3]Frank Hyneman Knight (1885-1972) was an American economist who is remembered for making a distinction between risk (a situation where the probability distribution of losses in known) and uncertainty, as the situation in which the distribution is unknown (Knight, 1921 [39], available online for free at: http://www.econlib.org/library/Knight/knRUP.html. The pertinence of this distinction is generally recognized, and situations of uncertainty abound; however, there is little in the way of a consensus about how to model it.

have this amount as another allele in the chromosome. Since we would like to identify the amount the market bets given the identification of a condition, we employ a variant of this scheme, based on the specificity of a rule. Rule specificity is easier to define in the context of a fixed-length rule representation, in the next section.

Fixed-length generalized trading rules

We can obtain a more parsimonious representation for simple trading rules by setting a global $v_{boundary}$ vector for the whole population of rules. An obvious benefit of this is economy of computer memory, but it is also a good idea because it allows us to impose as a precondition on the genetic algorithm what we know of the behaviour of real traders, namely that they are give more weight in their decision process to more recent events. This restricts the search space to regions that we know are more likely to contain reasonable solutions. With $H = 10$, we can for instance use $v_{boundary} = \{1,\ 2,\ 3,\ 5\}$, which emphasizes recent returns over older ones.

This convention allows us to define rule specificity in the following obvious fashion: if the global $v_{boundary}$ has k elements, the specificity S of a fixed-length generalized rule is defined as $9(k+1)$ minus the sum of the $(k+1)$ integers defined from the $v_{conditions}$ and the $v_{threshold}$ sets, where the summands are:

$$t_\alpha, \text{ if condition} = \text{"a"}$$

$$8 - t_\alpha, \text{ if condition} = \text{"b"}$$

$$\max(t_\alpha,\ t_\beta) - \min(t_\alpha,\ t_\beta) + 1,\ \text{if condition} = \text{"c"}$$

$$(\min(t_\alpha,\ t_\beta) - 1) + (8 - \max(t_\alpha,\ t_\beta)),\ \text{if condition} = \text{"d"}$$

This is simply the sum of the "off" octiles for each condition in the rule. Since we have a fixed-length representation, the maximum specificity of a rule is $9(k+1)$ (a rule that can never be satisfied), and the minimum specificity is 0 (a rule that is always satisfied). It seems reasonable to have the trading amount depend proportionally on specificity: for instance, as $\$100\frac{S}{9(k+1)}$. That way, a rule that is hard to satisfy will bet more money when it fires, and a rule that gets satisfied more often will bet less at each time. This setup ensures that populations do not get dominated by sets of looser rules, who provide little forecasting power. Setting the amount of money bet based on specificity also reflects the behaviour of real traders.

From the definition, the sum of the specificities of a rule and its reciprocal is $9(k+1)$. The trading amount of an oscillator can therefore be based on the specificity of the component that gets activated at any given time, or on an average of both component specificities.

Relationship to agent-based models

While it is useful to think of a population of trading rules as encoding the behaviour of trading agents, this population in no way constitutes a complete agent-based model. True agent models incorporate mechanisms responsible for keeping track of past actions and states

and deciding about new courses of action; moreover, financial decision-making involves by its very nature the presence of budget constraints, which we do not take into account. In comparison, a population of rules is mechanistic and memoryless. We are indeed not so much interested in achieving realism in agent behaviour as to determine how the market *reacts to itself endogenously.*

For instance, suppose an opportunity investment presents itself in the market and is recognized independently by many traders. This may be due to a statistically significant drop in price during the last five periods, followed by a sharp rise in the previous day. We can expect that some investors will send buy orders to the market, even if some of them are broke because of previous bad investments. The verisimilitude of the agent's investment histories need not concern us as its identification is not our objective. Our objective it to capturing the extent to which past price patterns may generate future price patterns. Budget constraints, in this optic, would be a hindrance to the recognition of this temporal pattern dependency, since they would condition the actuation of a given pattern upon not only on the pattern history that preceded it, but on other variables, and render it harder to recognize.

4.1.2 Calibrating an empirical market-maker

We now describe how the market-maker updates its quotes as it receives buy and sell orders from the population of rules, keeping in mind the desiderata of section 3.3. Our approach is based on empirical distributions.

Market-makers try to keep inventories low; since inventories are stochastic, we can also compute their empirical distribution on specific samples of prices with specific rule

populations. Using our data sample and a given population of rules, we can simulate trading without any active market-making. Depending on price patterns, rules will post buy and sell orders, generating inventory values. This can be described as a variant of the forward market simulation of figure 3.3, a process we call "empirical market-maker calibration":

- a historical price time-series of at least 100 observations is selected.
- the market considers the H first prices from the time-series as the current price stack.
- all the "IF" conditions are evaluated on the H prices in the stack, causing a subset of rules from the population to fire.
- firing rules create *orders* that are put in the book and immediately submitted to the market-maker. These orders constitute a market demand for the current moment in time.
- the market-maker fills the orders at the current price and computes his new stock inventory.
- the value of the stock inventory of the day is saved. Initially, the inventory is zero.
- the time-counter is increased; the market-maker DOES NOT push a new price on the stack, but the next historical price is pushed onto the stack.

Figure 4.2 summarizes these steps.

Since the state of the inventory is recorded at the end of each day, we end up with a sample of n inventory observations $y_1, y_2, y_3, \ldots, y_n$. As we did for continuous price

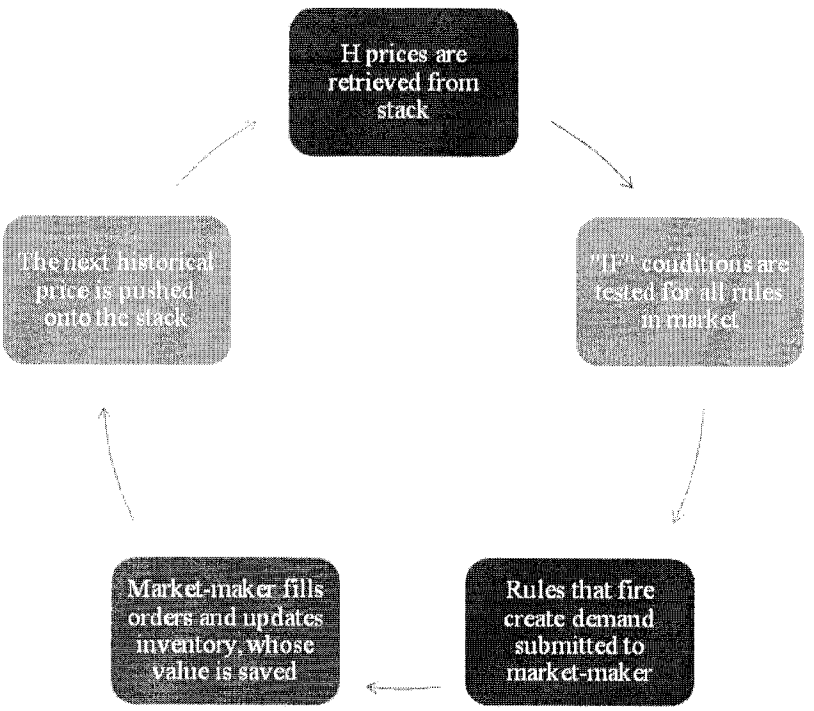


Figure 4.2: Empirical market-maker calibration

returns, we define the empirical distribution of inventory values:

$$\widehat{G_n[y]} = \frac{1}{n} \cdot \#\{i;\, 1 \leq i \leq n;\, y_i \leq y\}$$

Recall that market-makers having a large inventory must attempt to lower it by selling stock to the public; however, the public can be coaxed to sell only if prices get lower. Extreme positive levels of inventory will therefore cause an extreme negative price movement; conversely, extreme negative levels of inventory will cause an extreme positive price movement. We can therefore infer a (stylized) behavioural rule for the market-maker (calibrated to the past market prices and to the given rule population) by matching the two empirical distributions octile to octile, keeping in mind that as inventory octiles go from 1 to 8, corresponding price octiles go from 8 to 1, reflecting the behavioural rule of the market-makers. For any probability $p \in (0,1)$, the percentile to percentile matching condition can be written as:

$$1 - \widehat{G_n[y]} = \widehat{F_n[x]},$$

and a behavioural response function that maps inventory levels to price changes expressed as a continuously compounded rate x is given by:

$$x = F^{-1}\left\{\widehat{1 - G_n[y]}\right\}$$

At this stage, we can substitute the empirical distribution with a parameterized form, such as a logistic distribution:

$$F(\alpha, \beta) = \frac{1}{1 + e^{(-\frac{x-\alpha}{\beta})}}$$

There are many ways to do so; we can optimize the parameters α and β by using steepest descent while minimizing the Kolmogorov-Smirnov statistic, given by:

$$D = \max_i D_i = \max_i \left[\left| F(\alpha, \beta) - \widehat{F_n[x]} \right| \right]$$

However, running an optimization algorithm is too costly to be performed at each genome evaluation in a genetic algorithm; instead, we linearly interpolate the response function from octile to octile. This constitutes the price-adjustment mechanism of the empirical market maker, and this mechanism satisfies the *desiderata* we stated in section 3.3.

An interesting fact about our calibrating procedure is that genomes depend on the nature of the market-maker to affect prices via forward simulation. Not unlike actual biological genes, our trading rules can be traced to a position in the genome *in abstracto*, but they do not exist independently when the genome is actuated in a real environment. Depending on the environment (in this case the past price history), this or that different gene may or may not be expressed (or responded to by the price-adjustment mechanism) as much as another. This is an important aspect of our method, as no predetermined gene hierarchy is hard-coded in the genome. Gene dominance is a purely emergent phenomenon.

4.2 The genetic algorithm

4.2.1 Training data

We want to find what is the best population of trading rules that will generate market prices that are closest to a given market price history. Once a population of rules is evolved and generates prices that are close to a given historical interval on a training sample, we consider this population as a reconstruction of the underlying dynamic of the market and we forward simulate the market out of sample, generating a forecast.

We divide a historical sample of data into two intervals:

1. the pre-processing interval, used to construct the empirical distributions $\widehat{F_n[x]}$ and $\widehat{G_n[y]}$. $\widehat{F_n[x]}$ is pre-computed and stored, whereas $\widehat{G_n[y]}$ is computed at each genome evaluation. We recommend a pre-processing sample of at least 100 observations for statistical significance.

2. the training interval, composed of the data that comes next to the pre-processing sample, used by the inner GA to evaluate the fitness of a given genome. The length of this interval can be selected at will. The trade-off is that the larger the interval, the more lengthy the resulting genome, and the more processing time. In the experiments described in the next chapter, a training sample length of 50 weekly observations has been used, for forecasts of up to one year. Judging from the amount of time it takes to evolve on a standard machine, it would be quite impractical to attempt to train on much longer intervals.

When a best of generation individual has been found, it can be used to simulate a

market run beyond the training interval, and these new generated prices can be used as a forecast. If the model is used in a business setting to generate forecasts, the training interval ends at the present day and a forecast can be made for the desired number of periods. We call *forecast interval* the interval immediately following the training sample. This interval contains historical data that the model has never seen during the evolution process, and is used to evaluate the out-of-sample forecast performance of the model in the next chapter.

Forecast accuracy is expected to get worse, the greater the forecast length; forecasts are best evaluated over a length that is commensurate with the training interval: it would indeed be surprising that a model fit over a small sample be robust enough to generate reliable forecasts over a much longer period.

This data setup is relatively standard; there are other more sophisticated ways of selecting training data in the literature; for instance, Wagner and Michalewicz 2007 [56] suggest using a sliding training window procedure to minimize overfitting and maximize forecast accuracy: this is the basis of "Dynamic Forecast" (DyFor) GP. Notwithstanding the merits of their scheme, it is quite impracticable in the present context because of the computational burden it would cause. This will become apparent as we discuss the evolution loop and the background steps that must be taken before the objective function can be evaluated. That is not to say that the risk of data-mining is neglected in the current model; to the contrary, overfitting is mitigated through an explicit parsimony parameter in the objective function; it can also be mitigated by selecting a larger training window.

4.2.2 Genome structure

Since our rules are composed of tuples, they can be easily expressed as a variable-length or a fixed length representation. In practice, variable-length rules are implemented as tuples having a specific maximum length, which keeps memory usage under reasonable bounds. Rules are collected into a variable-length set that constitutes a genome, the unity of selection. The data structure used for a genome is a two-dimensional array of alleles, composed of symbols taken from the set $\{1,\ 2,\ \ldots,\ 8\}$. This is similar to the genomes that we considered for the Mastermind game of section 2.1. We do not need the symbols $\{B,\ S\}$; we interpret the second allele as meaning a "B" if the allele is between 1 and 4, and a "S" otherwise.

To implement a variable number of rules per genome within a data structure of fixed dimensions, we interpret the very first allele of every line as an "on/off" expression allele, subject to mutation like all other alleles. Genomes therefore carry junk genetic baggage that play no role in the current generation, but may be reinterpreted and play an useful role in future generations if the expression allele flips due to a mutation.

It is important to underscore that the units of selection are populations of rules, and not individual rules. Selecting the best individual rules by means of evolution is not a feasible strategy because of the nonlinear interactions of rules within a population. The contribution of a single rule within a population cannot be isolated, and depends on the contribution of other rules in unpredictable ways: the end result is that no fitness can be assigned to a given rule taken in isolation from others. However, we can easily assign a fitness function to a population of rules.

We have already mentioned the important parameter H that we call "rule look-back", which controls how far in the past the rules of a genome can look to take their decision. This should intuitively set to match the number of lags that are significant in the data correlogram and autocorrelogram; we will return on this topic in chapter 5, while discussing ARIMA models. Rule lookback should not be set too high since it yields a model that becomes very costly in terms of processing time, as well as making convergence much longer to achieve.

4.2.3 Evolution operators and parent selection

We evolve populations of genomes (which are themselves populations) within an overlapping (steady-state) genetic algorithm. Each genome is a two-dimensional array of dimensions $N \times M$. N represents the maximum number of rules in a genome and M is the maximum rule representation length, which is simply defined as $4H+5$ to allow for sufficient space to represent even the longest rules. Parents are selected by 3-way tournaments and crossover is one-point. Mutation is uniform. Table 4.1 summarizes the genetic parameter values of a typical run on a training sample size of 50 price observations. We will describe in the next chapter runs with different parameters.

We use a simple static parameter strategy combined to a grid search over the space of model-specific parameters.

4.2.4 Fitness evaluation and objective function

The objective function is bipartite; it is calculated for each genome as a weighted sum of two criteria:

Genetic parameter	value
elitism proportion	40%
population size	200 (constant)
maximum number of rules (N)	70
crossover type	one-point
mutation type	uniform, $p = 10\%$
parent selection	3-way tournament
termination criterion	number of generations (500000)

Table 4.1: Genetic algorithm parameters for a typical run

1. a statistical closeness criteria that compares the price path generated by a genome to the prices in the training sample, namely the sum of squared errors between the price forecast given by market forward simulation and the actual price training sample.

2. a parsimony criterion, the effective genome length, defined as the number of "on" rules in the current genome. Recall that this varies from one individual to the other, as a specific allele on every line defines whether the corresponding rule is active or not.

   Formally, the objective function of a genome is:

$$Obj(genome) = w(SSE) + (1 - w)(len(genome))$$

   , where the parameter $w \in [0, 1]$ is the relative importance of the error minimization part of the objective function. From this definition, it is plain that the higher the objective function, the lower the fitness of an individual.

The objective function of an individual genome is found with the following steps:

1. the empirical market-maker calibration process (figure 4.2) is performed for the market defined by the genome, using the pre-processing interval.

2. the market defined by the genome is forward simulated (figure 3.3) to generate as many prices as there are in the training interval. The initial price conditions for this forward simulation are the H last prices from the pre-processing interval.

3. These forward simulated prices are compared to the actual training data to compute the sum of squared errors over the training interval. Denote the actual prices as $a_1, a_2, a_3, \ldots, a_n$, and the simulated prices as $s_1, s_2, s_3, \ldots, s_n$, we have:

$$SSE = \sum_{i}^{n} (s_i - a_i)^2$$

4. the length of the genome is calculated. At this point, we have all the required elements to compute the objective function of the individual.

Evaluating the fitness of an individual is a costly process in terms of processing time; this justifies the usage of a steady-state GA combined with a tournament, which minimizes this time.

Chapter 5

Model evaluation

5.1 Software implementation

The genetic algorithm used in this work were written on top of the GAlib open source C++ genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology[1]. Instances of the model were run in parallel on the Engineering and Computer Science (ENCS) cluster, a machine composed of 712 2.2 GHz AMD Opteron 64-bit processor cores, utilizing the InfiniBand network as the means for node-to-node communication and for I/O to the cluster file system. If they had been run on an individual 2.2 GHz processor, the various experiments described in this chapter would have taken a computation time in excess of four years of continuous use. Each model evaluation (with the parameters of table 4.1) takes about 9 hours of computing time for each 2.2 GHz processor.

[1]This library is available at:
http://lancet.mit.edu/

5.2 Experimental data selection

It would be a waste of time to apply the model on time-series that exhibit a clear trend; at the cost of much computation, the end result would be a forecast that would be very similar to that obtained at very little cost by fitting a linear trend line to the data. More importantly, testing a nonlinear model on linearly trending data would tend to over-estimate the true performance of the nonlinear model: it is indeed an easy problem to infer a linear tendency in data. For instance, the MA 26/12 rule (and most other technical rules) that we discussed does very well on data with trends. If the underlying market dynamic is truly reconstructed by our genome, then this fact should only be tested in situations where the problem is hard, since it is going to be useful in these situations only.

So the first question is to identify which data sets do not exhibit a clear trend. This is the notion of time-series *stationarity*. Formally, a stochastic process y_t (indexed by time t) is weakly stationary (or covariance-stationary) (see Greene, 2003 [27]) if:

1. $E[y_t]$ is independent of t.

2. $Var[y_t]$ is independent of t, and

3. $Cov[y_t,\ y_s]$ is a finite function of $|t-s|$, but not of only t or s.

Nonstationarity processes exhibit "explosive" behaviour, which usually requires prior differencing before analysis can proceed. There are many statistical procedures that allow detecting various kinds of nonstationarity in time-series. One such general tests is the portmanteau statistic of Brock, Dechert and Scheinkman (BDS) 1991 [11], which is commonly used to test non-linearities (and nonstationarity) for chaotic processes in physics.

In finance and economics, researchers usually test whether the data follows a random walk process, which is a very common kind of nonstationarity. Suppose for instance a time-series of prices described by the following process:

$$y_t = \alpha y_{t-1} + \varepsilon_t$$

where ε_t is a Gaussian white noise (a sequence of independent draws from the normal distribution with constant variance). Subtracting y_{t-1} from every side, one gets:

$$\Delta y_t = (\alpha - 1) y_{t-1} + \varepsilon_t$$

, which suggests that if Δy_t were regressed on y_{t-1}, the t-statistic on the slope coefficient could be used to test whether $\alpha = 1$, which would mean that the changes in prices Δy_t are just white noise, hence impossible to forecast. If $\alpha \neq 1$, then changes in prices can be explained (to some extent) by the recent past prices, so a forecast is possible. This testing procedure is known as a *unit root test* for reasons that will not be discussed here. While multiple tests exist in the literature with various robustness and power characteristics (Hamilton 1994 [28] is a comprehensive resource on this topic), we have chosen to use the Dickey and Fuller test (Dickey and Fuller, 1981 [17]), one of the most popular. Dickey and Fuller have shown that a true t-statistic does not obtain under the null and have tabulated the correct critical values. We have used the augmented version of the test, which has greater power to identify random walks with the presence of autocorrelation (for instance, cf. Kennedy, 1998 [38]). Taking the components of the S&P500 index over the 2004-2006 period, we have found 276 of them to be stationary according to the Dickey-Fuller test.

200 stocks from that sample were used generally in our experiments[2]. Figure 5.1 graphs a subset of 50 stocks.

The first panel of figure 5.1 shows the price time-series of the first 50 selected stocks of our sample of 200. It appears that the Dickey-Fuller test was successful at removing stocks with a clear tendency. Panel 2 of the same figure normalizes all stocks to have a value of 100 at the beginning of the forecast period. The symmetry on both sides confirms that this group of stocks exhibits no upward or downward tendency. Since markets are generally up trending, if we had kept all 500 stocks, panel 2 would have shown a generally positive slope overall.

5.3 Parameter optimization

Any evolutionary computation model has a wealth of parameters; as shown in table 4.1, we have chosen most general GA parameters *a priori* (with some degree of experimentation in the course of developing the software). This is justified by the literature, as it has been shown by trials on test suites that GAs are quite robust with respect to their parameter setting (see for instance Goldberg 2002 [26] and Rechenberg 1965[52]).

However, it is apparent from our discussion that the value of parameter w in the

[2]The ticker symbols of these stocks are ASH, AIZ, ADSK, ADP, AN, AZO, AVY, AVP, BHI, BLL, BK, BCR, BRL, BAX, BBT, BBBY, BBY, BIIB, BJS, BDK, HRB, BMY, BRCM, BNI, CHRW, CA, CAM, COF, CAH, CCL, CAT, CBG ,CNP, CTX, CHK, CVX, CIEN, CI, CTAS, CC, CSCO, CIT, C, CTXS, CCU, CLX, COH, CCE, CMA, CBH, CSC, CPWR, CAG, COP, CNX, ED, STZ, CEG, CBE, GLW, COST, CFC, CVH, DHI, DRI, DE, DVN, DDS, D, RRD, DOW, DTE, DD, ETFC, EMN, EK, ETN, EBAY, EIX, EP, ERTS, EDS, EMC, ESV, EOG, EFX, EL, ESRX, FDO, FNM, FRE, FII, FDX, FIS, FITB, FHN, FRX, FO, GPS, GE, GM, GPC, GENZ, GR, GT, GOOG, GWW, HAL, HOG, HAR, HAS, HNZ, HPC, HES, HD, HON, HSP, HBAN, IACI, ITW, RX, IR, TEG, INTC, IBM, IFF, IP, IPG, ITT, JBL, JEC, JNS, JDSU, JNJ, JCI, JNY, KBH, KMB, KG, KLAC, LLL, LH, LM, LEG, LEH, LEN, LUK, LXK, LLY, LIZ, LOW, LSI, MMC, MAS, MAT, MBI, MKC, MHP, MCK, MWV, MHS, MDT, MDP, MTG, MCHP, MU, MSFT, MIL, MOLX, TAP, MNST, MUR, MYL, NBR, NCC, NOV, NSM, NTAP, NEM, NKE, NI, NE, NBL, NSC, NOVL, NVLS, NYX, PLL, PH, PAYX, BTU, PKI, PFE, PNW, PBI, PCL, PPG, PGR, PRU and PHM.

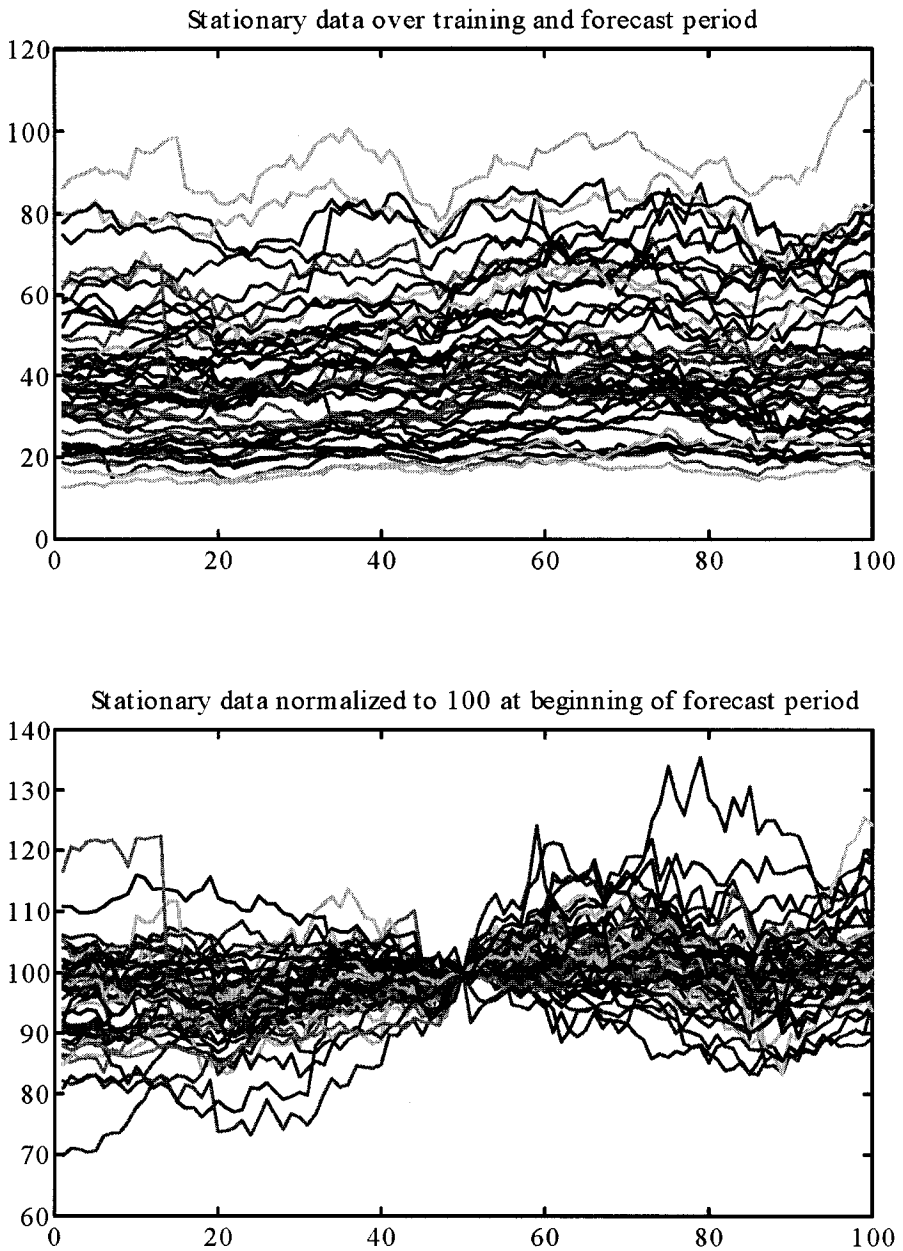


Figure 5.1: Sample of 50 weekly stock prices from 200 S&P500 stationary stocks, 2004–2007 over training and forecast intervals

objective function (the relative importance of the SSE, which is model-specific) has an important impact on the nature of the solution that is found at the end of the evolution process: no *a priori* choice is obvious in this case. Another problematic model-specific parameter is the rule lookback H, which chooses how far back in the past rules look to make their decisions, and represents the maximum memory length of traders. Ideally, all GA and model parameters should be selected as to maximize forecast performance of the forward simulated price on the forecast interval, but in practice this is too computationally expensive to do. We performed instead a grid search over a range of plausible values for these two problematic parameters. Figure 5.2 shows the average SSE of forecast errors for 20 stationary stocks[3] over the forecast interval of 50 weekly observations of 2006, produced by allocating the value of the lookback parameter H to the range $\{3,\ 4, \ldots,\ 9\}$ and of the SSE weight parameter w to the range $\{0.91,\ 0.92, \ldots,\ 0.99\}$, all the other parameters being set to the values shown in table 4.1. To reduce processing time, we have implemented the fixed-length generalized trading rules version of the model. Producing this graph alone required 1400 model evaluations, which would have taken on a serial 2.2 GHz processor about 18 months of continuous processing time.

The SSE of forecasts is minimized by selecting $H = 6$ (each rule looks back 6 periods in the past) and $w = 93\%$ (weight SSE criterion in the objective function as 93% of the total). The graph has the shape of a bowl, but is quite noisy, a fact that is not surprising considering the stochastic nature of financial prices. As a matter of fact, the point we have selected is not the lowest on figure 5.2: $H = 9$ and $w = 92\%$ is. However, since this better

[3]These stocks are MMM, ABT, ADBE, AMD, AET, ACS, AFL, A, AA, ALTR, AMZN, AIG, AMGN, APC, ADI, BUD, APA, AAPL, AMAT and ADM.

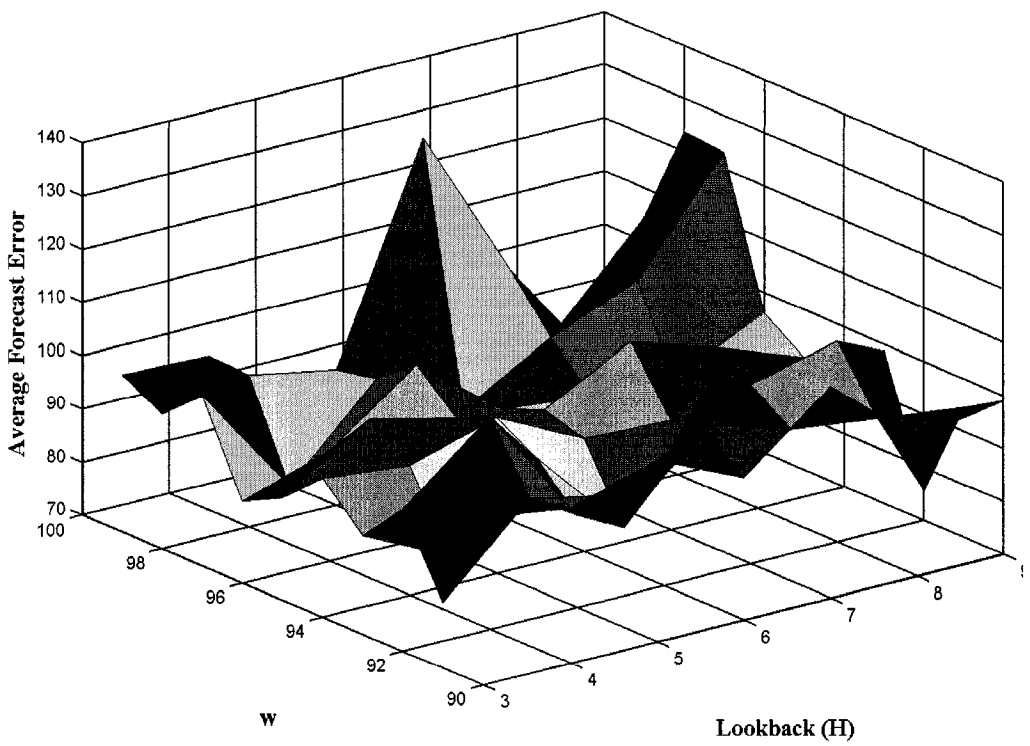


Figure 5.2: Average SSE of forecast errors for 20 stationary stocks, for parameters w and H.

point is on a corner, and that it beats the selected point by only a small margin, it was rejected. Moreover, model evaluations for $H = 9$ are twice as expensive in terms of time (19 hours) as for $H = 6$ (9 hours). An important detail is that the 20 stocks selected for making this graph were chosen at random from the stationary sample, and were not re-used in any further experiment, to eliminate data-snooping bias.

Optimizing parameter values from a fitness landscape is not the only way to proceed; another approach to this problem is to set up a nested evolution strategy, the top-level GA evolving the parameters of the lower-level GA. De Jong 2007 [33] cogently remarks that this leaves open the question as to which parameters are to be selected for the top-level GA, opening the door to an infinite regression. Nested evolution therefore does not solve the problem of parameter selection, but transfers it at a meta-level, where parameter choice is even less intuitive. Apart from the computational cost of a nested scheme, we feel that due to the large part of noise present in financial data, the parameter's "sweet spot" is probably going to be quite large. Figure 5.2 somehow warrants this view, at least for the 20 stocks we have selected. However, it is by no means certain that this fitness landscape would stay stable under more stocks over a different period.

In what follows, we will apply the fixed-length generalized trading rule version of the model for different experiments using the optimized parameters we just found. We first discuss reasons why we should not be too optimistic about the results of these experiments.

5.4 Theoretical difficulties of the model

There are clear theoretical difficulties with establishing forecasts using any deterministic model. Let's mention a few at the outset:

- *Stochastic nature of markets*: "real" stock markets are arguably probabilistic, entirely or at least partly. By "partly probabilistic", we mean that there sould be a signal-to-noise ratio in stock prices. Stocks are indeed affected by random events happening in the world, which are of course unpredictable unless one is intimately acquainted with the Laplacian deamon. The importance of this random element in a given market causes a limitation on the possibility of forecast by any method.

- *History dependence*: parameters must be derived from historical data, but there is no guarantee that these parameters are going to remain stable in the near future. For instance, in September 2008, in a context of plummeting financials, the Securities and Exchange Commission decided to forbid short selling on stocks whose short open position was beyond 10% of total market capitalization; the effect of this decision could clearly be seen in market prices. It created a distinct market "regime"[4] after the decision. Calibrating an artificial market on a sample of prices prior to such an event would be a woefully inadequate way of predicting what would happen after the cut. Of course, dramatic events like this do not happen every day, but one could easily believe that minor events would also change the price regime. History-based methods are therefore useless for predicting something that is out of the ordinary.

[4]The term "regime" is intentionally vague; it is meant by that a characteristic of the time-series of prices that is visible to the eye.

- *Human behaviour*: people can choose to act differently from how they acted in the past based on their changing expectations. Since markets are composed of people, we would ideally need a theory of how people change their mind, based in part on their own individual reasons and based in part on the influence of others. Herding phenomena, fads and panics are very hard to model, and even harder to forecast.

Obstacles more specific to our model related to its parameterization procedure include:

- *Forecast stability*: market inversion is based on a genetic algorithm, which must achieve convergence before reaching an answer. It is quite easy to measure convergence: for instance, one may measure it as the ratio of the Nth previous best-of-generation average objective function score to the current best-of-generation score. Alternately, one may look at population diversity, which must lower gradually and become suitably low at the end. Population diversity may be defined as the variance of objective function scores of the whole population (which is costly to compute and not an option in the context of our model), or based on indirect measures such as average genome entropy, which does not require objective function evaluation. The solution we retained here was to setup a preliminary experiment during which we followed population diversity for a very large number of generations (2 million). On the basis of that experiment, we selected a threshold number of generations for which population diversity was achieved since a long time. This threshold number is 500 000 generations; usually reasonable convergence is achieved in the range 100 000–200 000 generations. But despite these cautions, we can never unambiguously decide

that convergence is truly achieved in all cases with 500 000 generations. Therefore, a given forecast based on a best-of-generation individual may be inherently unstable and would differ, had the genetic algorithm been allowed to evolve for more generations. We can however test for forecast stability.

- *Non-unicity of forecasts*: basing forecasts on best-of-generation individuals is the obvious way to proceed; however, depending on the length of the training period, it is entirely possible that many different genomes could be found that achieve equal fitness. Indeed, genomes of different sizes may be applied to the same problem, and it is inevitable that the larger genomes may fit the data as well as smaller genomes but also carry unexpressed genetic material that may change the nature of their forecast. We may also imagine that different genomes of equal length that fit equally well may not generate the same forecasts. The question would therefore be of which of these "best" chromosomes should be selected for a forecast. Taking together all forecasts from these chromosomes in the attempt of getting a sort of statistical future price distribution might be tempting, but we should resist this temptation. There is indeed no justification in using an underspecified deterministic model to bootstrap a probability distribution. A way out of this problem could be to fit the model on data sets that are progressively bigger until a unique best individual would emerge for a fixed genome size, which would be very computationally expensive. Or inversely to fit the most parsimonious model for any given data set, which is the approach we have implicitly used by rewarding parsimony in the objective function.

- *Informational bottleneck*: market prices are generated from the complex aggregate be-

haviour of multiple rules; however, prices represent a single *trace* produced from that complex system. Not unlike the problem of inferring brain activity from ECG's, it may well be that the informational bottleneck price time-series represent causes a loss of too much information for allowing one to reconstruct unequivocally its underlying dynamical system. To test for this, we could select a randomly generated market chromosome, use it to generate a price history, and apply the method to reverse-engineer that original chromosome. We could therefore see whether that the informational bottleneck prevents us from reconstructing it.

We must therefore admit at the outset that there is a strong likelihood that our method may not yield a single generating chromosome, but a whole set of different generating chromosomes. The shorter the original time-series to be fitted, the more severe this non-uniqueness will be. This raises the question of the practical value of the inversion method: what if the reconstructed dynamics of the artificial market is simply a case of overfitting, having no predictive power out-of-sample? At first view, it would appear that market forecasting would be the best way of evaluating the value of the solutions obtained: low forecasting power would mean that our method of calibration would be defeated by the non-uniqueness problem, and high forecasting power would be evidence of calibration success. But it may also be the case that forecasting quality and reconstruction quality may not be so strongly coupled.

5.5 Forecast quality evaluation

Our GA model, apart from its theoretical interest, ultimately results in a forecast; from a practical point of view, one may want to know whether it improves upon other standard methods. Since much capital is at stake in forecasting financial markets, many attempts have been made, using varied angles of attack. For instance, studies have proposed using attractor reconstruction from chaos theory, or Fourier analysis, wavelet analysis, Hilbert transforms, artificial neural networks, genetic programming, kernels, evolutionary strategies and many other approaches, with various degrees of reported success. There is even a mythical folklore of forecasting; for instance, several popular books report that chaos theory has been applied to financial forecasting with great success, but of course the precise way to do so remains secret (for instance, Bass 2000 [7]).

It would be pointless to attempt to compare our method with any of these models, serious or not, since in many cases, they are quite complex (or can take a huge variety of forms, as for neural networks), sometimes ill-described and often proprietary. Complexity of implementation casts a doubt upon comparative studies, as one may doubt whether the modelling of competing models has been done according to the art. There is also a strong incentive to publish only the unsuccessful models in the literature and keep the successful ones under wraps for private use: this casts further doubt upon comparison attempts.

So in order to evaluate the forecasting quality of our model, we compare it with forecasting methods that must:

- be widely known and recognized by market practitioners, academics and statisticians as an accepted way to forecast stocks.

- be correctly implemented without any doubt. This is achieved by using not our own implementation, but code developed by recognized specialists.

We now discuss the two models that we use as a basis for comparison.

5.5.1 The lognormal random walk model

We have already described this model in the introductory chapter; the expected value of the stock at time T is $S_0 \cdot e^{\mu T}$. The parameter μ can be estimated by linear regression, which involves no technical difficulty, but is known to give an estimate with unreasonably high variance. Essentially, the model is agnostic with respect to the future, and is expected to compare badly to any other forecasting method. It is nevertheless very mainstream: distinguished theories such as the Black-Sholes-Merton option pricing model assume the lognormal model as the true stochastic process of stock prices. Because of this central role to financial economics, we have to include it here. It should also be said that many technical rules base their decisions following this linear forecast (or deviations thereof). Since the prediction of the lognormal model is based on a linear trend, its performance will vary according to whether such a trend is actually present in data, which is frequently true. But as we were mentioning, there is more interest to models that can inform us of nonlinear trends.

Implementing a linear regression is easy; however, to be consistent with our implementation standard, we used the R[5] library called "sde"[6] to estimate the lognormal parameter μ. This library was developed by Stefano Iacus and is described in Iacus 2008

[5]The R environment is the *de facto* standard open-source statistical modelling environment. Details can be found at www.r-project.org.

[6]available at www.r-project.org.

[31].

5.5.2 The ARIMA model

Statisticians have studied linear models since the 18th century, and the properties of optimal linear forecasts are well-understood. In 1970, Box and Jenkins [9] described an algorithmic approach to forecasting time-series using the standard linear statistical model, dubbed "ARIMA", which stands for "autoregressive integrated moving average". According to Kennedy 1998 [38], the model was initially dismissed by economists on the grounds that it was intrinsically shallow, founded on data alone, and not on any economic explanatory variable. However, as comparative forecast studies started to get published in the 1970's, the method was found to outperform all economic models, to the point that:

> "Some even claimed that the economic approach would be wiped off the map whenever it had to compete against forecasts generated from multivariate Box-Jenkins models" (Kennedy 1998 [38], p. 265).

The Box-Jenkins procedure (Box and Jenkins [9]) can be summarized with the following steps:

- the data is transformed by differencing until it is stationary. This is what we have already done by excluding nonstationary series from our sample, for reasons unrelated to Box-Jenkins.

- denoting Y^* as the stationary series, the model to estimate is:

$$Y_t^* = \phi_1 Y_{t-1}^* + \phi_2 Y_{t-2}^* + \ldots + \phi_p Y_{t-p}^* + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q}$$

, where the θ_i and ϕ_i are the parameters to be estimated (along with the correct lags

p and q), and ε_i is Gaussian white noise. Any time-series is therefore explained solely by its previous values.

The parameters are estimated by doing:

1. *model identification*: the lags p and q are identified from the sample autocorrelogram and partial autocorrelogram.

2. *estimation*: using a recursive/iterative method to approximate the maximum likelihood estimator

3. *diagnostic checking*: the model residuals are tested for presence of either serial correlation or autocorrelation. If there is not any, the specification is correct. If not, the procedure is repeated.

Denoting $cov(Y_t^*, Y_{t-s}^*)$ by γ_s, the autocorrelation function using the standard sample estimators is defined as:

$$\widehat{\rho_s} = \widehat{corr(Y_t^*, Y_{t-s}^*)} = \frac{\widehat{\gamma_s}}{\widehat{\gamma_0}}$$

The sample autocorrelogram is the plot of $\widehat{\gamma_s}$ against s. The partial autocorrelogram is the graph obtained by plotting the corresponding partial sample autocorrelations. The partial autocorrelation at lag k is defined as the regression coefficient on Y_{t-k}^* when Y_t^* is regressed on all its lagged values Y_{t-1}^*, $Y_{t-2}^*, \ldots$ Y_{t-k}^*. The reason it is called "partial" is because regressing on all the intermediate lags removes the effect of their correlations. If the pattern of autocorrelation is one that can be captured by an autoregression of order less than k, then the partial autocorrelation at lag will be close to zero.

The lags are determined by comparing the size of these autocorrelations with the Box-Pierce-Ljung statistic (Box *et al.*, 1970[10], Ljung *et al.*, 1978 [46]) and retaining only the first p and q significant lags in the final specification.

ARIMA estimation is available in most statistical packages. To guarantee implementation quality, all the ARIMA estimations we report were conducted using the System Identification Toolbox^{TM} of Matlab^{TM}, a highly reputable commercial product[7].

Admittedly, the random walk model is an easy benchmark to beat. The ARIMA model is, to the contrary, a very tough one. In fact, our genetic algorithm model should likely be beaten by ARIMA, which is the optimal linear forecast.

5.5.3 Comparison to benchmarks

Since we produced forecasts on more than 200 stocks, we can only show a few graphs to give the reader an intuitive feeling of how the three methods compare. Figures 5.3–5.6 each display the model forecasts for three randomly selected stocks. On these figures, the blue line represents the 2007 historical weekly data that none of the models have seen in their estimation procedure. Each model used only the 2006 data as a training period. Our model ("GA") was evaluated with 500 000 generations.

Turning our attention first to figure 5.3, we can see that in the case of ASH, the forecasts from ARIMA and the GA are very comparable. However, the GA exhibits a trending behaviour for EP and JCI. ARIMA seems excellent overall, except for MKC (figure 5.4). We often see the GA model either trending (as for DOW and TTE), either exhibiting

[7]the toolbox is available at http://www.mathworks.com/products/sysid/
. The toolbox main architect is Prof. Lennart Ljung.

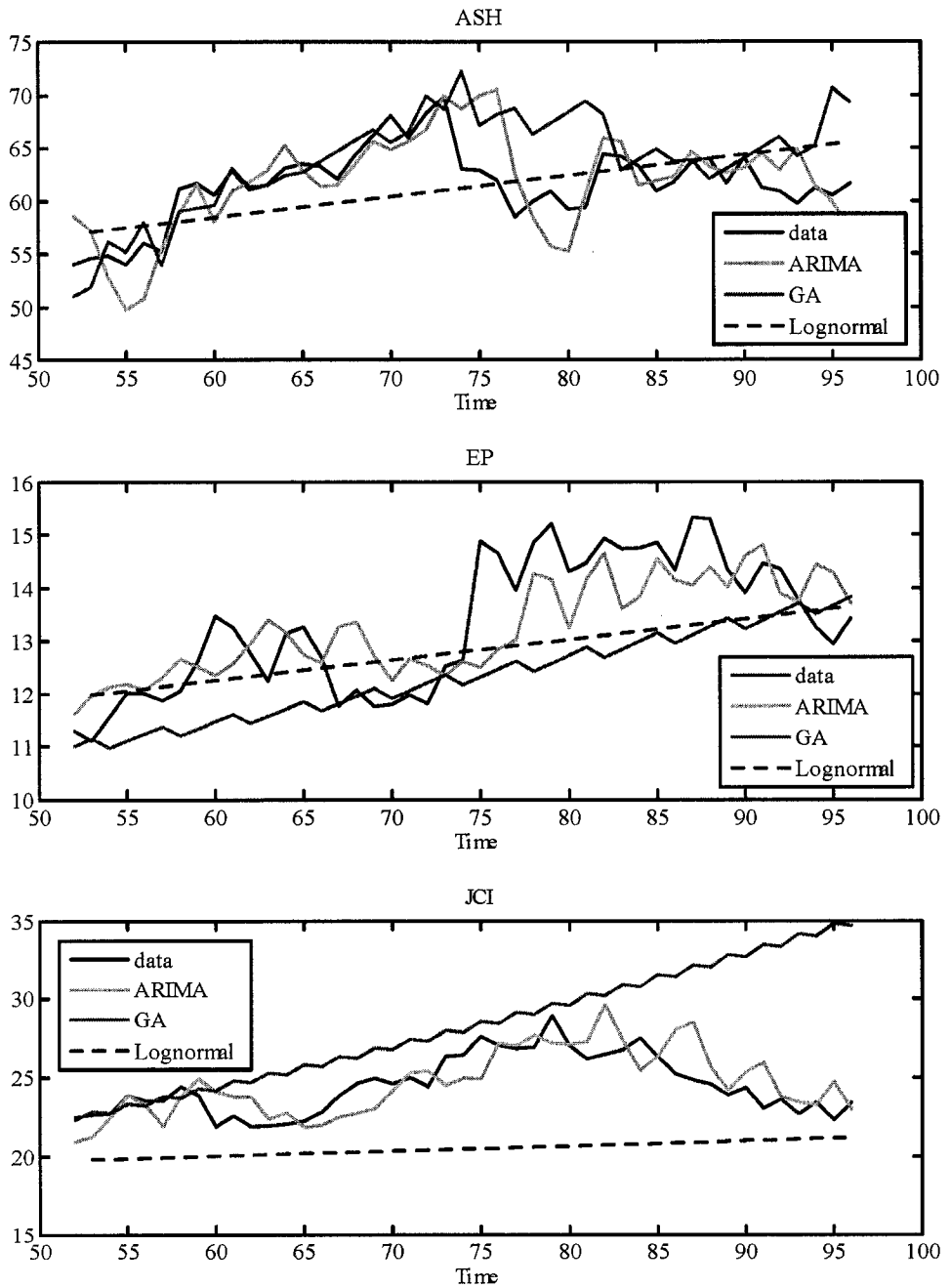


Figure 5.3: Forecast comparison (2007) for ASH, EP and JCI

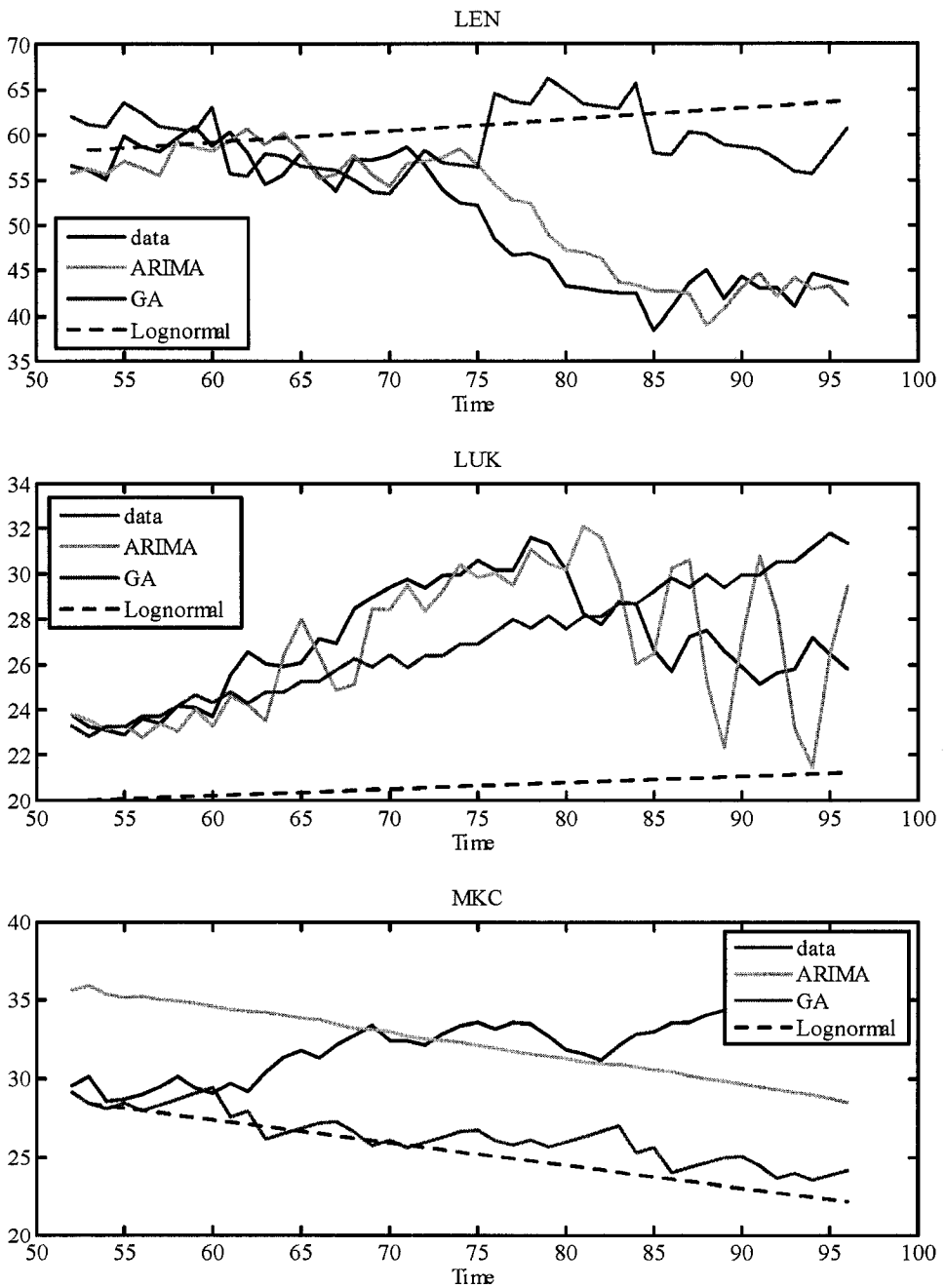


Figure 5.4: Forecast comparison (2007) for LEN, LUK and MKC

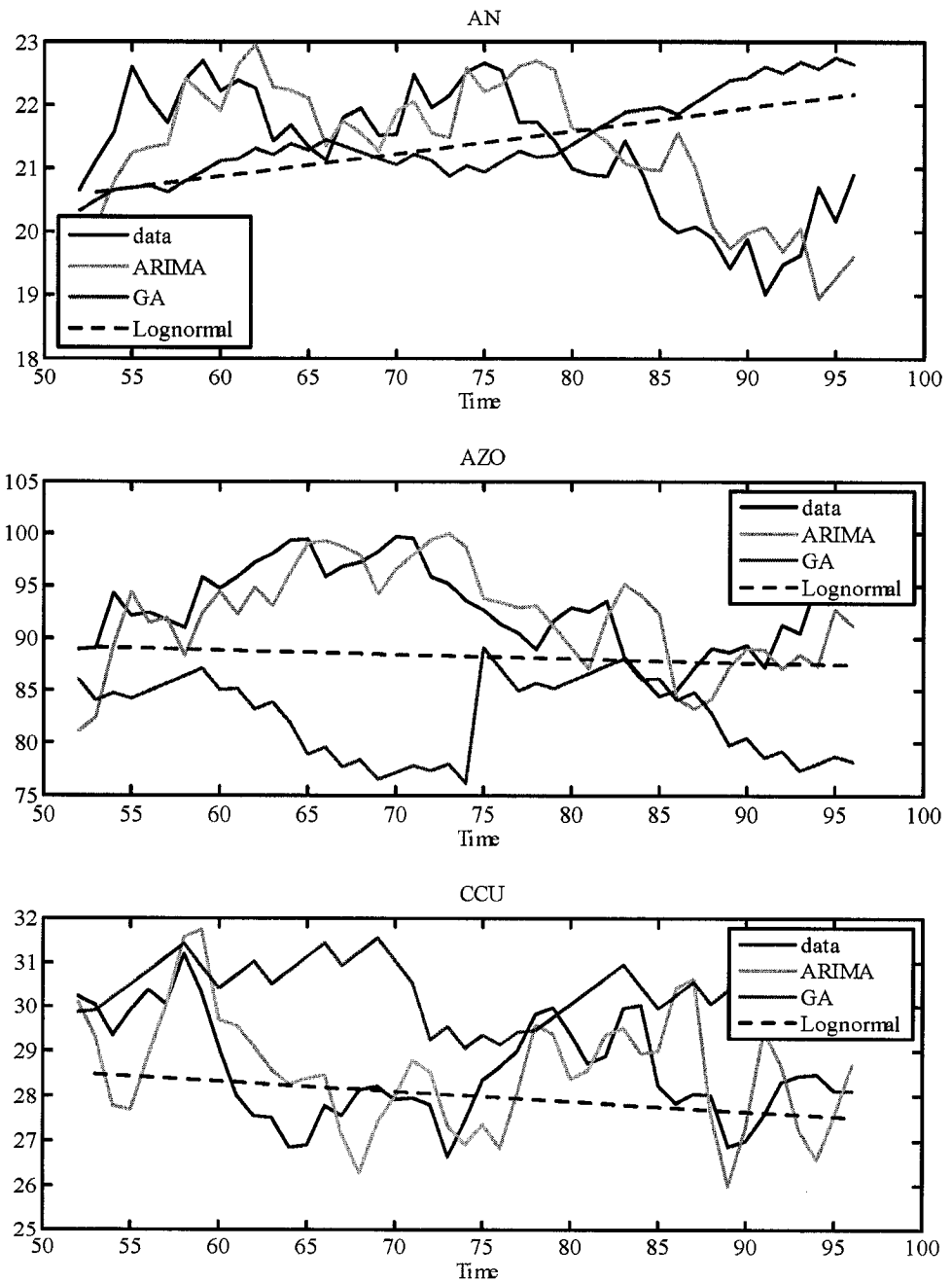


Figure 5.5: Forecast comparison (2007) for AN, AZO and CCU

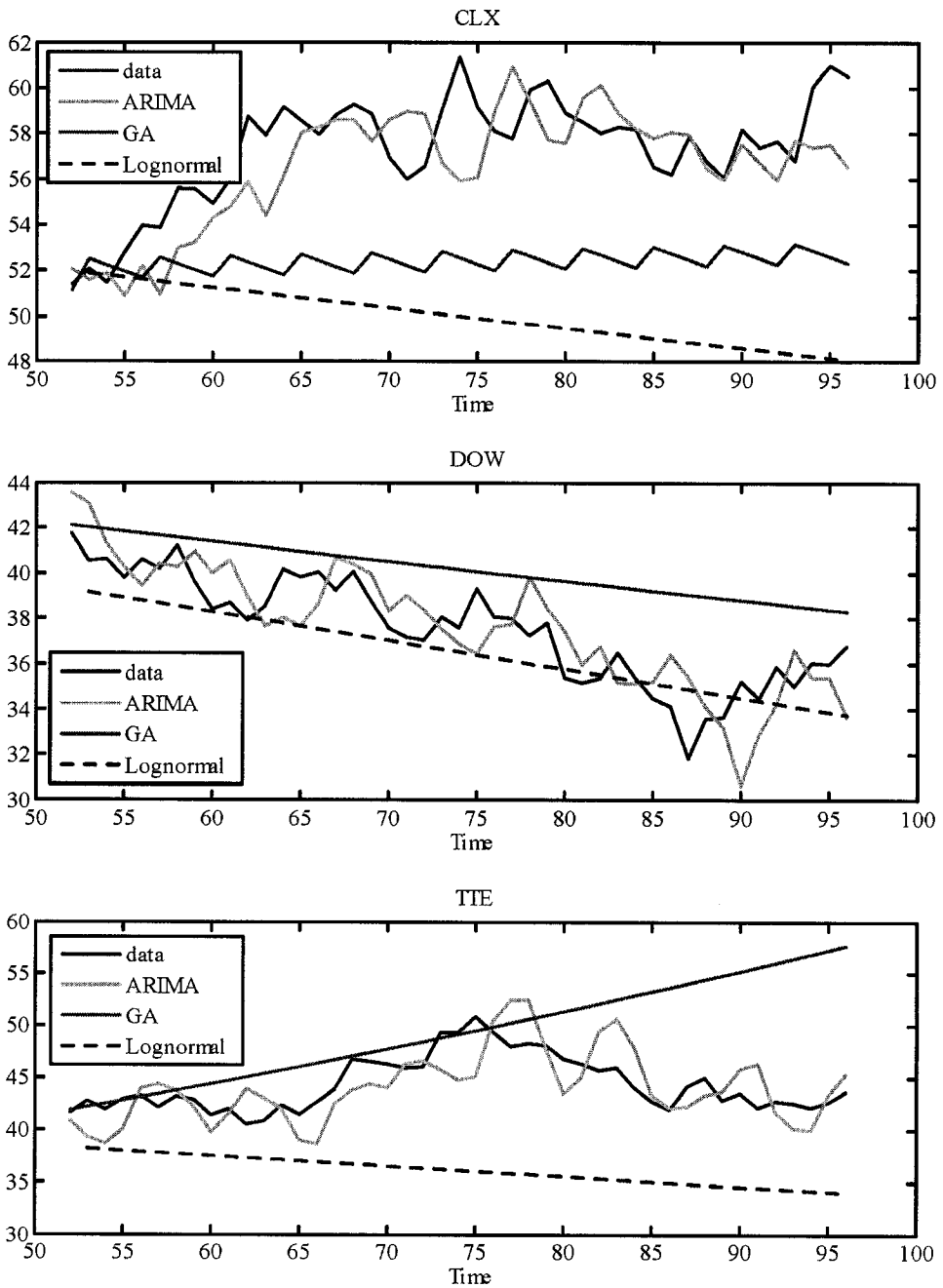


Figure 5.6: Forecast comparison (2007) for CLX, DOW and TTE

Model	Average RMSE over 200 stocks	Average MAD over 200 stocks
lognormal	50.54	6.40
ARIMA	58.61	3.71
GA	64.08	7.58

Table 5.1: Average RMSE and MAD of forecasts for all 200 stocks, year 2007

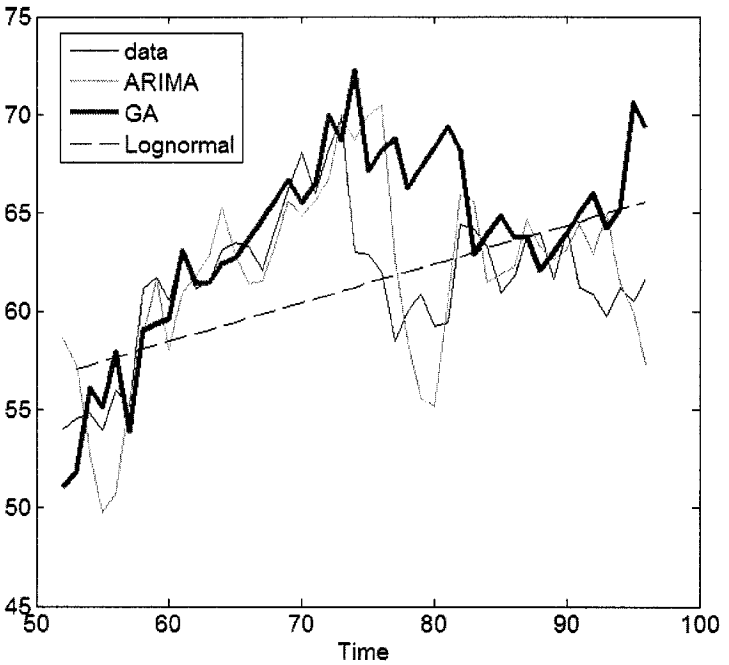
a cyclical behaviour (as for CCU and CLX). So at first view, ARIMA seems to be a better forecasting method than our model: ARIMA is never far from the general data trend to the point that is seems to "guess" the future quite well, even for a forecast period of one year. However, the ARIMA forecast is always a bit lagging with respect to the real data. The GA model is sometimes good, but most of the time is far off the mark. It tends to exhibit explosive or cyclical behaviour quite often.

The fact that the GA model often shows cyclical behaviour opens the door to an interesting question: what would be the performance of the GA model if we filtered out the cyclical forecasts? This can be done by a user of the system in a non-subjective way by having recourse to the Box-Pierce-Ljung "Q-statistic" mentioned in our discussion of ARIMA (Box *et al.*, 1970[10], Ljung *et al.*, 1978 [46]). This Q-statistic is a test statistic for the null hypothesis that there is no autocorrelation up to order k and is computed as:

$$Q = T(T+2)\sum_{j=1}^{k}\frac{\tau_j^2}{T-j}$$

, where τ_j is the j-th autocorrelation and T is the number of observations. Q is asymptotically distributed as a χ^2 statistic with degrees of freedom adjusted according to the number of terms in the ARIMA process.

Figure 5.7 shows on the top panel a *noncyclical* GA forecast (in red) for a stock.



Date: 09/20/08 Time: 18:27
Sample: 1 46
Included observations: 45

Autocorrelation	Partial Correlation		AC	PAC	Q-Stat	Prob
		1	-0.322	-0.322	4.9845	0.026
		2	0.087	-0.018	5.3605	0.069
		3	-0.007	0.017	5.3632	0.147
		4	-0.126	-0.136	6.1860	0.186
		5	0.309	0.256	11.250	0.047
		6	-0.142	0.042	12.348	0.055
		7	0.050	-0.006	12.485	0.086
		8	0.172	0.223	14.166	0.078
		9	-0.164	-0.013	15.746	0.072
		10	0.028	-0.157	15.795	0.106
		11	-0.039	-0.021	15.890	0.145
		12	-0.080	-0.117	16.298	0.178
		13	0.193	0.022	18.761	0.131

Figure 5.7: Top panel: a noncyclical GA forecast (red). Bottom panel: correlograms with Q-statistics.

Model	Average RMSE over 200 stocks	Average RMSE over noncyclical subset
lognormal	50.54	45.18
ARIMA	58.61	57.41
GA	64.08	39.22
Model	Average MAD over 200 stocks	Average MAD over noncyclical subset
lognormal	6.40	5.62
ARIMA	3.71	3.63
GA	7.58	4.58

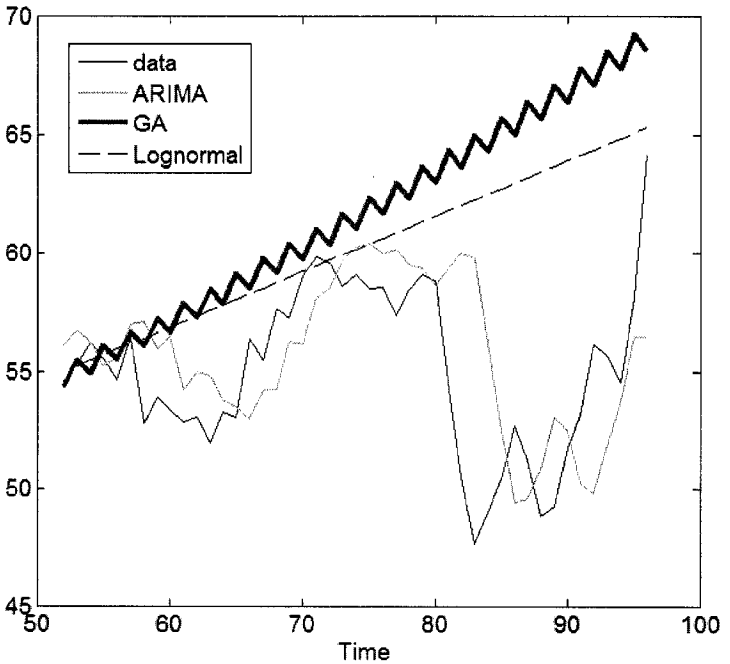
Table 5.2: Average RMSE and MAD of forecasts for all stocks and noncyclical subset

The bottom panel shows the autocorrelogram and partial autocorrelogram up to lag 13, since forecasts are conducted for 46 periods and we want at least 3 observations to estimate sample correlations. We also gave the Q-statistics for each lag as well as their p-values. We see that the null is not rejected at the 1% level for any of the 13 lags.

The top panel of Figure 5.8 displays a *cyclical* GA forecast (in red) for a stock. This is revealed by the correlograms of the bottom panel, since the null is rejected for all 13 lags. We therefore use this test procedure to retain only GA forecasts that exhibit no cyclical behaviour: any forecast with a Q-statistic rejected for any lag at the 1% level is excluded. This has the disadvantage of reducing drastically the number of usable forecasts; we find only 15 noncyclical forecasts over the original set of 200.

We can now revisit the previous table by comparing models again on that non-cyclical subset. Table 5.2 displays the results.

When the cyclical forecasts are removed, the GA model does better than ARIMA by the RMSE criterion, but is beaten by ARIMA by the MAD criterion. The lognormal model is the worst in both cases over that restricted subset. In all fairness, even if the models were ranked identically by both closeness criteria, this experiment would not be



Date: 09/20/08 Time: 18:53
Sample: 1 46
Included observations: 45

Autocorrelation	Partial Correlation		AC	PAC	Q-Stat	Prob
		1	-0.949	-0.949	43.309	0.000
		2	0.900	-0.010	83.152	0.000
		3	-0.851	0.026	119.61	0.000
		4	0.802	-0.018	152.82	0.000
		5	-0.755	0.020	182.95	0.000
		6	0.707	-0.025	210.07	0.000
		7	-0.661	0.016	234.38	0.000
		8	0.614	-0.031	255.95	0.000
		9	-0.569	0.013	274.98	0.000
		10	0.523	-0.036	291.53	0.000
		11	-0.480	0.011	305.83	0.000
		12	0.435	-0.039	317.94	0.000
		13	-0.392	0.010	328.09	0.000

Figure 5.8: Top panel: a cyclical GA forecast (red). Bottom panel: correlograms with Q-statistics.

conclusive statistically, since the number of stocks involved in the restricted subset is too small. We can however take note of an interesting fact: the GA objective function minimizes the MSE, and not the MAD. It is therefore encouraging to see that the GA model beats ARIMA with respect to that criterion, which is after all the one that we explicitly targeted.

We make one last point concerning RMSE and MAD. It could be argued that financial practitioners would be *a priori* more interested in models that are heavily penalized when they make large errors, which is what the RMSE criterion does. The reason is that large forecast deviations are likely to cause catastrophic financial losses, and any model that provides a closer fit for these large deviations is more useful than a model that fits well on average, but does not capture exceptional movements. This argument would tend to confirm that the minimization criterion should indeed be the RMSE, and that the GA would indeed be more useful than ARIMA.

What causes cyclical forecasts to appear in the GA model? A possible explanation lies in the fact that we selected genomes (through the objective function) that encouraged parsimonious representations. Obtaining cycles is a natural consequence of that selection pressure since cycles are about the most economical way to represent any given data, as in Fourier decompositions. But cycles do not do well out-of-sample, so they defeat the broader purpose of our model, which is forecasting. This phenomenon is very typical of genetic algorithms: the specification of the objective function can have consequences unintended by the modeller: evolution produces individuals exactly in accordance to the kind of selection pressure that is built-in.

We can check whether this parsimony pressure is responsible for cyclical behaviour

Number of generations	Noncyclical forecasts over 200 stocks
50 000	15
100 000	26
200 000	20
300 000	14
400 000	19
500 000	15

Table 5.3: Number of noncyclical forecasts by number of generations

	Average RMSE			Average MAD		
Number of generations	GA	lognormal	ARIMA	GA	lognormal	ARIMA
50 000	45.03	58.64	62.79	4.99	7.09	4.57
100 000	74.67	81.35	74.07	8.91	10.26	5.24
200 000	51.57	48.95	60.36	5.91	6.05	4.02
300 000	51.20	37.75	52.35	6.26	4.71	3.38
400 000	42.48	46.45	58.50	4.83	5.63	3.77
500 000	39.21	45.18	57.41	4.58	5.62	3.63

Table 5.4: Average RMSE and MAD of forecasts by number of generations

by examining how many noncyclical forecasts obtain when we evolve our model or a lesser number of generations on the same stocks. Table 5.3 shows the number of noncyclical forecasts over our 200 stocks, when we use to Q-statistic to weed out the cyclical forecasts.

Table 5.3 indicates no clear pattern. It seems that parsimony may not be responsible and that cyclicality may be a feature of the model. We will attempt to shed more light on that matter in the next section.

It may be interesting to see how the average RMSE and MAD of the forecasts produced when we evolve the algorithm for a lesser number of generations compare to those obtained when the model is fully evolved (table 5.4). Notice that in this table, the lognormal and ARIMA models do not give always the same RMSE, since they are evaluated not on all samples, but only those determined by the Q-statistic for the GA forecast.

Again, the results of table 5.4 send a mixed message. GA usually outperforms both models by the RMSE criterion except for the 300 000 generations noncyclical sample. ARIMA is always better by the MAD criterion. We conclude that it is probably better to let evolution follow its course, even if that means that only a small percentage of forecasts are going to be noncyclical; at any rate, there is no clear advantage of evolving less.

In practice, a end-user of the GA model with enough patience can always obtain a cyclical forecast: evolution can indeed be re-initiated from another random seed until a noncyclical forecast is obtained. If we model this process as a geometric random variable with the probability of success being 15/200, we would need on average 13 trials before getting a noncyclical forecast, so 120 hours of continuous processing time on a 2.2 GHz machine. This limits quite severely the practical use of the model, unless a way to exclude cyclical forecasts during the course of evolution is found, or if the end-users have access to a parallel implementation.

5.6 Tests of forecast encompassing

Ordering forecasts on the basis of RMSE or MAD is quite usual in the literature, and it is why we have provided these statistics in the previous section. However, an important shortcoming of this procedure lies in the fact that we cannot tell what constitutes a significant difference between RMSE or MAD scores: suppose for instance that the RMSE of model A is 2% higher than the RMSE of model B. Does that constitute sufficient reason to prefer model A over model B? We do not know, since the statistical distribution of RMSE and MAD scores is unknown. A remedy to this situation would be to get bootstrap esti-

mates of these scores and test significance on that basis. Since this is slightly controversial, another avenue is to use encompassing-in-forecast tests.

Encompassing-in-forecast tests (Chong *et al.* 1986 [13], Harrald *et al.* 1997 [29]) provide a way of deciding whether forecasts produced by one model provide significantly more information than forecasts from another model. The intuition behind these tests is the following: if the forecast errors of model A are explained by the model B forecasts, then this is evidence that model A does not add anything new to our knowledge. But in order to be certain that model B is better than model A, one has to examine the converse, namely whether the forecast errors of model B are *not* explained by the model A forecasts. If that last proposition is not true, the test is inconclusive as we might have a case of forecast multicollinearity. The various combinations that can occur are:

1. forecast errors of model A are explained by model B forecasts AND forecast errors of model B are unexplained by model A forecasts $\Longrightarrow$ then model B encompasses-in-forecast model A.

2. forecast errors of model B are explained by model A forecasts AND forecast errors of model A are unexplained by model B forecasts $\Longrightarrow$ then model A encompasses-in-forecast model B.

3. forecast errors of model A are explained by model B forecasts AND forecast errors of model B are explained by model A forecasts $\Longrightarrow$ then the test is inconclusive.

4. forecast errors of model A are unexplained by model B forecasts AND forecast errors of model B are unexplained by model A forecasts $\Longrightarrow$ then the test is inconclusive.

Regressor	Regressand		
↓	GA	ARIMA	Lognormal
GA	–	0.0356	0.0607
ARIMA	0.2761	–	0.1714
Lognormal	0.1047	0.0141	–

Table 5.5: Encompassing-in-forecast tests for the three models

To test whether the forecast errors of a model are explained or not by another, we perform the following ordinary least-squares regression:

$$\widehat{Y_{A_t}} - Y_t = \beta_0 + \beta_1 \widehat{Y_{B_t}} + \varepsilon_t$$

, where $\widehat{Y_{A_t}}$ refers to the forecast of model A at time t on the forecast interval and Y_t to the realized value. If the coefficient β_1 is significantly different from zero, we conclude that forecast errors of model A are explained by model B forecasts. Table 5.5 presents the p-values of the t-statistics of the regression coefficient β_1 for all possible combinations of regressor and regressand permitted by our three models:

At the 5% significance level, ARIMA is explained by GA, but GA is not explained by ARIMA. We conclude that GA encompasses-in-forecast ARIMA, and this is good news for our model, but this conclusion does not hold at the 1% level. More data would therefore be needed to confirm or infirm. Likewise, ARIMA is explained by the lognormal, but the lognormal is not explained by ARIMA. We conclude that the lognormal encompasses-in-forecast ARIMA. No other statistic pair is conclusive.

We now turn our attention to the question of whether the genome found as the best-of-generation individual faithfully reconstructs the underlying market dynamic.

5.7 Known dynamic reconstruction

Can our method recover a known genome from its forward-simulated price time-series? This is the question of the unicity of the reconstructed chromosome. Fortunately, it is quite easy to find an answer to that question experimentally.

The way we conducted this test is as follows. A genome was evolved over an arbitrary time-series of prices, and evolution was stopped after a few thousand generations. The result was an arbitrary chromosome with 50 fixed-length generalized trading rules that we call the "target" chromosome hereafter. We then forward-simulated 50 stock prices from that target chromosome with arbitrary initial conditions, which were then used as a time-series input to a normal run of the GA model. If the reconstruction process is effective, then the chromosome inferred by the model should come close to the target chromosome after a sufficiently large number of generations. Were this not to occur, we would conclude that the model cannot identify a unique chromosome based on the price information. Due to the low informational content of prices, the likely result of the experiment is a failure of reconstruction, as we discussed earlier. However, failure may not entire; it is indeed possible that the reconstructed chromosome share a degree of characteristics with the target chromosome, and that the model still prove to be useful even if reconstruction is not unique.

The main stumbling block of this experiment is that deciding whether one chromosome is close to another is not as straightforward as it may seem. We need to define a measure of similarity between chromosomes, but an easy solution such as computing the Levenshtein distance between alleles in corresponding positions is not suitable, for the following reasons:

- rules are in no specific order in chromosomes, so we can neither compare genes (i.e. rules) nor alleles based on their position within the genome.
- even though rules are of the fixed-length type, the chromosomes themselves are of variable length, with a maximum set to 70. We know as experiment designers that our target chromosome has length 50, but this is not an information that is made available to the GA model: the reconstructed chromosome can therefore end up having any length. So we need a similarity measure that will still work if presented with chromosomes of various lengths.
- recalling that "don't care" conditions are possible, one rule in one chromosome may be expressed as a set of rules in another. It is also possible that a rule or a rule set cancel another rule in a given chromosome, if the respective rule activation conditions are identical and if one rule is a "buy" and the other a "sell".

In light of these problems, we suggest to compare markets via their demand function, and not from the raw chromosomes, which are too complex to handle. First recall that by "demand" we mean the sum of individual rule demands when the chromosome is presented with a particular market run, which is a stack of H discretized historical prices in order. Using octiles, we denote this stack by $\{1,\ 2,\ \ldots,\ 8\}^H$. Formally, a demand function is a mapping:

$$D : \{1,\ 2,\ \ldots,\ 8\}^H \longmapsto \mathbb{R}$$

The set of H-tuples $\{1,\ 2,\ \ldots,\ 8\}^H$ constitutes a H-dimensional lattice over the integers 1 to 8. Recall that the market-maker is presented the demand for each period, and

applies its own response function, which is another mapping:

$$f : \mathbb{R} \longmapsto \mathbb{R}$$

and that this mapping produces a new price which is discretized based on an empirical price distribution:

$$g : \mathbb{R} \longmapsto \{1,\ 2,\ \ldots,\ 8\}$$

, and pushed onto the current price stack, which is another (rather straightforward) function

$$h : \{1,\ 2,\ \ldots,\ 8\}^H \longmapsto \{1,\ 2,\ \ldots,\ 8\}^H.$$

With given initial conditions, the repeated application of these four functions creates an orbit of a dynamical system. A market orbit will therefore repeat after a certain time, since the lattice is finite in size. This explains why prices often exhibit cyclical behaviour: as a matter of fact, they *always* exhibit cyclical behaviour, and all we did previously was to exclude cases where the period is too short. The largest possible period is obviously Q^H, where Q is the number of quantiles used for discretization, but in practice the average period is much shorter. Forecast cyclicality is therefore an unavoidable property of the model, and the frequency of cyclical behaviour should be inversely proportional to the value of the lookback parameter H. However, attempting to raise the value of H in order to produce a higher proportion of noncyclical forecasts may not be a good idea, since each run of the model would be lengthier both in time and memory requirements. Unsurprisingly,

there is a trade-off between computation time and output complexity, assuming we measure complexity by the forecast period.

Let us put that question aside and return to our discussion of the demand function. Since markets are defined by four functions, a demand function is the structural component of but "one quarter" of an artificial market. It is therefore only a partial and static representation of what is by nature dynamic. In spite of these deficiencies, demand functions provide appreciable insight into the nature of the artificial market, since they are easier to represent and compare to one another than market chromosomes.

Recalling that the optimized lookback parameter was found to be $H = 6$, we now describe a graphical representation of a demand function on the $\{1, 2, \ldots, 8\}^6$ lattice to $\mathbb{R}$. Since there is no convenient way to represent such a high-dimensional mapping, we project each point of the lattice to a two-dimensional square array indexed by a row and a column number:

```
i = 0;
For (a = 1 ... 8)
   For (b = 1 ... 8)
      For (c = 1 ... 8)
         For (d = 1 ... 8)
            For (e = 1 ... 8)
               For (f = 1 ... 8)
                  {
                      array[floor(i/8^3), i mod (8^3)] = demand(a,b,c,d,e,f)
                      i <- i + 1
                  }
```

Once this is done, we normalize global demand to the interval $[0, 1]$ and interpret each entry in the square array as a pixel, whose color is set with a colormap[8]. Obviously, a projection from a high-dimensional space to a square destroys the proximity relation: proximity of two pixels in the image does not mean proximity in the lattice. This unfortunately cannot be avoided, since we are compressing six dimensions into two, but at least we get pictures that can be compared visually, and we know that the human brain is a good judge of similarity. Independently of the visual observer, reconstructed markets can be objectively compared by computing the Pearson correlation between the pixel intensities of the two images.

To give the reader an example of this process for one run, figures 5.9–5.12 show such pixel colormaps. Figure 5.9 displays the target demand and a demand resulting from a chromosome inferred from evolving the GA model during 1000 generations. The colormap itself is displayed to the right of each image. Since the market is stationary and demand has been normalized, blue pixels correspond to situations when the market is selling, red pixels to when the market is buying, and green to no demand. Due to the projection that was

[8]Specifically, this is the "jet" colormap from MATLAB^{TM}, which maps low values to blue and higher values to red.

carried on, one can see many repeating motifs in these graphs, which are mere artefacts. The Pearson correlation (ρ) with the target market is displayed over each pixel map. When examining these graphs, one must recall that the goal is to evolve a pixel map that will reproduce the target color map of figure 5.9.

Judging from figures 5.9–5.12, it would seem that reconstruction was not effective for this run. After 500 000 generations, the Pearson correlation between the target and reconstructed markets ends up being small and negative. It would seem as if there is no relationship at all between target and reconstructed markets, and indeed the average correlation for 100 runs after 500 000 generations is close to zero. What may be the reason for that negative result?

A moment of reflection shows that this lack of correlation is normal and should in fact have been expected.

Recall that the model is applied to a training sample with length 50, namely one year of weekly observations. On the other hand, the graphical representation of the demand function shows demands for all possible inputs, a set of $8^6 = 262144$ different price stacks. There is huge discrepancy in size between the full range of behaviours that can be inferred from the genome and the restricted part of that range acted upon by evolution, which represents only a fraction of $\frac{19}{100000}$ of it.

We can liken this to the genotype–phenotype dichotomy in living organisms: in the context of our model, the full demand function represents the phenotype. This phenotype is determined by the genotype, but it can potentially exhibit a range of behaviours far wider than what may be revealed in a lifetime. Evolution does not select organisms on

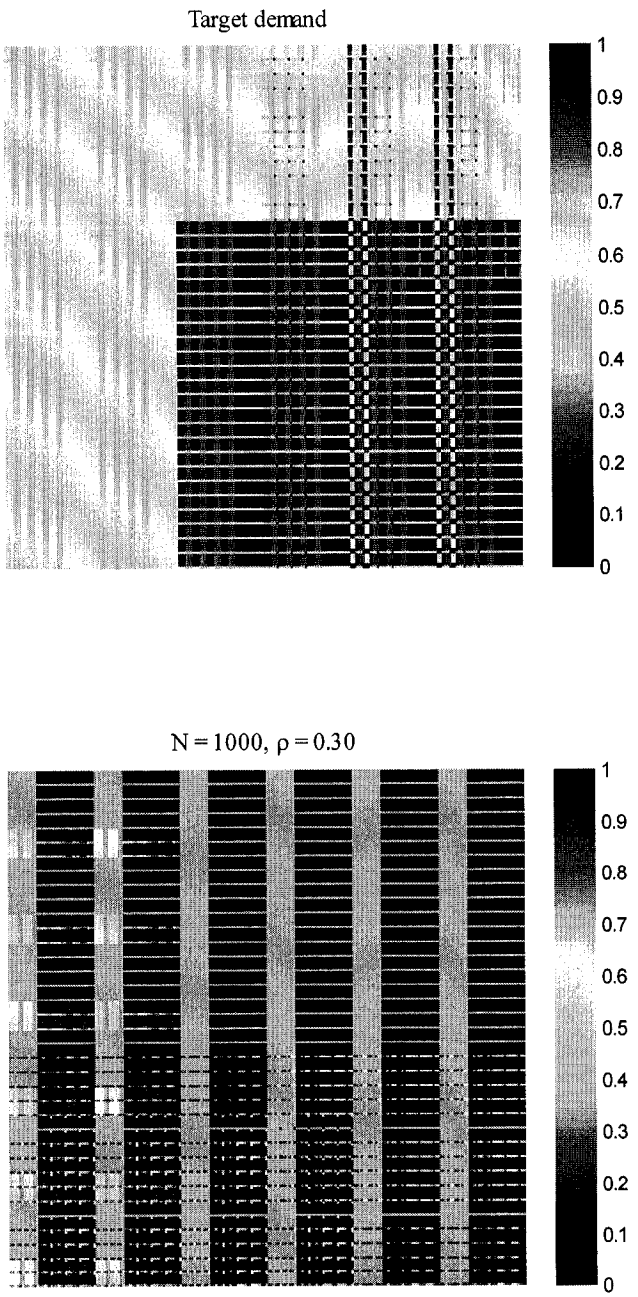


Figure 5.9: Graphical representation of demand functions, target and 1000 generations

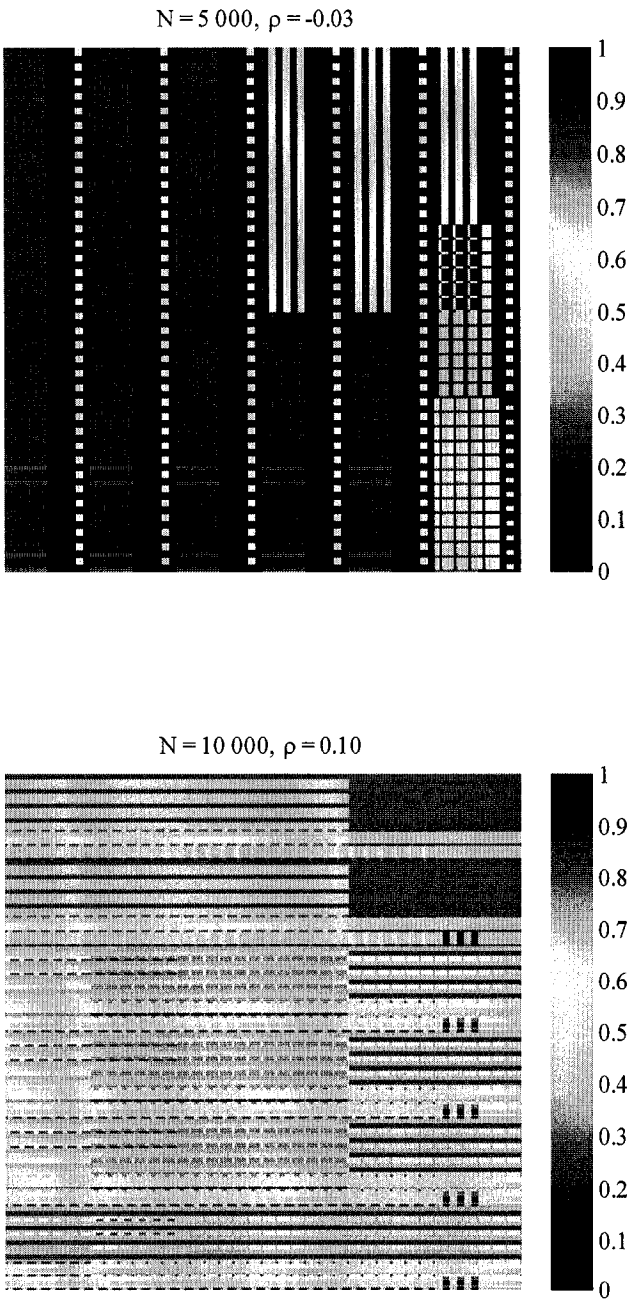


Figure 5.10: Graphical representation of demand functions, 5000 and 10000 generations

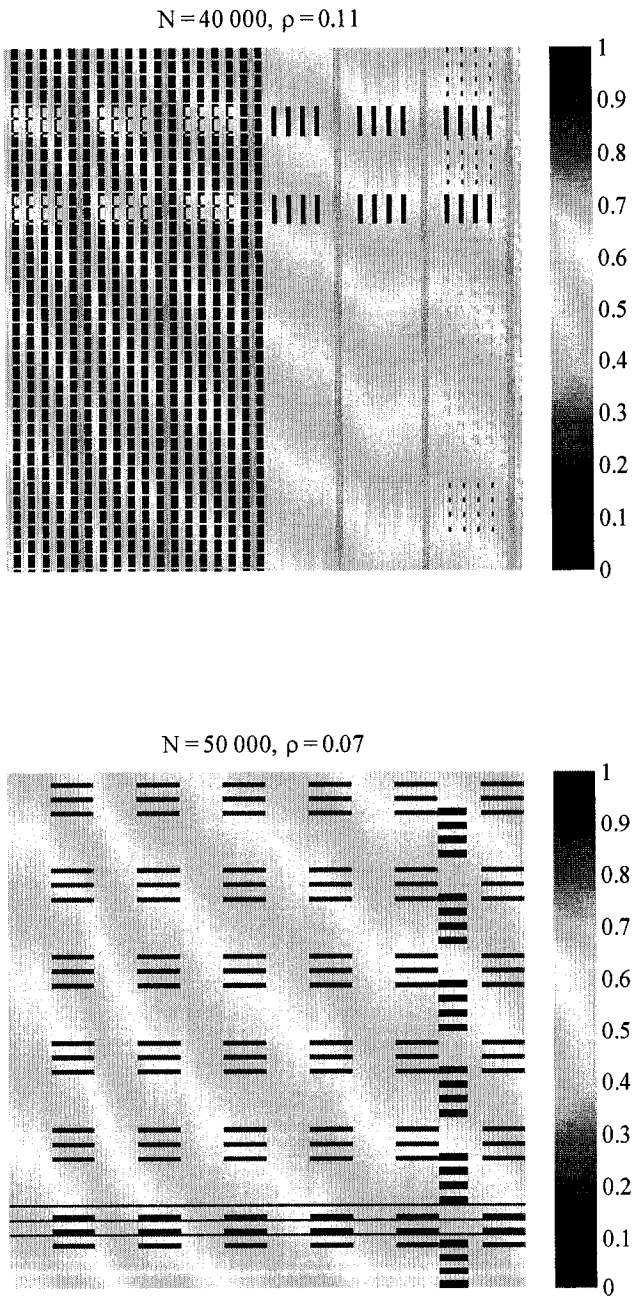


Figure 5.11: Graphical representation of demand functions, 40000 and 50000 generations

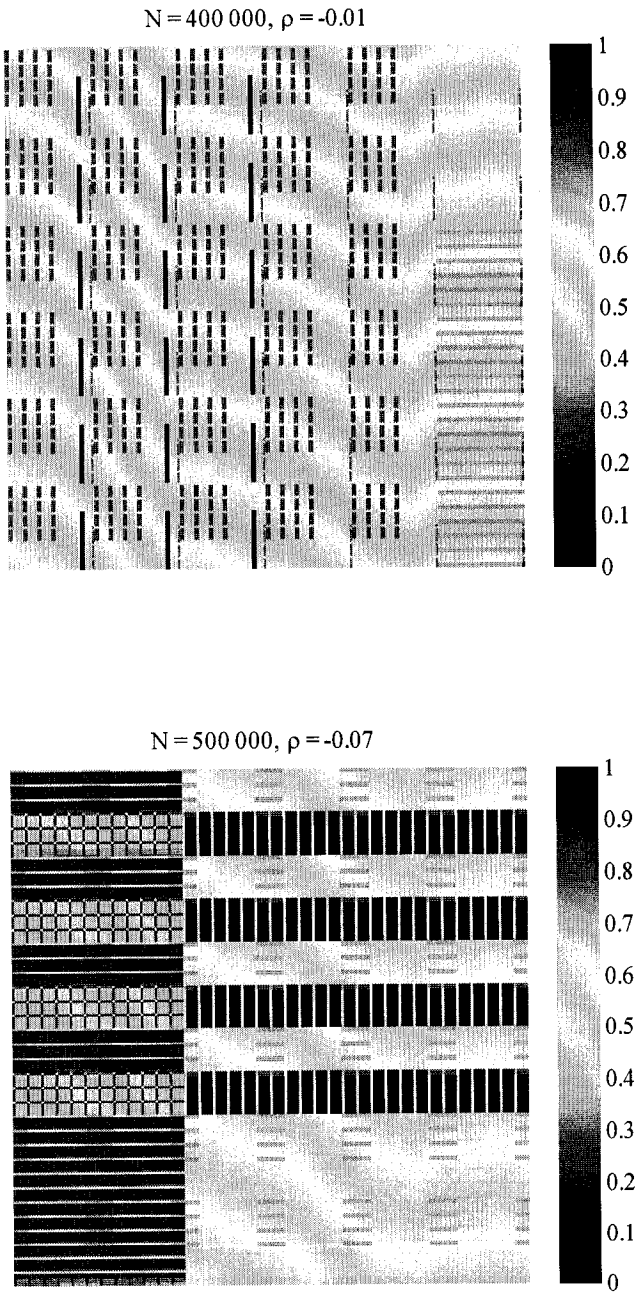


Figure 5.12: Graphical representation of demand functions, 400000 and 500000 generations

the basis of their possible range of response, but on the particular random circumstances that life happens to throw at them[9]. This simple realization confirms our intuition that the reconstruction process will not yield a unique chromosome; as a matter of fact, an astronomical number of artificial markets are compatible with any given price history of practical length. So we must fall back on Friedman's admonition to evaluate models based on their predictions, not their nature (Friedman, 1953 [22]).

So the question of reconstruction unicity can be answered with a resounding "no", and this answer is not specific to our model but rather a common characteristic of evolutionary systems acting upon complex phenotypes[10].

We may still examine whether evolution is effective on the part of the genome that is restricted to the historical set of H-tuples $\{1,\ 2,\ \ldots,\ 8\}^H$ that were encountered in the training sample. Since there were 50 of them, we remove the first to get arrays of 7^2 pixels. Figures 5.13–5.15 display the pixel colormaps of an arbitrary run for different numbers of generations.

We have reproduced the target demand in the top left corner of each of figures 5.13–5.15. The correlation between target and reconstructed demand is still very modest over that run, and the fact that oftentimes we have negative correlations is also intriguing. There seems to be considerable variance in the correlations. Naturally, no conclusion can be drawn from a single run, so we repeated the experiment 100 times. Table 5.6 shows the average correlations over these 100 runs by number of generations, and the standard

[9]Evolution is existentialist, not essentialist.

[10]We should be grateful this is the case. Bartok and Prokofiev were examples of phenotypes whose behaviour was not selected by evolution. If evolution were really effective for maximizing reproductive capacity, Bartok and Prokofiev would have been bacteria with millions of offspring, and probably not a long list of compositions.

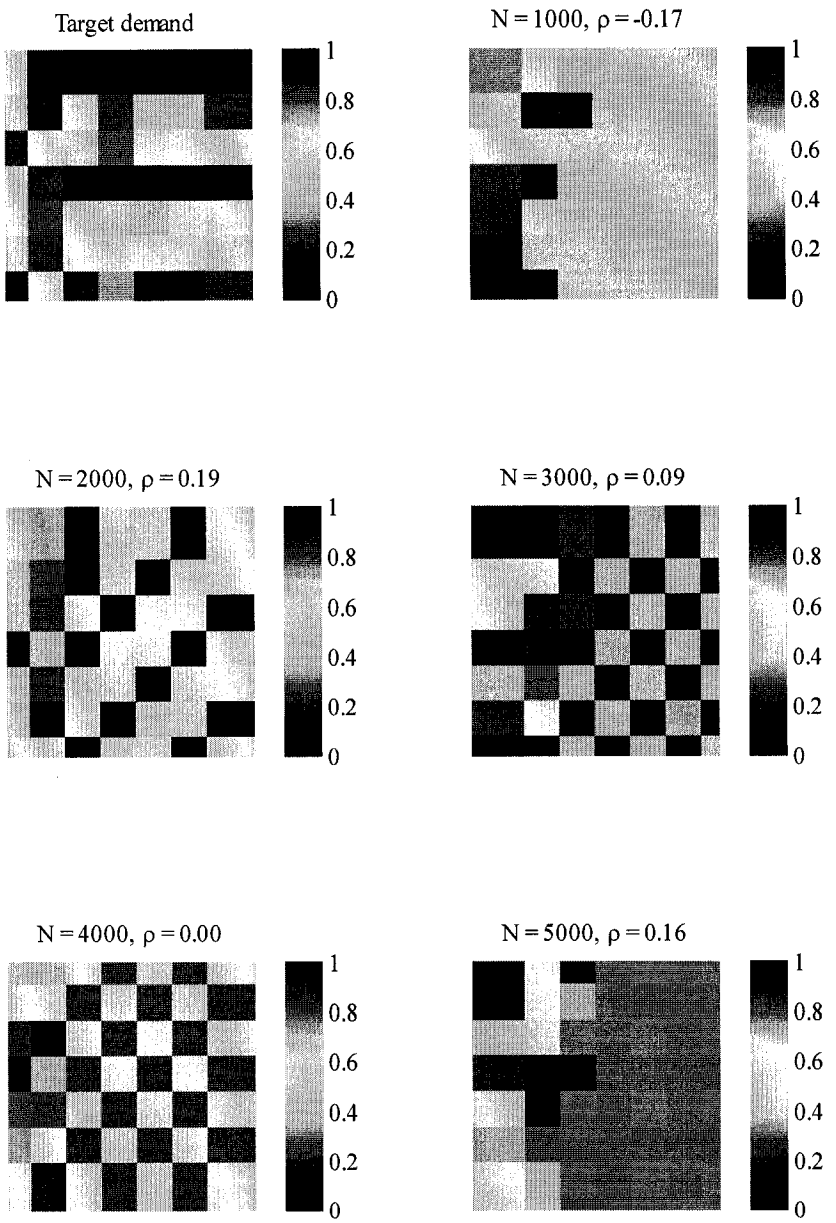


Figure 5.13: Graphical representation of restricted demand functions, 1000–5000 generations

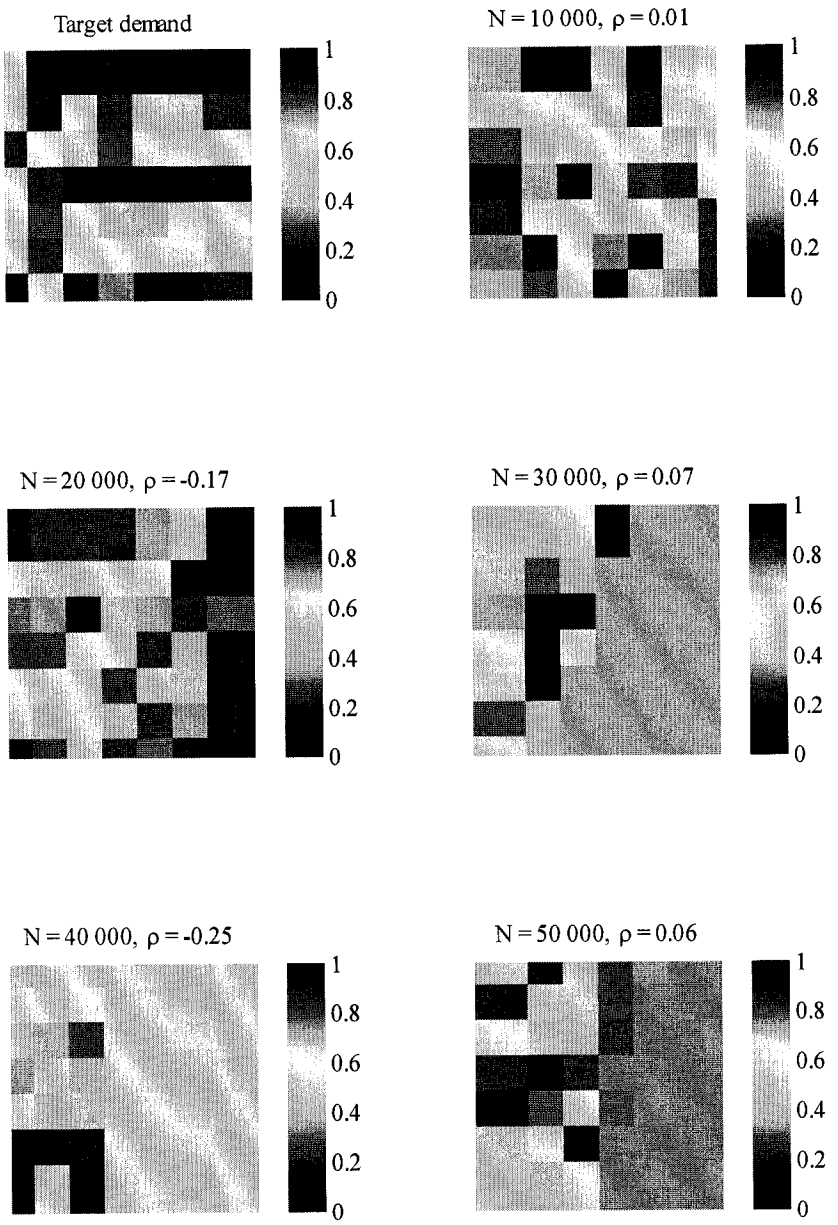


Figure 5.14: Graphical representation of restricted demand functions, 10000–50000 generations

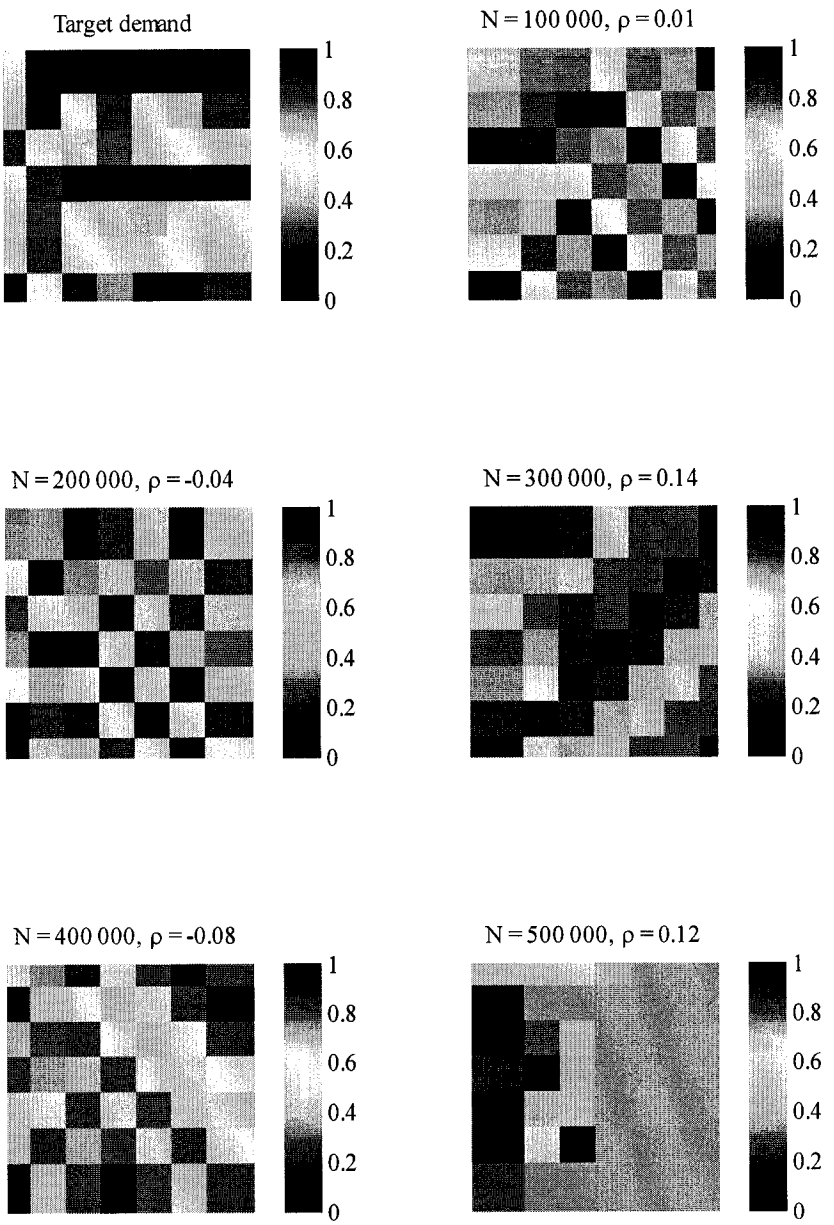


Figure 5.15: Graphical representation of restricted demand functions, 100000–500000 generations

Number of generations	Restricted demand $\overline{\rho}$ with target (100 runs)	$\widehat{\sigma_\rho}$
50 000	10.74%	7.38%
100 000	9.83%	6.64%
200 000	12.34%	6.52%
300 000	19.42%	5.39%
400 000	22.50%	6.27%
500 000	21.45%	5.14%

Table 5.6: Average correlation by number of generations

deviation $\widehat{\sigma_\rho}$ of these 100 correlations.

Table 5.6 confirms that there is much instability in the evolution process, but that there seems also to be a stabilization of the average correlation to a value of around 20%. The distribution of correlations is left-skewed, so correlations under average are more frequent than larger than average.

By that measure, we could say that the reconstruction process is not a great success, even when one looks at the demand restricted to the training sample. This again should not be surprising, since evolution does not optimize the demand, but the price proximity of the forward-simulated price to the training data. It is possible that a modestly or even negatively correlated reconstructed demand still do a good job in satisfying the price proximity condition.

Chapter 6

Conclusion and future work

We have presented a model of an artificial stock market that is simple enough as to allow calibration to a time-series of market prices. To our knowledge, this has been attempted in the literature only once with a very different artificial market and it is the first time that this method has been used for forecasting. The main drawback of the method is a rather prohibitive computational cost; instances of the model had to be run in parallel on a cluster machine. If run on a serial processor, the experiments described in this paper would have taken in excess of four years of continuous processing.

While the model cannot be said to faithfully reconstruct the underlying market structure, some preliminary evidence has been found that forward-simulation may provide a better forecast over one year compared to the lognormal model and ARIMA. In particular, the test of encompassing-in-forecast shows that the GA model encompasses ARIMA at the 5% level over a sample of 15 stocks for which the GA forecast was noncyclical. An important negative result is that when cyclical forecasts are *not* removed from consideration, the model

forecasting performance is clearly dominated by both ARIMA and the lognormal model. Cyclical behaviour is a natural consequence of the price discretization that is used to simplify rule representation within chromosomes.

Cyclical forecasts are obtained in more than 9 cases over 10, which limits the practical use of the model unless a way to raise the proportion of noncyclical forecasts is found. The most likely avenue of exploration is to raise the lookback parameter H to a higher value, which also raises the computation burden in a significant way.

Future work on that model will involve applying the model to more stocks (or other financial time-series such as currency rates) to confirm or infirm the forecasting power evidence that was obtained here, and on a wider range of parameters. In particular, it would be interesting to explore the consequences of raising Q (the number of quantiles used for discretization), but that raise will also need to be accompanied by a raise in the number of rules, since rules will become more specific and will cause the demand function to be sparser. To that effect, it would be helpful to not only run the model in parallel on a cluster, but parallelize the objective function evaluations themselves, preferably by harnessing the computing power of a graphical processing unit (GPU), which is easily available on desktops.

We have frequently mentioned how the results that we have obtained are often a direct but unintended consequence of the objective function in the evolution process. A natural avenue of exploration would be to modify that objective function to accommodate further desirable properties.

Another related question that was not explored here is whether running the model multiple times with different random seeds would supply any useful information on the

distributional properties of future prices. If so, the model could be used by risk managers as an alternative to Monte-Carlo or bootstrap methods and could have implications on the pricing of derivatives and other contingent or path-dependent assets. The nonlinearity of the model could perhaps capture aspects of reality that escape current simulation methods. Currently, only stochastic models such as the lognormal are used for such purposes.

Yet another line of exploration would be to generalize the model to bivariate or multivariate forecasts. Trading rules could involve two or more stocks and take decisions not only on the basis of a single stock, but on which stock from a given subset would constitute a better investment. Needless to say, the computational burden of such an extended model would be tremendous.

Bibliography

[1] Achelis, Steven B. *Technical Analysis from A to Z*. USA: Equis publishing, 2003.

[2] Arthur, W. Brian, et al. "Artificial Economic Life: A Simple Model of a Stock Market," *Physica D, 75* (1994).

[3] Arthur, W. Brian, et al. "Asset Pricing under Endogenous Expectations in an Artificial Stock Market." *The Economy as an Evolving complex System, II* Number XXVII in SFI Studies in the Sciences of Complexity, edited by W. Brian Arthur, et al., 15–44, Addison Wesley, 1997.

[4] Ashlock, Daniel. *Evolutionary Computation for Modeling and Optimization*. New York: Springer, 2005.

[5] Aster, Richard C., et al. *Parameter Estimation and Inverse Problems*. Amsterdam: Elsevier, 2005.

[6] Back, Thomas, et al. "Evolutionary Computation: Comments on the History and Current State," *IEEE Transactions on Evolutionary Computation, 1(1)*:3–17 (1997).

[7] Bass, Thomas A. *The Predictors: How a Band of Maverick Physicists Used Chaos Theory to Trade Their Way to a Fortune on Wall Street.* USA: Holt, 2000.

[8] Black, Fisher and Myron Scholes. "The Pricing of Options and of Corporate Liabilities," *Journal of Political Economy, 81*:637–659 (1973).

[9] Box, G. E. P. and G. M. Jenkins. *Time Series Analysis: Forecasting and Control.* San Francisco: Holden Day, 1970.

[10] Box, G. E. P. and David A. Pierce. "Distribution of Residual Autocorrelations in Autoregressive Integrated Moving Average Time Series Models," *Journal of the American Statistical Association, 65*:1509–1526 (1970).

[11] Brock, William B., et al. *A Test for Nonlinear Dynamics, Chaos and Instability: Theory and Evidence.* Boston: MIT Press, 1991.

[12] Chan, Nicholas Tung and C. Shelton. *An Electronic Market-Maker.* Technical Report, Cambridge, MA: Massachusetts Institute of Technology AI Memo, 2001.

[13] Chong, Y. Y. and D. F. Hendry. "Econometric Evaluation of Linear Macroeconomic Models," *Review of Economic Studies, 53*:671–690 (1986).

[14] Christie, W. G. and P. H. Shultz. "Why do NASDAQ Market-Makers Avoid Odd-Eight Quotes?," *Journal of Finance, 49*:1813–1840 (1994).

[15] Dawkins, Richard. *The Blind Watchmaker.* W. W. Norton, 1996.

[16] de la Maza, Michael and Deniz Yuret. *A Futures Market Simulation with Non-Rational*

Participants. Technical Report, Massachusetts Institute of Technology, Cambridge, MA: Artificial Intelligence Laboratory, 1900.

[17] Dickey, David A. and W. Fuller. "Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root," *Econometrica*, *49*:1057–1072 (1981).

[18] Dickey, David A. and Edward Said. "Testing for Unit Roots in Autoregressive Moving Average Models of Unknown Order," *Biometrika*, *71*:599–607 (1984).

[19] Ecemis, Ihsan, et al. "Interactive Estimation of Agent-Based Financial Markets Models: Modularity and Learning." *GECCO'05, June 25-29, 2005, Washington, DC, USA*, edited by ACM. 1897–1904. June 2005.

[20] Eiben, A. E. and J. E. Smith. *Introduction to Evolutionary Computing*. Berlin: Springer, 1998.

[21] Fagiolo, Georgio, et al. *Empirical Validation of Agent-Based Models: A Critical Survey*. Technical Report, Pisa, Italy: LEM Working paper 2006/14, Laboratory of Economics and Management, Sant'Anna School of Advanced Studies, May 2006.

[22] Friedman, Milton. *Essays in Positive Economics*. Chicago, Il, USA: University of Chicago Press, 1953.

[23] Garman, M. "Market Microstructure," *Journal of Financial Economics*, (3):257–275 (1985).

[24] Glosten, L. R. and P. R. Milgrom. "Bid, Ask and Transaction Prices in a Specialist

Market with Heterogeneously Informed Traders," *Journal of Financial Economics*, (14):71–100 (1985).

[25] Gode, D. and S. Sunder. "Allocative Efficiency of Markets with Zero-Intelligence Traders: the Market as a Partial Substitute for Individual Rationality," *The Journal of Political Economy*, *100*(1):119–137 (1993).

[26] Goldberg, David. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston: Kluwer, 2002.

[27] Greene, William H. *Econometric Analysis*. New Jersey, USA: Prentice-Hall, 2003.

[28] Hamilton, James D. *Time Series Analysis*. Princeton, NJ: Princeton University Press, 1994.

[29] Harrald, Paul G. and Mark Kamstra. "Evolving Artificial Neural Networks to Combine Financial Forecasts," *IEEE Transactions on Evolutionary Computation*, *1*(1):40–52 (1997).

[30] Harris, Larry. *Trading and Exchanges: Market Microstructure for Practitioners*. USA: Oxford University Press, 2002.

[31] Iacus, Stefano M. *Simulation and Inference for Stochastic Differential Equations*. Springer Series in Statistics, USA: Springer, 2008.

[32] Ip, G. and S. Craig. "NYSE's 'Specialist' Probe Puts Precious Asset at Risk: Trust," *The Wall Street Journal, April 18, 2003* (2003).

[33] Jong, Kenneth De. "Parameter Setting in EAs: a 30 years Perspective." *Parameter Setting in Evolutionary Algorithms* Number 54 in Studies in Computational Intelligence, edited by Fernando G. Lobo, et al., 1–18, Springer, 2007.

[34] Joshi, Shareen, et al. "Technical Trading May Create a Prisoner's Dilemma." *Proceedings of the Sixth International Conference on Computational Finance, New York, january 5–11, 1999*. January 1999.

[35] Joshi, Shareen, et al. "Financial Markets Can be at Sub-Optimal Equilibria," *Computational Economics*, *19*(1):5–23 (2002).

[36] Kaboudan, Mahmoud A. "Genetic Programming Prediction of Stock Prices," *Computational Economics*, *6*(3):207–236 (2000).

[37] Kahneman, Daniel and Amos Tversky. "Prospect Theory of Decisions Under Risk," *Econometrica*, *47*(2) (1979).

[38] Kennedy, Peter. *A Guide to Econometrics*. Cambridge, Massachusetts: MIT Press, 1998.

[39] Knight, Frank Hyneman. *Risk, Uncertainty and Profit*. Boston, MA: Houghton Mifflin, 1921.

[40] Koza, John. *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[41] Kyle, Albert S. "Continuous Auctions and Insider Trading," *Econometrica*, *53*(6):1315–1338 (1985).

[42] LeBaron, Blake. *Calibrating an Agent-Based Financial Market to Macroeconomic Time-Series*. Technical Report, Waltham, MA: Working paper, Graduate School of International Economics and Finance, Brandeis University, 2003.

[43] LeBaron, Blake, et al. "Time-Series Properties of an Artificial Stock Market," *The Journal of Economic Dynamics and Control*, *23*:1487–1516 (1999).

[44] Levy, Moshe, et al. *Microscopic Simulation of financial Markets*. London, UK: Academic Press, 2000.

[45] Lintner, John. "Security Prices, Risk and Maxima Gains from Diversification," *The Journal of Finance*, *20* (1965).

[46] Ljung, G. M. and G. E. P. Box. "On a Measure of Lack of Fit in Time Series Models," *Biometrika*, *65*:297–303 (1978).

[47] Lévy, Paul. *Processus Stochastiques et Mouvement Brownien*. Paris: Gautier-Villars, 1965.

[48] Merton, Robert. "Theory of Rational Option Pricing," *Bell Journal of Economics and Management Science*, *4(1)*:141–183 (1973).

[49] Mirowski, Philip. *More Heat than Light: Economics as Social Physics, Physics as Nature's Economics*. Cambridge, UK: Cambridge University Press, 1989.

[50] Murphy, Christopher M., et al. "Artificial Stupidity," *The Journal of Portfolio Management*, *Winter 1997*:24–27 (1997).

[51] Nevmyvaka, Yuriy, et al. "Fundamental Issues in Automated Market Making." *Computational Economics: A Perspective from Computational Intelligence* Computational Intelligence and its Applications, edited by Shu-Heng Chen, et al., 118–148, Idea Group Publishing, 2006.

[52] Rechenberg, I. "Cybernetic Solution Path of an Experimental Problem." Library Translation 1122, Royal Aircraft Establishment, 1965.

[53] Sharpe, William F. "Capital Asset Prices: A Theory of Market Equilibrium Under Conditions of Risk," *Journal of Finance*, *19*(3):425–442 (1964).

[54] Shiller, Robert J. *Market Volatility*. Cambridge, MA: MIT Press, 1989.

[55] Shirayaev, A. N. *Probability*. Springer, 1989.

[56] Wagner, Neal and Zbigniew Michalewicz. "Parameter Adaptation for GP Forecasting Applications." *Parameter Setting in Evolutionary Algorithms* Number 54 in Studies in Computational Intelligence, edited by Fernando G. Lobo, et al., 295–309, Springer, 2007.

[57] Walras, Leon. *Éléments d'économie politique pure, ou théorie de la richesse sociale*. 1874.

[58] White, Halbert. "A Reality Check for Data Snooping," *Econometrica*, *68*(5) (2000).