Financial Time Series Analysis

Chaos and Neurodynamics Approach

Antonio Sawaya

Master thesis report,

in fullfillment of the requirements for the master degree in Applied Artificial Intelligence.

June 7, 2010

HÖGSKOLAN DALARNA

DEGREE PROJECT
Computer Engineering

Programme Applied Artificial Intelligence	Reg number	Extent 15 ECTS
Name of student ANTONIO SAWAYA	Submission date 31-05-2010	
Supervisor Dr. Hasan Fleyeh	Examiner	
Company/Department	Supervisor at the Company/Department	
Title Financial time series analysis: Chaos and neurodynamics approach		
Keywords Chaos, fractals, neural networks, cognitive process. time series, stock markets, finance, artificial intelligence, Hurst, Lyapunov, Takens Embedding Theorem, Monte Carlo simulation, predictive modeling		

Abstract

This work aims at combining the Chaos theory postulates and Artificial Neural Networks classification and predictive capability, in the field of financial time series prediction. Chaos theory, provides valuable qualitative and quantitative tools to decide on the predictability of a chaotic system. Quantitative measurements based on Chaos theory, are used, to decide a-priori whether a time series, or a portion of a time series is predictable, while Chaos theory based qualitative tools are used to provide further observations and analysis on the predictability, in cases where measurements provide negative answers. Phase space reconstruction is achieved by time delay embedding resulting in multiple embedded vectors. The cognitive approach suggested, is inspired by the capability of some chartists to predict the direction of an index by looking at the price time series. Thus, in this work, the calculation of the embedding dimension and the separation, in Takens' embedding theorem for phase space reconstruction, is not limited to False Nearest Neighbor, Differential Entropy or other specific method, rather, this work is interested in all embedding dimensions and separations that are regarded as different ways of looking at a time series by different chartists, based on their expectations. Prior to the prediction, the embedded vectors of the phase space are classified with Fuzzy-ART, then, for each class a back propagation Neural Network is trained to predict the last element of each vector, whereas all previous elements of a vector are used as features.

Acknowledgments

Special thanks to,

Dr. Hasan Fleyeh, *my supervisor, for his illuminating advices, efforts and genuine support,*

Mr. Imad Laham*, for his friendship, support, constructive criticism, expertise and passion in the fields of trading and computing, and the long nights he spent analyzing results,*

Dr. Elia Abi Jaoudé, *for his friendship and continuous encouragement*,

Mr. Roberto Pasquali*, for his help with the formatting of data collected from the NYSE,*

Mr. Paolo Trasarti *for the nice photos of the pine tree, there where we observed how nature fantastically embedded its processes.*

Dedication,

To Dr. Samir Geagea, whom sacrifices and wisdom have flourished a lovely Lebanese garden, with the right evolution heuristic of survival…

To all the Lebanese Forces martyrs for whom I owe my existence…

To my family: father, mother, wife, sister and brothers: Antoine, Fadia, Dimitra, Sabine, Charbel and Fady; for their infinite and unconditional love, patience and trust.

Last but not least, to my cousin, Joseph Sawaya; thank you for guiding me from up there, over the rainbow…

Non nobis Domine, non nobis, sed nomini Tuo da gloriam…

Table of Contents

List of figures

List of Tables

Introduction

1.1. The issue

In its' short version, the problem we have in hand is "Time series analysis". Since the very rise of the Homo Sapiens, we became aware of time and curious about its' relation with the events around our existence. From rainfall to river discharges, from the movement of the stars to the length of a day, from earthquakes to volcano eruptions, from the branching in a tree to the branching in the mammalian lung, and so on are the infinite examples of problems reduced to time series. Fed by the beauty of the human mind, curiosity and ingenious aspect of humanity, the scientific community across time, has developed a valuable set of tools to analyze time series. For the purpose of this paper, time series related to the stock market shall be analyzed, with the hope that such analysis and approaches can be extended to other fields. The complexity given by the stock market movement has been for long, the attraction to many mathematicians. One of the first examples in history pertaining to this curiosity, dates back to the fifth century BC, as recorded by Aristotle in [1]: "There is the anecdote of Thales the Milesian and his financial device, which involves a principle of universal application, but is attributed to him on account of his reputation for wisdom. He was reproached for his poverty, which was supposed to show that philosophy was of no use. According to the story, he knew by his skill in the stars while it was yet winter that there would be a great harvest of olives in the coming year; so, having a little money, he gave deposits for the use of all the olive-presses in Chios and Miletus, which he hired at a low price because no one bid against him. When the harvest-time came, and many were wanted all at once and of a sudden, he let them out at any rate which he pleased, and made a quantity of money. Thus he showed the world that philosophers can easily be rich if they like, but that their ambition is of another sort. He is supposed to have given a striking proof of his wisdom, but, as I was saying, his device for getting wealth is of universal application, and is nothing but the creation of a monopoly. It is an art often practiced by cities when they are in want of money; they make a monopoly of provisions."

In our current days, what Thales did is called "Option Trading", actually, the Thales trade is referred to as "The First Option Trade in history "[2]. Nowadays, the stock market is decisively more complex than the olive harvest then, yet, speculation, although vantages of mathematical tools, is still based on intuition and human intelligence as back then.

When Louis Jean-Baptiste Alphonse Bachelier (March 11, 1870 – April 28, 1946), modeled the stochastic process called Brownian Motion, in his PhD thesis "The theory of speculation", he stated that: "the mathematical expectation of the speculator is null". His work gave birth lately in the 1960s to the "Efficient Market Hypothesis(EMH)", developed by Professor Eugene Fama at the University of Chicago Booth School of Business. The Efficient Market Hypothesis assumes that the market follows a random walk, and thus, the statistical structure of the market falls into the boundaries of normal distribution. The EMH assumes that on average the population is correct, even no one person is. This means that the theory assumes a random behavior of individuals participating to the market, where some are wrong and others are right(in a normal distribution),

however, the market itself is always right. Nowadays, market pricing, options, swaps, derivatives and portfolio theory, all assume a random walk of the market, yet, such a theory was more than once challenged by some traders whom exact predictions redirect us again to our ongoing quest on the nature of intelligence. Jesse Lauriston Livermore (July 26, 1877 — November 28, 1940), is only one of these examples; by the age of fifteen, he had earned profits of over $1000 (which equates to about $20,000 today), and lately, he was known for making and losing a fortune, as well as short selling the markets during the stock market crashes in 1907 and 1929, against all the expectations of normal distribution, random walk and Gaussian statistics.[3]

As mentioned before, the work presented in this paper, intends to deal with the analysis of financial time series from a cognitive approach. What we basically try to do, is to predict future values of the time series, for as many future steps ahead as possible. In this context it becomes inevitable to adopt Artificial Neural Networks(ANN) and other tools available in the literature of Artificial Intelligence. As intelligence itself has yet no clear definition, the paper intends to investigate also into Chaos and Fractal theories in the context of intelligence and ANN.

The problem in hand raises two fundamental questions: The first, what is deterministic? The second, what is stochastic? The reason for such questions lies in the heart of the chaos theory itself and the three stages of analysis involved; diagnose, count, measure. First we determine the intrinsic dimension of the system – the minimum number of degrees of freedom necessary to capture its essential dynamics. If the system is very turbulent (description of its long time dynamics requires a space of high intrinsic dimension) we are, at present, out of luck. As a consequence, we can deduce that our real problem lies in the dimension of the phase space, where every growth of such a dimension is indeed a growth in complexity to extents that are currently beyond our reach.[6]

The basic stochastic models available in the literature of computational finance, under the EMH are[4]:

a. Autoregressive(AR)
b. Moving average(MA)
c. Autoregressive moving average (ARMA)
d. Autoregressive integrated moving average (ARIMA)
e. Autoregressive conditional heteroskedastic (ARCH)

Some variations exist for each of the above basic models. Many papers have compared various models, and implemented the above in ANN showing better results. Other papers focused on the comparison of various neural networks that implement the above models[5]. In this work, we keep being interested in the comparison of various models and ANN architectures, however, our main interest is in features selection and in the statistical structure of the problem.

We justify such a choice by assuming that the information available to traders regarding a security, bond or equity is the same. Some speculative traders however, have higher success rates than others; it is possible then, that the consideration of the data and the selection of features in the cognitive process of successful speculators is different than that in others. The use of chaos theory postulations shall hopefully help us to extract more relevant features than provided by the classical models. The random walk, assumes that trading the market and playing dices in the casino have

both a Gaussian statistical structure, thus, follow a normal distribution. Yet, a new emerging theory, "Fractal Market Hypothesis", suggests that the statistical structure of the EMH is based on unsupported assumptions.

We are basically looking for self similar structures and thus features, called fractals, and through one of our experiments "The Game of Chaos" we shall observe one of the postulates of chaos theory: "Local randomness and global determinism can coexist to create a stable, self similar structure."[4]. Finally, we shall present a suggestion for an embeding that we believe is a pseudo-computational method for clustering the data in self similar patterns, we recognize that such a method should be further justified and rationalized for it to be considered valid, a task that we leave for future work.

1.2. The Scope

A chartist (also known as a technical trader or technical analyst) is one who utilizes charts to assess patterns of activity that might be helpful in making future predictions. Most commonly, chartists use technical analysis in the financial world to evaluate financial securities.[7] For example, a chartist may plot past values of stock prices in an attempt to denote a trend from which he or she might infer future stock prices. The chartist's philosophy is that "history repeats itself".[8] It is a matter of fact, that many chartists, do make excellent speculations. It is obvious, that each of them has his own way of looking at data and of seeing "cycles". It is a very hard task to prove that there are cycles in the stock charts and in economics. Most mathematicians have abandoned that search, for example: "Using the Fourier analysis, we assume that irregularly shaped time series are the sum of periodic sine waves, each with different frequency and amplitude. Granger(1964) was the first to suggest that spectral analysis could be applied to market time series, his results were inconclusive. Over the years, various transformations of the data were performed to find evidence of cycles that, intuitively, were felt to be there… but none was found. Finally, most of the field gave up and decided that the cycles were like the lucky runs of gamblers-an illusion".[4] The alternative to the above and all previous studies seems to be in Chaos theory, actually there is an emerging theory since the 1990', "Fractal Markets Hypothesis". In chaos theory, non periodic cycles exist, they have an average duration, but the exact duration of a future cycle is unknown. Furthermore, unlike Euclidean geometry that allows mostly perfect similarities and symmetries, Chaos theory refers to fractal geometry, that makes allowance for non perfect similarities and symmetries.

The scope of this work, is to use Chaos theory postulations for qualitative and quantitative analysis, pertaining to the predictability of a given time series. The approach, makes extensive use of embedding, as an assumption on how a chartist might look at a series. Should we accept that a chartist do really visually embed a time series, then, his cognitive process subject to experience and lessons from the past, allows him to make predictions. Therefore, the use of neural networks, once the series is embedded, becomes the only analogy we have with the real process to attempt good predictions.

Background and literature review

When dealing with time series, one should inevitably analyze and understand the context it refers too. Such an understanding would give an insinuation on the assumptions already made or that might be made to describe the underlying process. Any modeling process is inherent to the context, where sometimes qualitative analysis based on observations in the studied fields, justify the assumptions. In our case, we are dealing with financial time series, and thus, a review of the basic literature, assumptions and models of computational finance becomes inevitable. We are also referring to Chaos theory and Artificial Neural Networks, and thus, it is fair enough that we also provide a review on that as well. By nature, the field we are analyzing is interdisciplinary, and thus requires openness to many scientific fields. In what follows, we shall provide the essentials of such a review, in relation to this work.

1.3. The Efficient Market Hypothesis(EMH)

Based on the work of Bachelier, the EMH was formally developed by Professor Eugene Fama at the University of Chicago Booth School of Business. The AMH basically hypothesizes that:

- The markets are efficient as for the information they provide
- Consistent returns are not possible in excess of average market returns

The EMH has three major versions:

- Weak EMH
- Semi-Strong EMH
- Strong EMH

Weak EMH states that current prices of traded assets reveal all past available information.

Semi-Strong EMH states the same as weak EMH, plus the fact that instant change in current prices reveals new public information.

Strong EMH states that all the above is true, plus the fact that prices instant change, also reflect hidden information, or what is known as "insider" information.

The stock market so far, has been modeled and ,probably, mostly traded based on the EMH, yet, the hypothesis stands in front of many detractors since the 1990s, and more violently after the 2007-2010 financial crisis.

In our daily lives, we are all faced by decisions that we have to make, whether we are driving a car or trading the markets. New information acquired, play a crucial role in the decision process at any level. EMH suggests that traders, when faced with new information, would either overreact or under react. Actually, the EMH cares less for the correct reaction of the single trader,

it only requires that the whole traders' reaction be a random normal distribution. Should we accept the EMH, our analysis would become focused on normal distribution and Gaussian statistics.

Much of the critics to the EMH comes from behavioral finance, as well as from empirical evidences, however, the question whether the markets are efficient or not is still not answered. [9]

1.4. Basic stochastic models in computational finance

When faced with Gaussian statistics, a hypothesis should be tested against the normal distribution. The EMH assuming a random behavior of market participants, has its' best tools in random number generators, stochastic processes and normal distribution. For this purpose, it becomes important the review the features exhibited by the basic stochastic processes mostly adopted in computational finance[4]:

1. Autoregressive(AR)
2. Moving average(MA)
3. Autoregressive moving average (ARMA)
4. Autoregressive integrated moving average (ARIMA)
5. Autoregressive conditional heteroskedastic (ARCH)

1.4.1. **Autoregressive(AR)**

An Autoregressive process(AR), assumes that the value in a series observed at time (t), is correlated to one and/or more previous value(s) of the same series, by a parameter, a constant and a white noise. The order of an Autoregressive process is given by the number of previous values correlated to the current result, p As such, an Autoregressive process, AR(p), is expressed as follows:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \qquad \text{eq(1)}$$

Where,

p, is the order of the model,

$\varphi_1, \cdots, \varphi_p$, are the parameters,

C, is a constant

and,

ε_t , is a white noise.

1.4.2. **Moving average(MA)**

A Moving Average(MA) process, assumes that the value in a series observed at time (t), is correlated to the mean of the data plus some factors of a white noise. The order of a Moving Average process is given by the number of previous noise terms, q, assumed to be correlated to the current result. As such, a Moving Average process, MA(q), is expressed as follows:

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \qquad \text{eq(2)}$$

Where

q, is the order of the model,

μ is the mean of the data,

$\theta_1, ..., \theta_q$, are the parameters

and,

$\varepsilon_t, \varepsilon_{t-1}, ...$ are a white noise

1.4.3. **Autoregressive Moving Average(ARMA)**

An Autoregressive Moving Average, assumes that the current value in a series observed at time (t), is correlated to the previous noise terms in the data and, one and/or more previous values of the same series, by a factor, a constant and a white noise. The order of an Autoregressive Moving Average process is given by the number of previous values and previous noise terms' values assumed to be correlated to the current result, p and q. As such, an Autoregressive Moving Average process, ARMA(p,q), is expressed as follows:

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}. \qquad \text{eq(3)}$$

Note that the ARMA(p,q) model, contains both AR(p) and MA(q) models.

1.4.4. **Autoregressive Integrated Moving Average(ARIMA)**

Both AR and ARMA models can be absorbed into a more general class of processes[4]. Consider eq(3) of the ARMA process, it is expressed in terms of previous values of the series X_{t-i} and in terms of a white noise values, $\varepsilon_{t-1,}$ of an unobserved time series. From Eq(3) (the constant term *C* is omitted from simplicity) :

$$\left(1 - \sum_{i=1}^{p} \varphi_i X_{t-i}\right) X_t = \left(1 - \sum_{i=1}^{q} \boldsymbol{\theta}_i \varepsilon_{t-i}\right) \varepsilon_t \qquad \text{eq(4)}$$

Given a time series $X = \{X_1, X_2, X_3 \ldots X_n\}$, the lag operator L is defined as:

$$LX_t = X_{t-1} \quad , \text{for all } t > 1$$

L can be raised to integer powers so that:

$L^{-1}X_t = X_{t+1}$, in general: $L^k X_t = X_{t-k}$ eq(5)

By expressing X_{t-i} and ε_{t-i} in terms of the lag operator L, in equation 4, we have:

$$\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right) X_t = \left(1 - \sum_{i=1}^{q} \boldsymbol{\theta}_i L^i\right) \varepsilon_t \qquad \text{eq(6)}$$

An ARIMA process, assumes that the polynomial $\left(1 - \sum_{i=1}^{p} \varphi_i L^i\right)$ has a unitary root of multiplicity *d* and so, an ARIMA(p,d,q) process, is expressed as follows:

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d X_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t \qquad \text{eq(7)}$$

1.4.5. **Autoregressive Conditional Heteroskedasticity (ARCH)**

Autoregressive Conditional Heteroskedasticity (ARCH), developed by Engle(1982), have become popular for a number of reasons: “

1. They are a family of nonlinear stochastic processes, as opposed to the linear-dependent AR and MA processes;
2. Their frequency distribution is a high-peaked, fat tailed one;
3. Empirical studies have shown that financial time series exhibit statistically significant ARCH. “ [4]

An ARCH, considers the variance of the current white noise term(otherwise referred to as error term or innovation) as a quadratic function of the previous noise terms. Thus, in ARCH models, the time series is as well sampled from a normal distribution, however, the expected variance is conditional on what it was previously. The basic ARCH was expressed as follows:

$$C_n = S_n * e_n$$

$$S_n^2 = f_0 + f * e_{n-1}^2 \qquad \text{eq(8)}$$

Where e = a standard normal random variable
f = a constant [4]

The ARCH model was modified to make the s variable dependent on the past as well. GARCH, or generalized ARCH was then formalized by Bollerslev(1986) in the following manner:

$$C_n = S_n * e_n$$

$$S_n^2 = f_0 + f * e_{n-1}^2 + g * S_{n-1}^2 \qquad \text{eq(9)}$$

As one can note, all above models are based on a white noise term. That is, an independent identically distributed random variables(iid), sampled from a normal distribution.

1.5. Random Walk

From the definition and the notation of the ARIMA(p,d,q) model given in equation 7, consider a special case, ARIMA(0,1,0), then, equation 7 would yield:

$$X_t = X_{t-1} + \varepsilon_t \qquad \text{eq(10)}$$

Which is the straight forward definition of a Random Walk. The Random Walk Hypothesis was first developed at the Massachusetts Institute of Technology, by professor Paul Cootner [10], [11]. The hypothesis states that the prices of the stock market proceeds according to a random walk, and thus cannot be predicted.

1.6. Brownian Motion

Brownian motion, named after Scottish botanist Robert Brown, is a mathematical model that describes the movement of particles over a fluid, where such movement seems random. It is a continuous-time stochastic process. In 1908, Albert Einstein predicted that, under Brownian motion, the displacement of a particle in a fluid in any direction, is proportional to the square root of time. More specifically, if T is the particle's thermodynamic temperature, then the motion of the particle is subject to a diffusion coefficient D=KBT/b, where KB is the Boltzmann's constant and b is the linear drag coefficient on the particle. An thus, the root mean square displacement of the particle in any direction at time t is $\sqrt{2Dt}$.[12]

1.7. Neural Networks

Neurons are the core components of the nervous system. In the field of neuroscience, biological neural networks, are real groups of neurons chemically connected to each others, and perform a specific physiological function. A single neuron can connect to many other neurons where the connections established are called synapses. Signaling between neurons is electrical and chemical. A neuron is basically formed of one cell body referred to as soma, one axon and many dendrites. While the soma can frequently give rise to multiple branching dendrites to result in a complex "dendritic tree", it never gives rise to more than one axon. The axon itself may branch many times before conneting to the dendrites of another neuron. Thus, a neuron can be seen generally as a processing cell, where the majority of synapses are sent from, through the axon of one neuron to the dendrites of another. There are, however, many exceptions to these rules: neurons that lack dendrites, neurons that have no axon, synapses that connect an axon to another axon or a dendrite to another dendrite, etc.

Specialized groups of neurons like sensory and motor neurons, as well as many interneurons that connect neurons to other neurons within the same region of the brain or spinal cord, are sought as the core of the cognitive process of the brain. Cognitive modeling is a field that attempts physical and mathematical modeling of the neural system's behavior. Artificial intelligence have successfully applied artificial neural networks models to some problems like; speech recognition, image analysis and adaptive control. It is important to note that most of the currently employed artificial neural networks in the field of AI are based on statistics, estimation, optimization and control theory. Most of the modeling, actually, is based on many assumptions. Many models were developed in an attempt to describe the cognitive process, each based on a different paradigm, inspired by the way biological neural networks process data and learn. Consequently, simulated neurons became common to all implementations of artificial neural networks. The network topologies of ANN fall basically into two types: feed forward and recurrent, and the learning approaches are basically: supervised learning, unsupervised learning reinforcement learning. In this work, we shall work with unsupervised and supervised learning. The classical statistical models used to analyze time series are parametric models, whereas ANN are nonparametric universal function approximators, which makes them good candidates for time series' predictive modeling.[13],[14],[15]

1.7.1. **Multi Layer Perceptrons**

Warren McCulloch and Walter Pitts [16]. argued that neurons with a binary threshold activation function were analogous to first order logic sentences. In 1949 Donald Hebb proposed that "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased." [17]. The finding of Hebbs, suggested that repeated firing from a neuron to another strengthens the

connection between them, and thus, more importantly, such a process was fundamental for learning and memory.

Frank Rosenblatt, based on the modeling of McCulloch and Pitts as well as the findings of Hebb, proposed a binary classifier called the percepron In 1957. The Perceptron maps an input vector *X* of real values to a single binary value output *f*(*x*) where a real value weight vector W is assigned to the input vector X, such that each value x_i in X has one corresponding weight w_i in W.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

Other activation functions such as the logistic sigmoid, and the hyperbolic tangent were later introduced for f(x) to yield real values output.

A multi layer Perceptron (MLP) neural network, is composed of an input layer, one or more intermediate layers and an output layer, where each layer is a row of perceptrons. A feed forward topology on a MLP requires all signals to propagate in the network only in one direction. Recurrent topologies, permit feedback across the layers.

1.7.2. **Supervised Learning – Back propagation networks**

The back propagation algorithm, is a supervised learning approach implemented on a feed forward network topology. The process of learning in such an algorithm can be split in two main phases:

- Forward propagation of the signal
- Weights update

Although the name "Back propagation" suggests a signal traveling backwards in the network, actually, the error is propagated backward in the network and distributed on the single units in the layers. In the first phase, the training pattern input is forwarded through the network in order to result in an action potential and an activated output. Then the activated output is cross validated with the training pattern's target(thus back propagated) in order to distribute the error differences(the deltas) on all output and hidden units. In the second phase, each unit's delta, is multiplied by its' input activation to obtain the gradient of the weight then the weight is moved in the opposite direction of the gradient by subtracting a ratio of it. The ratio to be subtracted influences the speed and the quality of learning and is thus called: The learning rate. The two phases are repeated until satisfactory results are obtained. The number of times the above is repeated is called the number of epochs. Two approaches can be adopted to train a back propagation neural network. One approach is "Sequential Training Mode" the other is "Batch Training Mode". [14]

Two approaches can be adopted to train a network, One approach is "Sequential Training Mode" the other is "Batch Training Mode". [14] The "Sequential Training Mode" became

"highly popular (particularly for solving pattern-classification problems) , for two important reasons:

- The algorithm is simple to implement
- It provides effective solutions to large problems" [14]

A momentum term, is a generalization of the Delta rule. The momentum is "A simple method of increasing the rate of learning yet avoiding the danger of instability" [14]

In general, the back-propagation algorithm cannot be shown to converge, and there are no well-defined criteria for stopping its operation. Rather, there are some reasonable criteria
Consequently, the following criteria can be used:

- Gradient of weight (derivative of the error surface with respect to the weight vector) reaches a sufficiently small gradient threshold
- The rate of change in the average squared error per epoch is sufficiently small

Actual algorithm for a 3-layer network (only one hidden layer):

```
  Initialize the weights in the network (often randomly)
  Do
         For each example e in the training set
              O = neural-net-output(network, e) ; forward pass
              T = teacher output for e
              Calculate error (T --O) at the output units
              Compute delta_wh for all weights from hidden layer to output
layer ; backward pass
              Compute delta_wi for all weights from input layer to hidden
layer ; backward pass continued
              Update the weights in the network
  Until all examples classified correctly or stopping criterion satisfied
  Return the network  [13]
```

1.7.3. **Unsupervised Learning – K-Means clustering**

Unsupervised learning, aims at categorizing input patterns in meaningful classes. K-means clustering, developed by J. MacQueen (1967) and then by J. A. Hartigan and M. A. Wong around (1975) is the simplest form of an unsupervised machine learning algorithm used to classify or to group vectors representing objects, based on attributes/features into K number of group where K is a positive integer. The grouping is done by minimizing an objective function, in this case a squared error function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

where $\left\| x_i^{(j)} - c_j \right\|^2$ is a chosen distance measure between a data point $x_i^{(j)}$ and the cluster centre c_j, is an indicator of the distance of the ***n*** data points from their respective cluster centers.

Step 1. Begin with a decision on the value of k = number of clusters

Step 2**.** Put any initial partition that classifies the data into k clusters. One may assign the training samples randomly, or systematically as the following:

1. Take the first k training sample as single-element clusters
2. Assign each of the remaining (N-k) training sample to the cluster with the nearest centroid. After each assignment, recomputed the centroid of the gaining cluster.

Step 3 **.** Take each sample in sequence and compute its distance from the centroid of each of the clusters. If a sample is not currently in the cluster with the closest centroid, switch this sample to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.

Step 4 **.** Repeat step 3 until convergence is achieved, that is until a pass through the training sample causes no new assignments.

If the number of data is less than the number of cluster then we assign each data as the centroid of the cluster. Each centroid will have a cluster number. If the number of data is bigger than the number of cluster, for each data, we calculate the distance to all centroid and get the minimum distance. This data is said belong to the cluster that has minimum distance from this data.

1.7.4. **Unsupervised Learning – Adaptive Resonance Theory**

Adaptive resonance theory, was developed by Stephen Grossberg in 1976 at the department of Cognitive and Neural systems at Boston University. ANN models of the ART were implemented for both, supervised and unsupervised learning. Models of unsupervised learning include ART 1 [18] for binary input patterns and fuzzy ART [19] for analog input patterns. Carpenter and Grossberg argue that a learning system, must be capable of plasticity in order to learn about significant new events yet maintains its' stability when faced with irrelevant events. The ART, is thus presented, as a solution for the Stability-Plasticity dilemma. The ART architecture is basically formed of two core subsystems: The attentional subsystem and the orienting subsystem. The former processes familiar events, while the later resets the attentional subsystem when faced with an unfamiliar event, thus requesting a new recognition code. Top-down expectations are built by the attentional sybsystem that help to stabilize the learned bottom-up codes of familiar events. In this work, we are interested in Fuzzy-ART for clustering analogue input patterns.

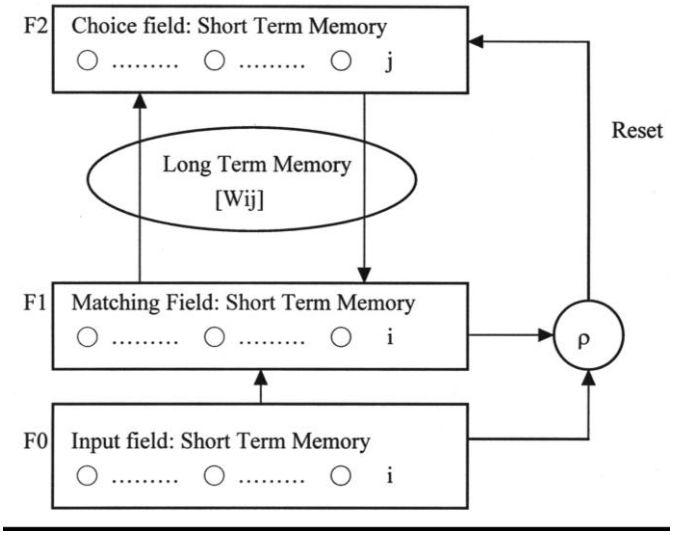


FIGURE 0.1: ART COMPONENTS

Fuzzy ART Architecture [20]

- It consists of two subsystems:
- The Attentional subsystem
- The Orienting subsystem.
- The attentional subsystem consists of two fields of nodes denoted as field F1and field F2.

Attentional Subsystem[20]:

- The F1 field is called the *input field* because input patterns are applied to it.
- The F2 field is called the *category or class representation field* because it is the field where category representations are formed.
- These category representations represent the clusters to which the input patterns belong (input patterns presented at the F1 field).

Orienting Subsystem[20]:

- Single node (called the *reset node*), which accepts inputs from the F1 field
- The input pattern applied across the F1 field
- The output of the reset node is affecting the nodes of the F2 field.

Preprocessing[20]:

Some preprocessing of the input patterns takes place before they are presented to Fuzzy ART. The first preprocessing stage transforms every input pattern into an input pattern whose every component lies in the interval [0, 1].

The second preprocessing stage takes an input Pattern

$$a=(a_1,a_2,\ldots,a_M)\ 0<=a_i<=1 \text{ and } 0<=i<=M$$

and transforms it into an input pattern **I** such that

$$I=(a,a^c) = (a1,\ldots,aM,a1c,\ldots,aMc)$$

Where

$$aic=1-ai \text{ and } 1<=i<=M$$

The aforementioned transformation of the input patterns is called complement coding. The complement coding of the input patterns is performed in Fuzzy ART at a preprocessor field designated by F0

1.8. Fractals and Chaos Theory

There is still no formal mathematical definition of the term "fractal"[4], however, fractals do have some measurable characteristics and properties suitable for modeling. The first and most important property of fractals is self-similarity; where parts somehow resembles and are related to the whole. An obvious example on that similarity is the Sierpinski triangle. The Sierpinski triangle is a basic example of simple self-similar sets, which are mathematically generated pattern that can be reproducible at any magnification or reduction. More complex sets include, the Julia set and the Mandelbrot set. To produce such a triangle one would proceed as follows:

- Start with any triangle in a plane
- Shrink the triangle to ½ height and ½ width, make three copies, and position the three shrunken triangles so that each triangle touches the two other triangles at a corner Note the emergence of the central hole —because the three shrunken triangles can between them cover only 3¾of the area of the original.
- Repeat step 2 with each of the smaller triangles. [4]

FIGURE 0.2: CONSTRUCTION OF THE SIERPINSKI TRIANGLE

As can be observed, the similarities in the Sierpinski triangle are precise, however, in real life, the self similarity is "qualitative"; that is, the object or process is similar at different scales, spatial or temporal, statistically. Each scale resembles the other scales, but is not identical. Individual branches of a tree are qualitatively self similar to the other branches, but each branch is also unique. This self similar property makes the fractal scale-invariant: It lacks a characteristic scale from which the other derive. [4]

Perhaps, one simple description of Fractals and Chaos is the following: Fractals are disorder in space while chaos is disorder in time.

The fractal dimension, D, is a statistical quantity that gives an indication of how an object fills its' space, as one zooms down to finer and finer scales. There are many specific definitions of fractal dimension.

In Euclidean geometry, objects exist and are described in integer dimensions, while fractal geometry objects can be described in non-integer dimensions. Euclidean geometry is a description of lines, ellipses, circles, etc. Fractal geometry, rather, is described in algorithms – a set of instructions on how to create a fractal.

The world as we know it is made up of objects which exist in integer dimensions, single dimensional points, one dimensional lines and curves, two dimension plane figures like circles and squares, and three dimensional solid objects such as spheres and cubes. However, as one can intuitively note, most of the objects in nature are better described better with fractional and non integer dimensions. The fractal dimension in the Euclidean plane, is thus a value between one and two, indicating how much space a fractal occupies as it curves and twists. The more a fractal fills a plane, the more it approaches two dimensions. Similarly, a landscape, with bumps and hills, would be somewhere between 2 and 3 depending on how much it fills the three dimensional space [4].

1.8.1. **The Chaos Game**

Chaos theory postulates that, in fractal time, chaos and order can coexist to form a stable and deterministic structure. One example to support this claim is "The Chaos Game", where randomness creates a stable structure, in this case, the Sierpinski triangle. The game goes as follows:

- Start with a triangle in the plane
- Label the vertices A(1,2), B(3,4), C(5,6)
- Chose randomly any point inside the triangle, plot it and can call it "Start point" S
- Throw a fair dice
- Based on the value obtained by the dice, plot the midpoint of the segment from S to the vertex whose label corresponds to that value. Now this point become S, the "Start Point"
- Repeat from step 4, for 500 or more times

The result of the above is a Sierpinski triangle.[4]

1.8.2. **The Hurst Exponent**

Early in the 20^{th} century, H.E.Hurst(1900-1978) was a hydrologist, working on the Nile River Dam Project. He was supplied back then with extensive data that the Egyptians kept of the Nile River's overflow; 847-year record. One of the primary aim when designing a dam, is the storage capacity of the reservoir. To decide on the reservoir's storage capacity, one must estimate the inflow and the need for outflow of water. The basic assumption to begin with, is that the water inflow is a random process. Based on Einstein's(1908) work on Brownian motion, Hurst assumed that the T to the one-half rule given by the Brownian motion, can be used to test the Nile River's overflows for randomness. What Hurst had actually, was a time series for annual overflows of the Nile River. After calculating the range of the system, the T to the half had to be generalized, so Hurst calculated the Rescaled Range (R/S)n, which is the adjusted range divided by the standard deviation of the sample, and thus, the generalization becomes:

$(R/S)n = c*n^k$ Where c is a constant, n is the time and k is the Hurst exponent.

Then Hurst calculated the constant K which is the slope of the log(R/Sn) versus log(n). That variable was later named H by Mandelbrot in honor of Hurst, and then defined the fractal dimension as D=2-H. For the Nile River, Hurst calculated H to be 0.91, which meant that the rescaled range was increasing at a faster rate than the square root of time, in other words, the distance covered was more than that of a system under Brownian Motion(i.e. more than a random process would do).

The range rescaling to zero mean and standard deviation of one performed by Hurst then, allowed diverse phenomena and time periods to be compared, eliminating the need of a characteristic scale. The Hurst exponent, which is a value between 0 and 1, has the following interesting properties:

- If H=0.5 then the series is completely random(Brownian motion)
- If 0.5<H<=1 then the series is said to be persistent, i.e. if it was going up then down in the previous period, it will then more probably continue going up then down in the next period.
- If 0<H<0.5 then the series is said to be Anti-persistent.(up then down will more probably become down then up) [4]

1.8.3. **The Lyapunov exponent**

The Lyapunov exponent is a quantity that describes the rate of separation of two neighbouring trajectories. The divergence of tow trajectories at time (t) with initial separation $\delta \mathbf{Z}_0$ at t=0, is expressed as:

$|\delta \mathbf{Z}(t)| \approx e^{\lambda t} |\delta \mathbf{Z}_0|$ where λ is the Lyapunov exponent.

Different orientations of initial separation vectors would result in different rate of separation and thus, a whole spectrum of the Lyapunov exponents. Reference is commonly made to the largest λ, called the Maximal Lyapunov Exponent:

$$\lambda = \lim_{t \to \infty} \frac{1}{t} \ln \frac{|\delta \mathbf{Z}(t)|}{|\delta \mathbf{Z}_0|}.$$

λ >0 is usually considered an indicator of chaos, λ<0 is usually considered an indicator of a mean reverting behavior and λ=0 is a characteristic of cyclic behavior. [21]

1.8.4. **Takens' embedding theorem**

In 1981, Floris Takens' suggested a simple method to analyze chaotic time series called embedding. The easiest form of embedding an equation is to plot the pairs of points xt and xt+1 of a series, if recognizable patterns are encountered then, the next point can be predicted by interpolation using a

smooth function. By plotting the pairs of points Xt and Xt+1, we have embedded the time series in two dimensions. Embedding thus, can be extended to more dimensions, and can be written as:

Xt = xt,x(s+t),x(2s+t),x(3s+1),…,x(ds+t)

Where X is the embedded vector, s is the separation and d is the embedding dimension. Takens' theorem states that, once correctly embedded, a chaotic series can be perfectly modeled by a smooth function. Takens also states that, the correct embedding dimension, the separation and the smooth function must be calculated empirically. Two main methods attempt to calculate the correct embedding and separation: The False Nearest Neighbor method and the Differential Entropy method. [21]

1.8.4.1. The False Nearest Neighbor Method

The False Nearest Neighbor Method, is a used to determine the minimal sufficient embedding dimension m, proposed by [22]. The idea behind the False Nearest Neighbor Method is quite intuitive, it first requires that trial embedding dimension is chosen, then, for each embedded vector, the nearest neighbor is determined and the Euclidean distance is calculated. Successively, the embedding dimension of the vector and its neighbor is increased by one by adding the appropriate values. If the distance between the new vectors is changed beyond a threshold, then the nearest neighbor is considered a "False Nearest Neighbor". Once the process is over for all embedding dimensions, the dimension that yielded the lowest number of "False Nearest Neighbors" is chosen as the best parameter.

The major assumption behind this method is that, the best embedding dimension is that where the neighboring vectors remain neighbors under most dimensions of the phase space.

1.8.4.2. The Differential Entropy Method

The measure of Entropy, is based on the idea that systems in nature tends to move from order to disorder. Quantitatively, Entropy measures the disorder in a system. In information theory, the Shannon Entropy measures the uncertainty associated with a random variable.

For a discrete random variable $X$with $n$outcomes $\{x_i : i = 1, \ldots, n\}$, the Shannon entropy denoted by $H(X)$, is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

where $p(x_i)$is the probability density function of outcome x_i.

Differential entropy differs from the, above in that the random variable need not be discrete. Given a continuous random variable X with a probability density function $f_X(x)$, the differential entropy $H(X)$ is defined as:

$$-\int_{-\infty}^{\infty} f_X(x) \ln f_X(x)\, dx = -\langle \ln f_X(x) \rangle.$$

In [23], Temujin Gautama, proposes a Differential Entropy based method to determine the best choice for the embedding parameters in Takens' theorem. "The set of optimal parameters (mopt, topt), yields a phase space representation which best reflects the dynamics of the underlying signal production system. Therefore, it is expected that this representation has a minimal differential entropy (minimal disorder), and that a deviation from (mopt,topt) results in an increase."
The method is thus based on an optimization of the Differential Entropy. In this method, the assumption is that the best embedding parameters are those that yield the minimum Differential Entropy.

The predictive model

1.9. Justification

In the absence of a clean cut theory, empirical methods are used, an thus, two main methods are mostly used to calculate the embedding dimension and the separation parameters in Takens' theorem, the False Nearest Neighbor and Differential Entropy. Both methods have been extensively used and do provide relatively acceptable results. In this work, we are trying to approach the problem of financial time series prediction from a cognitive approach. Our choice falls into ANN as the smooth function required by Takens' theorem to model a chaotic series. The problem we remain in front of, is to calculate the embedding dimension and the separation. Given that ANN are universal function approximators, and given the structure of Takens' embedding theorem, where the theorem itself contains no restrictions on the dimension and the separation parameters, except the fact that they should be empirically calculated. Given that the theorem states as well, that also the smooth function to model the series based on the embedding and the separation should be empirically calculated; for the purpose of research, we shall allow ourselves to break the scheme, and assume that for one or more embedding dimensions and separations in Takens' theorem, there may exist respectively one or more functions that model the series.

To illustrate the above, we should keep in mind that what we are trying to do here, is to find out how some "Chartists"(people in the stock market who make speculations by looking at the previous data of a time series in a chart), make good guesses. The eye somehow captures similar cycles and patterns in a chart. We agree that most of the time such cycles are pure illusions and biases of the chartist, however, some chartists do make good guesses, and it is hard to confirm that they all look parametrically at the same portion or features in the data and that all do use the same "function" to model the data. Consider the following example of S&P 500 index daily close price for 4000 day.

0.1 S&P500 DAILY CLOSE PRICE

For the eye, the above red up trends and green down trends in the chart looks as cycles. Although the metrics in price value and timing are different, however, probably whoever looks at that chart is biased by what appears to be cycles.

Fractals are based on self similar structures, so, we are no more looking for perfect symmetries and perfect metric values, we are looking for similarities, which are hard to find with calculations. As we said before, the way cycles are perceived in a cognitive process, might be subject to the chartist's bias. To keep it simple let us consider the following simple biases:

1- Where to start to look at in the series? (Starting point)
2- How far to walk before we stop and say this is a cycle? Then start again if finding the next?

It 's certainly almost impossible to answer the above questions, as we have no exact definition of what a cycle is "numerically" in the financial time series, but we know from fractals, that "similar" structures are what we are looking for. Now suppose we have a time series X={X1,X2,..,X12}, would not one way of looking at that data be as follows:

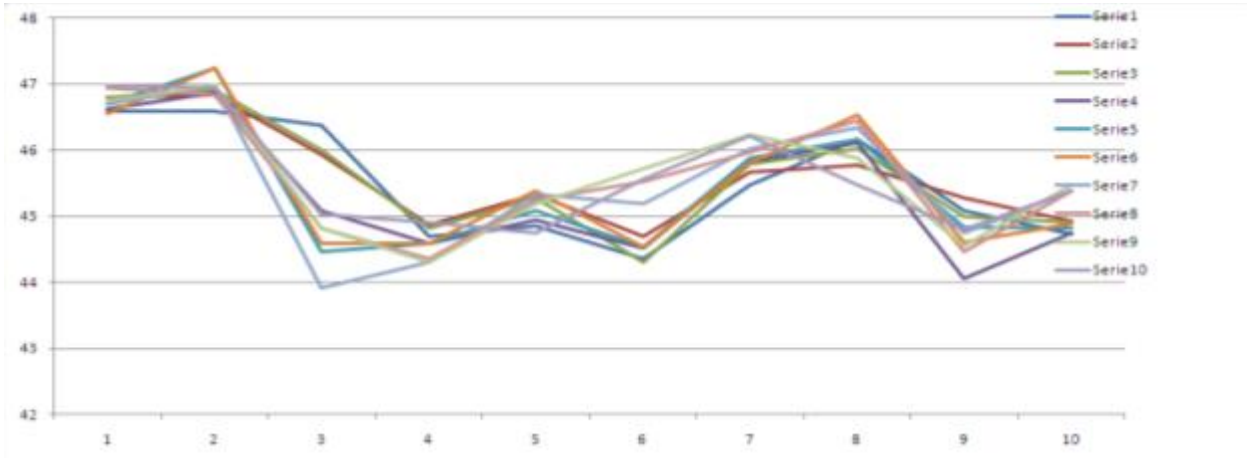
X1,X4,X7,X10
X2,X5,X8,X11
X3,X6,X9,X12

Actually, if we look at the above data, row by row, it will have the following visual characteristics:

1- Each row is similar to its' neighbor
2- Each row is similar to the whole shape of the observed series

To observe these characteristics, let us see the behavior of the first 100 data points from the above S&P500 chart when seen as explained above, with the selection of 10 features per row, yielding 10 rows (in Takens' theorem, D=10 and s=10).

0.2: FIRST OBSERVED 100 DATA POINTS IN S&P500



0.3:VISUAL RESULT OF THE FEATURES POSITIONED AS EXPLAINED

So far, what we have done above is nothing but intuitive, and the question is how to model it into a machine process. As in every model, we find ourselves obliged to justify this selection of features. From a process point of view, there must be something that nails better these features numerically. So, let us assume that there is a hidden process. The process we are looking for is a process where changes in its' initial conditions would yield chaos(and thus the series), a process that would be able to represent both, random and deterministic series, all under similar forms. To create such a process, let us start from the time series itself and do the following:

1- Let X={x1,x2,..,xn} be our time series
2- Let Y={y1,y2,…,yn} be a series generated by the cumulative sum of the observations in series X
3- Let Z={z1,z2,…,zn} be a series generated by the cumulative sum of the observations in series Y
4- Let L={l1,l2,…,ln} be a series generated by the natural logarithm of the observations in series Z
5- Let S={s1,s2,…,sn} be a series generated by the cumulative sum of the observations in series L

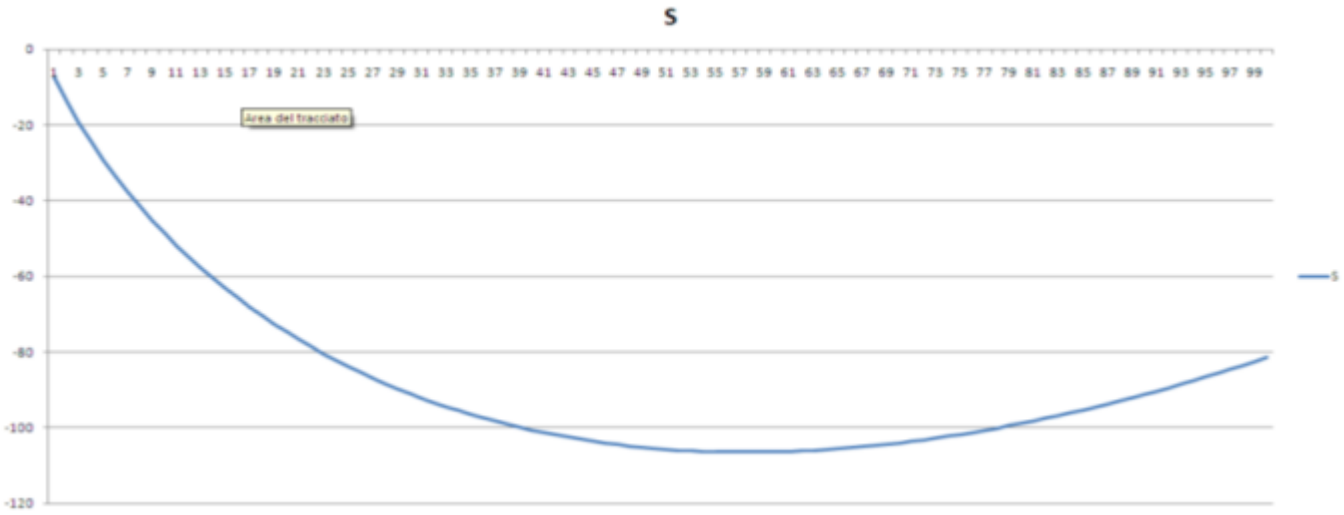
Applying the above on the first 100 data points of the S&P500 index, the S series chart looks as follows(100PointsExampleCalculation)

0.4: S CHART FOR FIRST 100 DATA POINTS

The above chart seems to have very few to say, however, if we add some changes to the above process as follows:

1- Let X={x1,x2,..,xn} be our time series
2- Let Y={y1,y2,…,yn} be a series generated by the cumulative sum of the observations in series X
3- Let Z={z1,z2,…,zn} be a series generated by the cumulative sum of the observations in series Y
4- Let M={m1,m2,…,mn} be a series generated by the ratio of the values in Z to the Mean of the Z series: m1=z1/Mean(Z),….,mn=zn/Mean(Z)
5- Let L={l1,l2,…,ln} be a series generated by the natural logarithm of the observations in series M
6- Let S={s1,s2,…,sn} be a series generated by the cumulative sum of the observations in series L

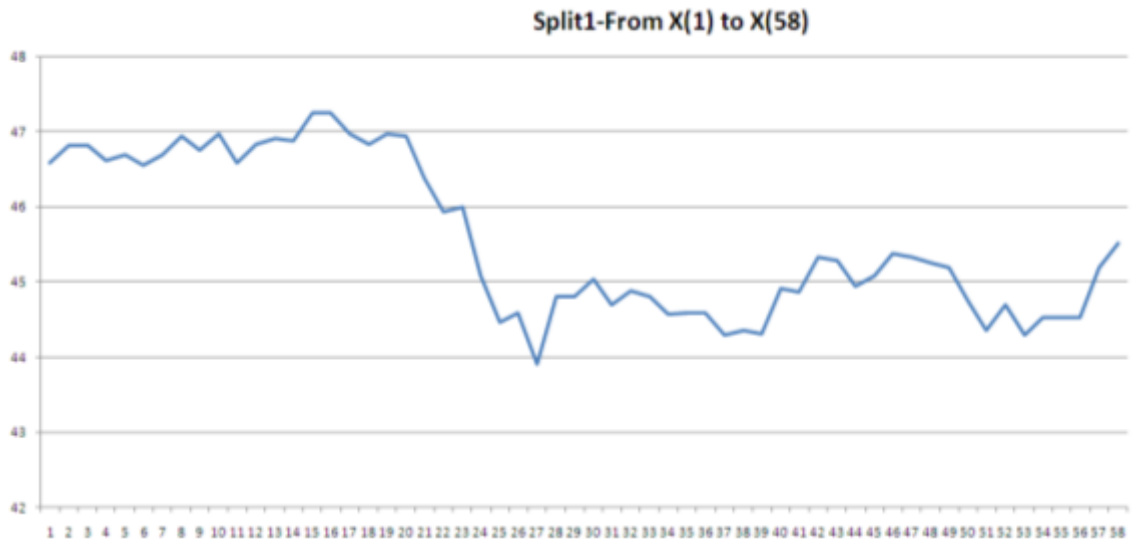
Applying the above, the S chart looks as follows:



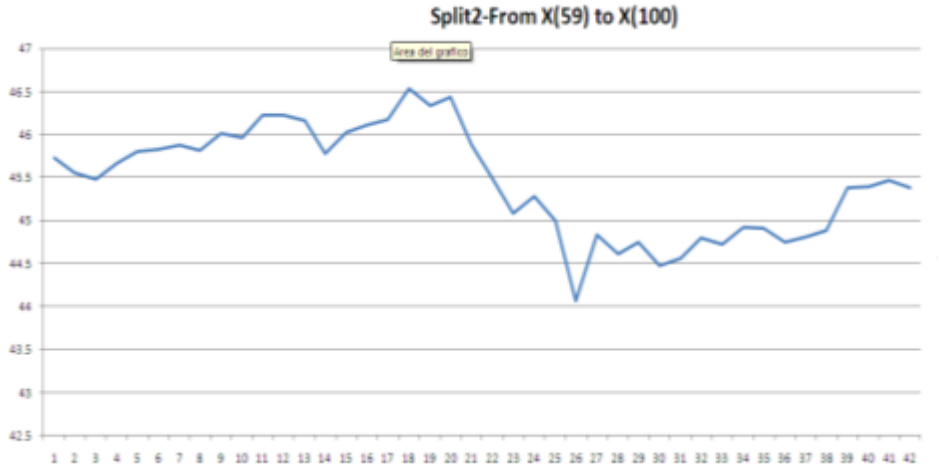
0.5: AVERAGED S CHART

It is intuitive that our curiosity would mostly be due to the minimum of the above chart. At a glance, it looks like some values are being repeated once the minimum is reached in S. By the nature of the process we modeled, those values can't be repeating and actually they are not. In the above S chart the minimum is:
Min(S) = -106.4718057 Coinciding with data point 58 in the S&P500 series: X(58).

Let us now split the S&P500(first 100 data points) exactly at point 58 into 2 splits and observed the following:



0.6: SPLIT 1 OF FIRST 100 DATA POINTS, FROM 1 TO 58



0.7: SPLIT 1 OF FIRST 100 DATA POINTS, FROM 59 TO 100

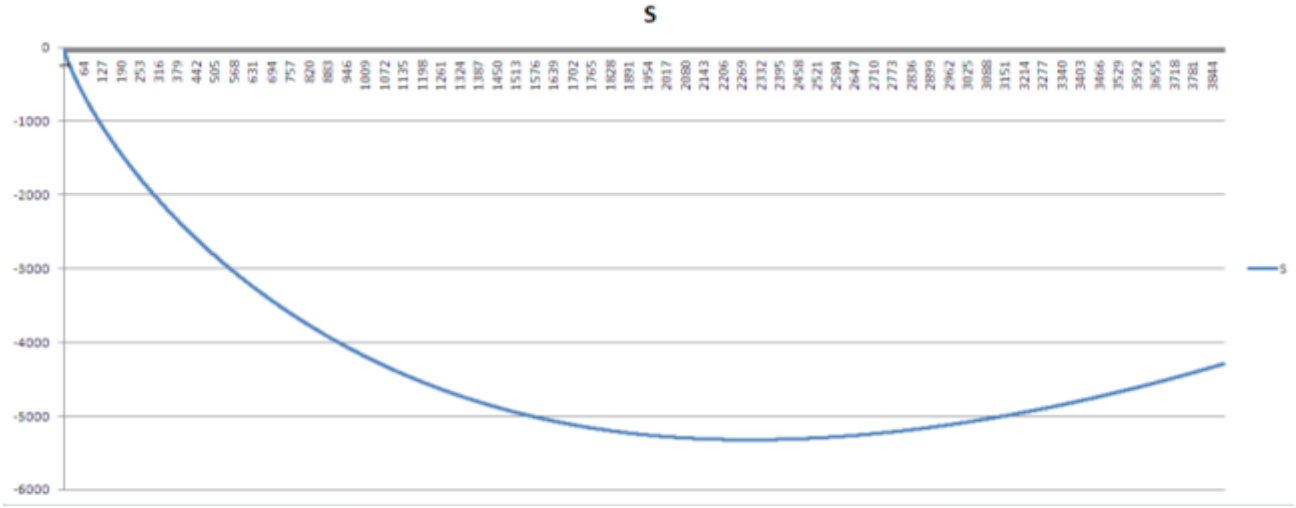
Then, here is a chart comparing both, the above splits:

0.8: COMPARISON OF THE 2 SPLITS

Both cycles look similar actually, not equal, not symmetrical as formally we have learnt to find them, however, only similar, as Chaos Theory postulates. To the chart in Split2(the red in the above chart) let us add future values, so that the number of data points in Split2 would equate those in Split1, and here are the results:

0.9: COMPARISON OF THE 2 SPLITS, WITH FUTURE DATA ADDED

Trying the same on all the 3887 data points of S&P500 index I got the following 2 splits:



0.10: S CHART OF THE FULL DATA FOR S&P500

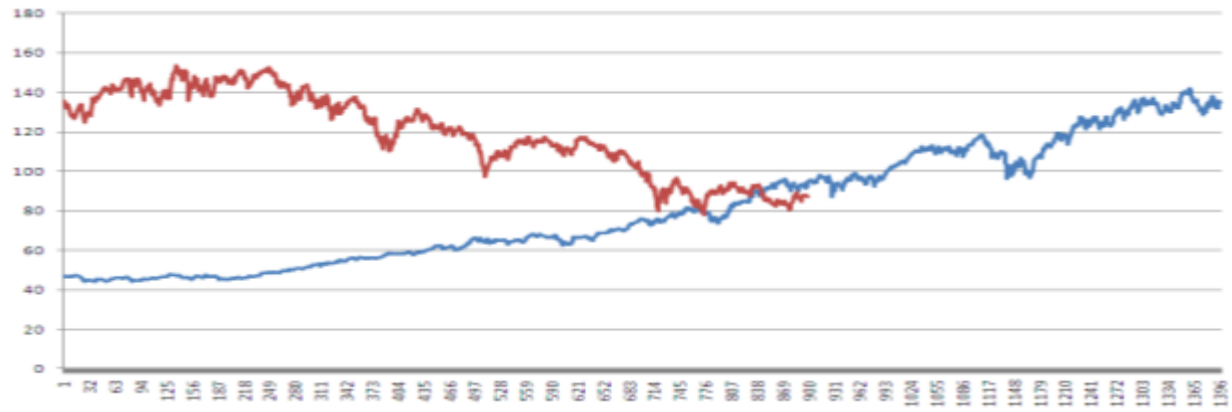
Min(S) = -5323.226284 coinciding with data point 2298 of the S&P500 series, from where i extracted 2 splits again, and the result is as follows:

0.11: THE 2 SPLITS FOR S&P500 FULL DATA

It seems that it split the chart in those two parts that at the beginning captured our eyes. Again, we repeat the same procedure for both splits in the above chart, consider the first split "Split-1":

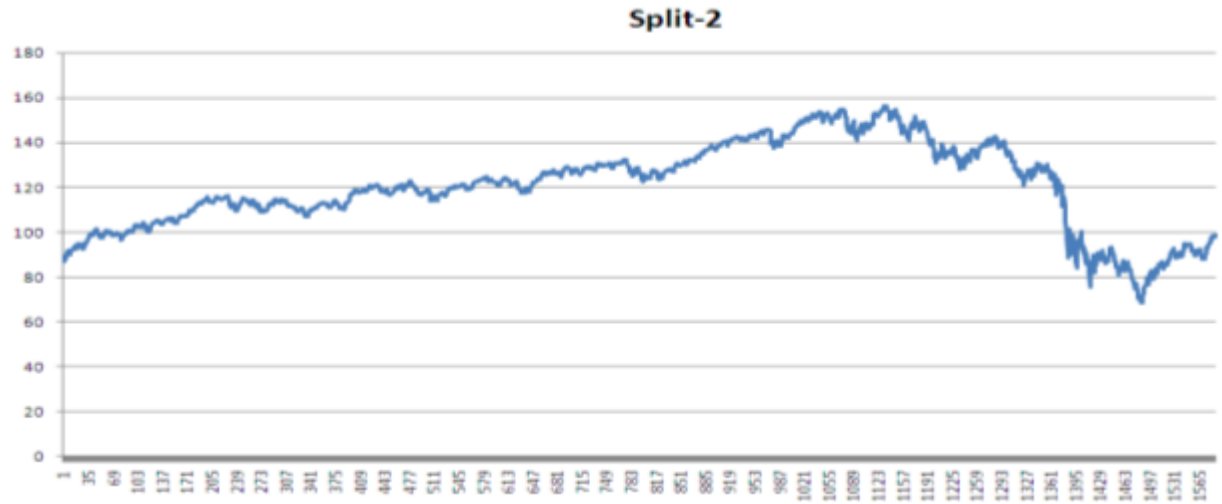
0.12: SPLIT 1 FOR S&P500

When subject to the previously described procedure, the resulting two sub-splits are as follows:



0.13: S&P500 SUB-SPLITS FOR SPLIT 1

Again, the second sub-split, subject to the procedure results the following:



0.14: SPLIT 2 FOR S&P500

0.15: S&P500 SUB-SPLITS FOR SPLIT 2

As one can see, in both cases in the above, "Split-1" and "Split-2" were divided in their two major trends. And thus, the minimum point of the S chart suggested, becomes a good candidate for our quest where we assume that a chartist embed the price time series in some way, where he observes similarities.

1.10. Methodology

As mentioned before, every chartists, based on his expectations, trading horizons and maybe other factors related to his cognitive process, might chose his own embedding dimension. For this purpose, we shall not focus on the best empirically calculated dimension, since, as said before, there is no clean cut theory that guides us on why a dimension is better than the other, we only have empirical methods that make assumptions. To proceed, we shall first decide how many features we

want our embedded vector to have, again such a decision is subjective to the chartist, in the real life scenario. In our experiments, we will assume 5 to 7 features.

Keeping in mind the following procedure suggested earlier:

- 1- Let X={x1,x2,..,xn} be our time series
- 2- Let Y={y1,y2,…,yn} be a series generated by the cumulative sum of the observations in series X
- 3- Let Z={z1,z2,…,zn} be a series generated by the cumulative sum of the observations in series Y
- 4- Let M={m1,m2,…,mn} be a series generated by the ratio of the values in Z to the Mean of the Z series: m1=z1/Mean(Z),….,mn=zn/Mean(Z)
- 5- Let L={l1,l2,…,ln} be a series generated by the natural logarithm of the observations in series M
- 6- Let S={s1,s2,…,sn} be a series generated by the cumulative sum of the observations in series L

The model we suggest in this work is best described as follows:

1- Decide arbitrarily on a number of features.
2- Based on the number of features needed chose a number of data points in the series for which the average will be used in step 4 in the above procedure.
3- Consider the minimum point of the S chart as the embedding dimension
4- Consider the separation as the number of data points from the beginning till the minimum point of the S chart.
5- Build C columns where each row is an embed vector.
6- Let T be the number of elements in each vector
7- Number of features = T-1 where the first elements of the vector are the features and the last element is the value to be predicted.
8- Classify the features with an unsupervised learning method.
9- For each class, train a multi layer Perceptron to predict the result.

More formally, consider a time series X={x1,x2,..,xn}, of total length n. Let m be the minimum point calculated by the S chart.

Then the number of embedded vectors would be: m

The number of elements (E) in each embedded vector, would be:

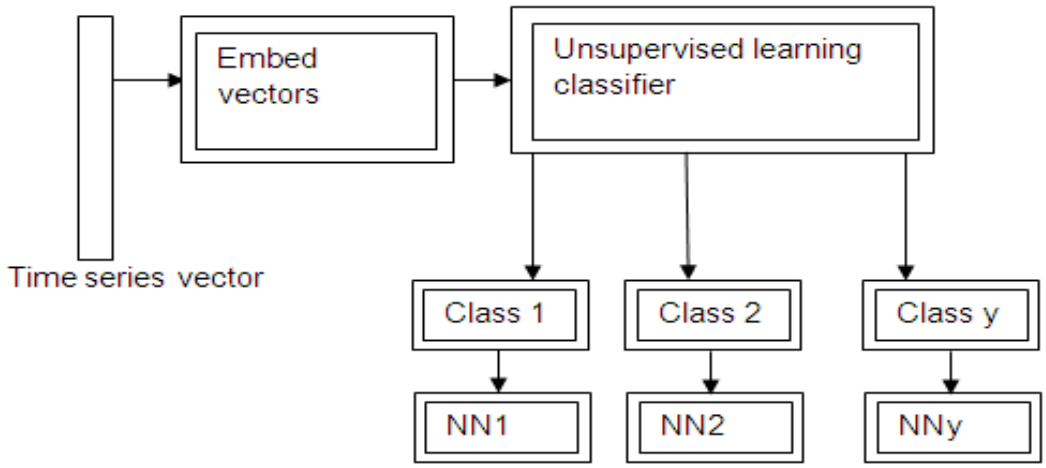
E=Ceiling(n/m)

And thus, the number of features F=E-1.

The last element, for some embedding dimensions and for the last vectors would result missing(which are the values we will try to predict), and so, the number of vectors containing all the elements(features and result) would be:

T = n-m*(E-1) = n-m*F

From the above, we have m vectors to be classified using an unsupervised method such as Fuzzy-ART or K-Means. For each class, we have a set of vectors with F features each and one result, for which a back propagation network shall be trained.



0.16: THE PREDICTIVE MODEL - BLOCK DIAGRAM

1.11. Experiments

To perform the experiments in this section, various financial time series of NYSE indexes were downloaded from yahoo finance, of which we shall show S&P500. The data analyzed is the daily close price for various indexes. Whenever a dataset is used with an ANN, the data is rescaled to have zero mean and unit variance. This is done by subtracting the mean from the data then dividing the result by the standard deviation, sometimes an alternative is used for convenience, which is simply dividing the result by the maximum of the absolute value of the data with mean zero. In experiments dealing with the Hurst exponent, the differences of the data is used. Most of the experiments in this work were focused on the S&P500 index, which visually exhibits cycles and self similar patterns.

1.11.1. **Hurst exponent : Monte Carlo simulation**

For most time series used in this work, the Hurst exponent was calculated for various data segments. As mentioned before, the Hurst exponent provides a measurement of the predictability of a series. The exponent is calculated using the Rescaled Range Analysis previously described. For a random time series, the expected value of the (R/S) was calculated by Feller [4], Anis and Lloyd[ref] then lately corrected empirically by Peters[4]. There are differences between the results each of the above calculation provide. So, in order to calculate the expected Hurst exponent, we have used Monte Carlo simulation as in [24] in order to have some confidence interval in the results. Again, like in [24] whenever we were in from of a time series of length n, we would generate 10.000 Gaussian random series of size n, we calculate H for each series, average the result, after

repeating the process 10 times. The average is expected to approximate the true value of H. Thus, for each experiment, we have calculated the Hurst Exponent for the portion of the data we are interested in, based on the above explained methodology using Monte Carlo simulation. We were however not lucky in finding suitable portions of our data with a significant Hurst Exponent to suggest a non-random process. The reason for that is probably the sensitivity of the Hurst Exponent to long term memory processes. Since we are interested in recent and short data of our series in order to perform practically useful predictions, we were out of luck with this approach. However, for the purpose of research, we include in Appendix A the MATLAB code we have developed for the (R/S) analysis and the Hurst exponent as well as the Monte Carlo simulation code.

1.11.2. **Mackey-Glass equation**

The Mackey-Glass(1977) equation was developed to model the red blood cell production. It states that:

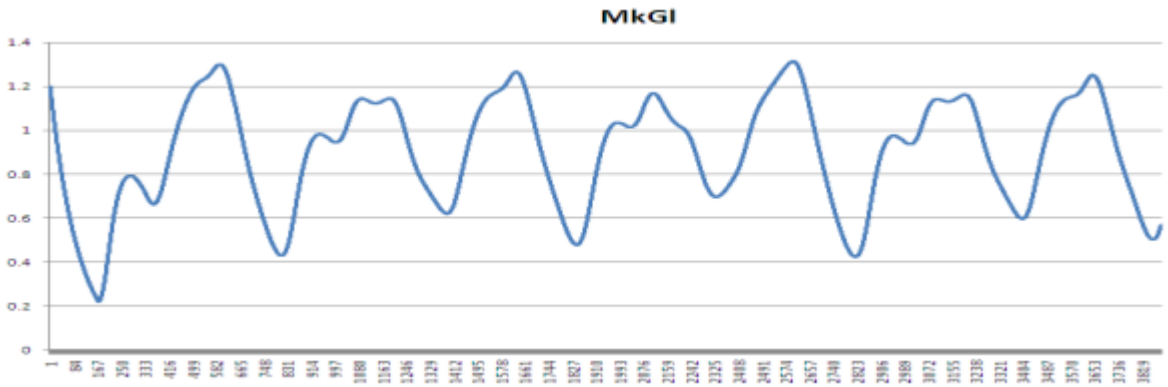
- The current production based on past productions and current measurements.
- A "Cycle" is related to the delay between production and the current levels.
- The system not being linear, overproduction and underproduction tend to be amplified, resulting in non-periodic cycles.
- It is a delay differential equation, so it has infinite degree of freedom much like the markets.

The above made it a good candidate for simulation in many studies referring to financial time series[4].

In the following, 80.000 values for the equation were generated using MATLAB. The first 3887 values were taken to equate the S&P500 data points we have.

The previously described process for embedding using the minimum point of th S chart was applied to the series after some modifications to the series in order to make it look like a price series:

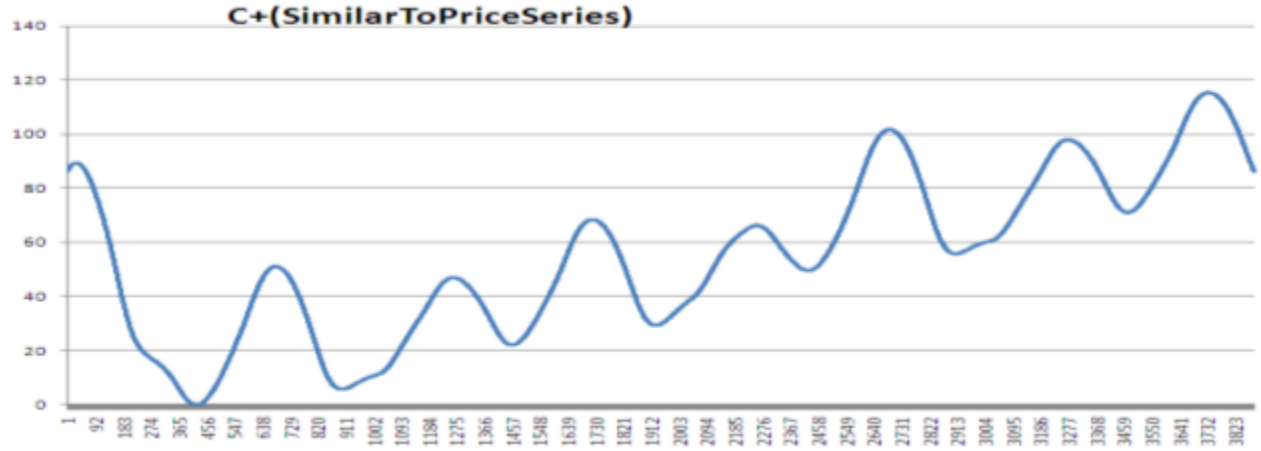
1- We select the first 3887 values of the Mackey-Glass equation



0.17: MACKEY-GLASS EQUATION: FIRST 3887 POINTS

2- The series is divide by the maximum value in the series to obtain a series with unit variance.
3- Subtract the mean: from the series, we subtracted the mean of the data from to obtain ZERO mean, so now the series has Zero mean and Unit variance now.
4- Here, the cumulative sum of the standardized result is calculated, since the Mackey-Glass is a differences equation, we want the cumulative sum in order to have a time series similar to the price series(price series being the cumulative sum of differences in price).
5- To eliminate negative values, w added to the series, the Absolute value of the Minimum.

Now the series looks like a price series as follows:



0.18: MACKEY-GLASS, SIMILAR TO PRICE SERIES

To obtain the minimum value of the S chart, the average of the first 1020 points was used, this is to extract more than 2 features (this division will give 6 features actually plus the 7^{th} as the result). The MIN of the S chart is calculated to be: -902.8418881 coinciding with data point 562 in the series.

Based on the above value, we partitioned the data of the series in 6 columns containing 562 data point each(c1,c2,c3,c4,c6) plus one column c7 containing the remaining 515 points of the series.

Before trying both models into a back propagation ANN, we have tested the models under simple linear regression:

By assuming an auto regression, we are considering that a value in c7 at time index (t) is correlated to the past data in c1,c2,c3,c4,c6 at time (t) of each partition. That is:

$c7(t) = a1*c1(t) + a2*c2(t) + a3*c3(t) + a4*c4(t) + a5*c5(t) + a6*c6(t) + e(t)$ where

a1,a2,a3,a4,a5,a6 are the parameters of the model (or weights in case of MLP)

e(t) is a random number (iid, independently identically distributed, sampled from a normal distribution)

The above assumes an autoregressive process occurring at these rows. The model summary and performance measurements are as follows:

Linear Regression Model: (70% of data was used to generate the model, the rest is test)

c7 = -0.4945 *c1 +-0.7895 * c3 + 0.4756*c4+-0.1832*c5+0.3719*c6+104.3763

Correlation coefficient	0.9998
Mean absolute error	0.2127
Root mean squared error	0.2797
Relative absolute error	1.6695 %
Root relative squared error	1.8962 %
Total Number of Instances	362

TABLE 0.1:LINEAR REGRESSION FOR EMBDDED VECTORS OF MACKEY-GLASS

Autoregressive AR(4) model:

The AR(4) model assumes that the current value depends on the weighed sum of the previous 4 values. The following are the model and the performance results:

Linear Regression Model (70% of data to build the model the rest is test):

c(t) = 0.9973 *c(t-4) + 0.083

Correlation coefficient	0.9993
Mean absolute error	0.6499
Root mean squared error	0.7671
Relative absolute error	3.6781 %
Root relative squared error:	3.6146 %
Total Number of Instances	2556

TABLE 0.2: LINEAR REGRESSION FOR MACKEY-GLASS AR(4) MODEL

Back propagation ANN:

In what comes next, we will try to predict the above series with a back propagation neural network assuming 3 different processes:

1- Autoregressive, AR(4) process
2- Using all embedded vectors

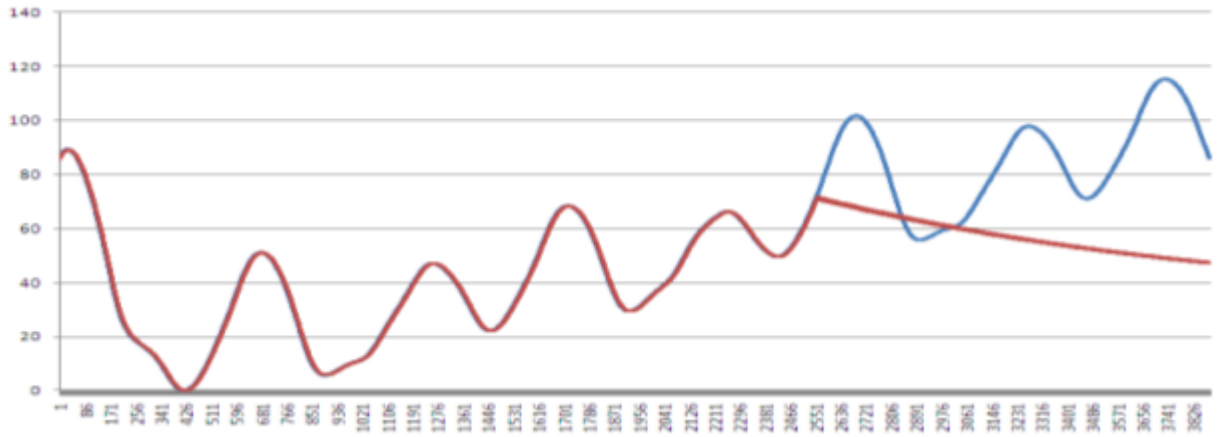
Measurement	AR(4) 1 step ahead	Embedded vectors 1 step ahead
RMSE	0.022	0.0822
Percentage split; Test-Train	70%	70%
Instances	2556	362

TABLE 0.3: RMSE COMPARISON, BACK PROPAGATION ANN, AR(4) PROCESS ASSUMED VS EMBEDDED VECTORS - MACKEY-GLASS

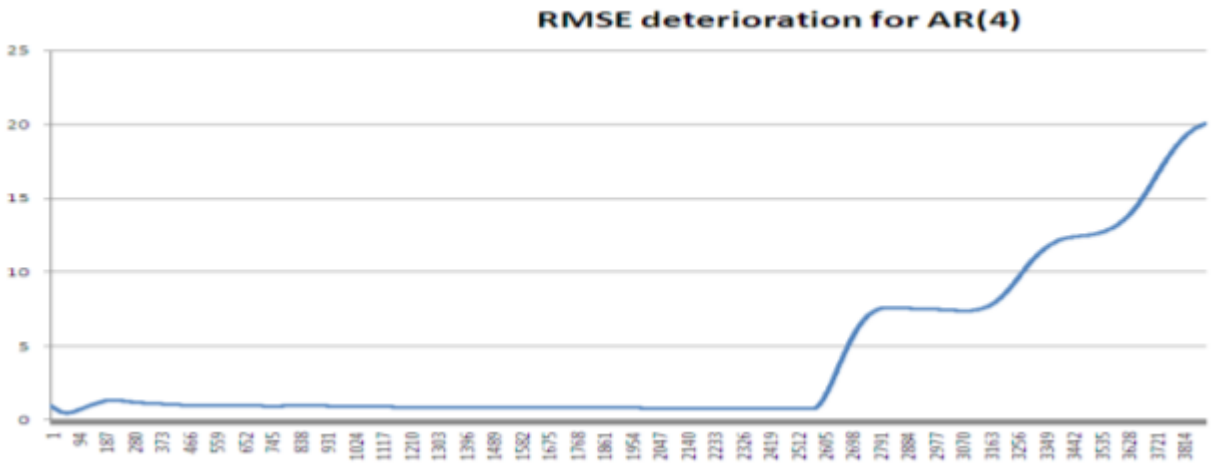
At first glance, the above results for one step ahead predictions, favor the AR(4) model. However, when attempting to use the AR(4) model output as input again in order to attempt further steps prediction, the RMSE deteriorates as follows for the AR(4) while it deteriorates less for the embedded vectors:

Measurement	AR(4) 15 steps ahead	Embedded vectors 15 steps ahead
RMSE	0.8	0.144
Percentage split; Test-Train	70%	70%
Instances	2556	362

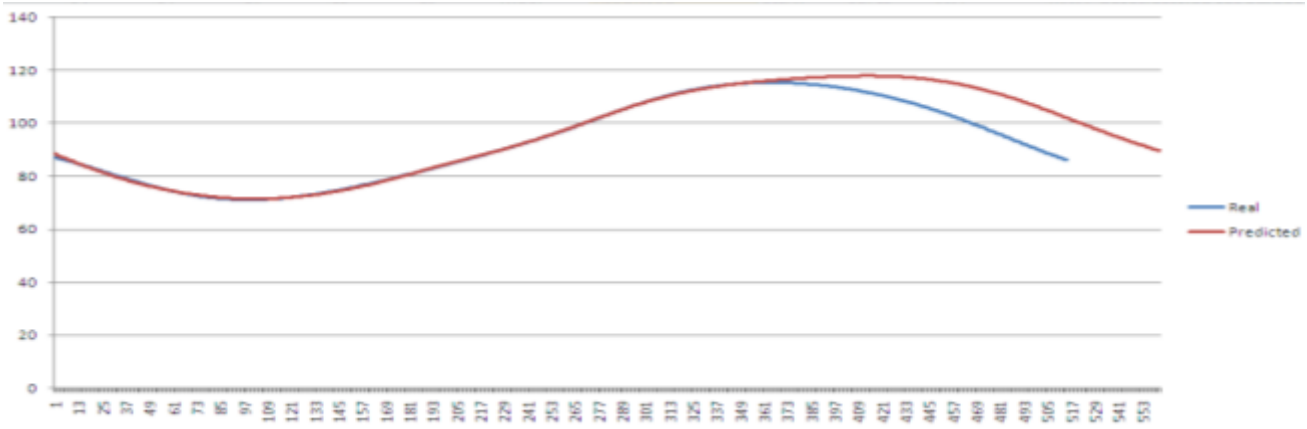
TABLE 0.4: RMSE COMPARISON, BACK PROPAGATION ANN, AR(4) PROCESS ASSUMED VS EMBEDDED VECTORS - MACKEY-GLASS - 15 STEPS AHEAD



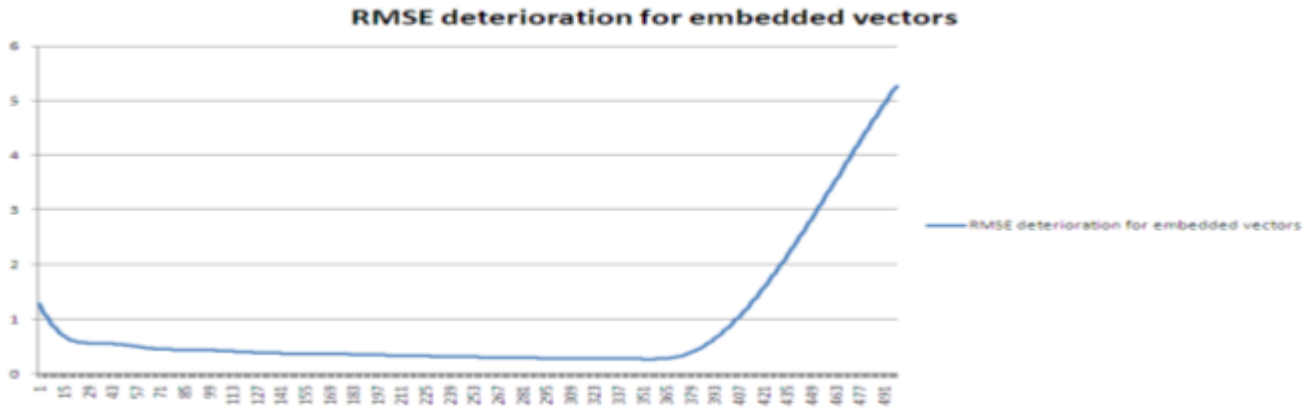
0.19: MACKEY-GLASS: AR(4) PREDICTIONS



0.20: RMSE DETERIORATION FOR AR(4) MODEL



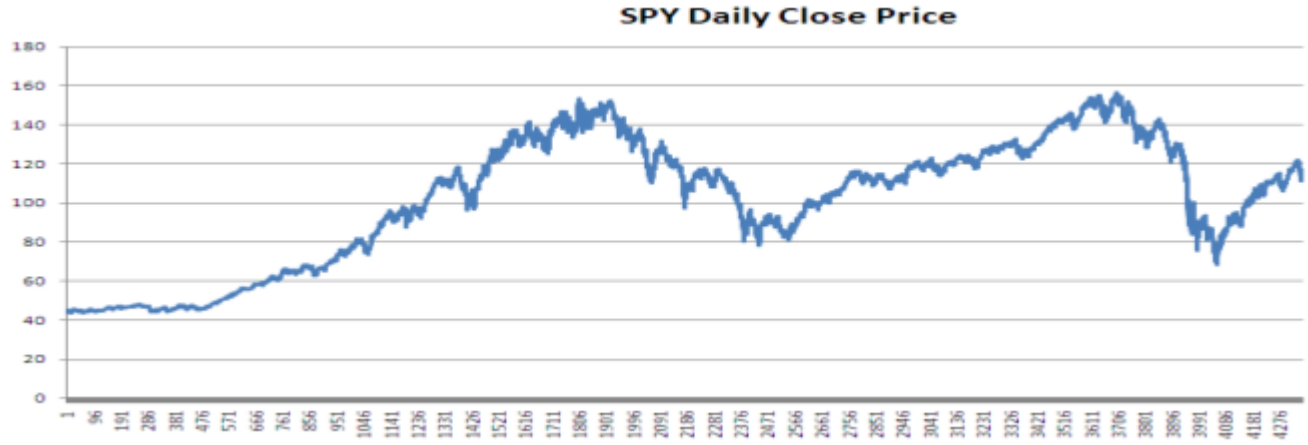
0.21: MACKEY-GLASS, EMBEDDED VECTORS PREDICTIONS



0.22: RMSE DETERIORATION FOR MACKEY-GLASS, EMBEDDED VECTORS MODEL

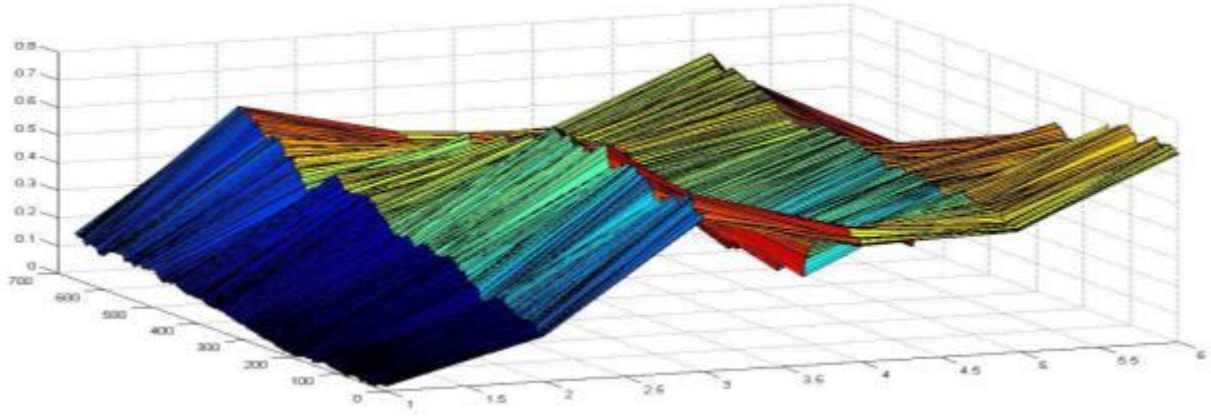
1.11.3.**S&P 500**

In the following experiment, we are using the S&P500 time series for daily close price since 29/01/1993 till 14/05/2010 downloaded from Yahoo Finance, SPY index.



0.23: S&P500, DAILY CLOSE PRICE SINCE 29/01/1993 TILL 14/05/2010

By repeating the same experiment on SPY data we obtain the following embedded vectors, visualized in as a 3D surface:



0.24: 3-D PLOT OF THE EMBEDDED VECTORS FOR S&P500

Measurement	AR(4) 1 step ahead	Embedded vectors multiple steps ahead
RMSE	1.91	4.94
Percentage split; Test-Train	70%	70%
Instances	451	457

TABLE 0.5: RMSE COMPARISON, BACK PROPAGATION ANN, AR(4) PROCESS ASSUMED VS EMBEDDED VECTORS - S&P500

At this stage, we implement our steps of the suggested model as follows:

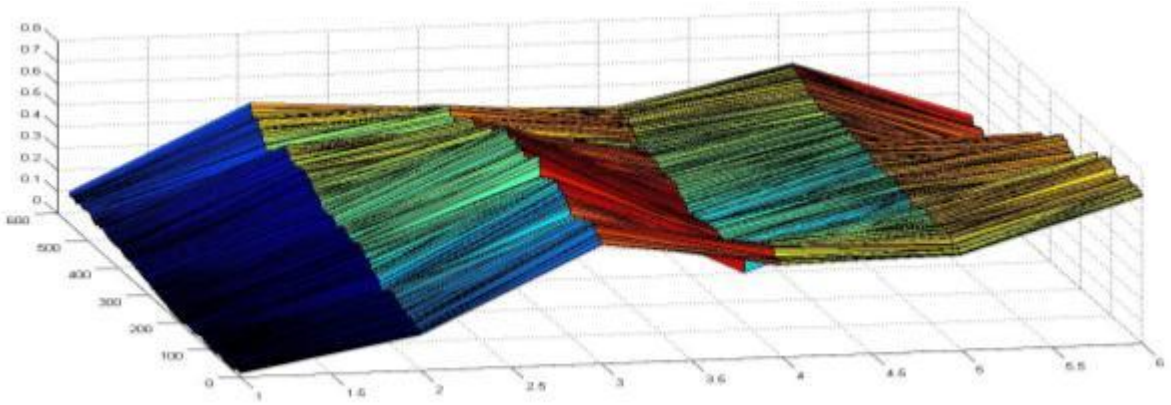
1- Classify the embedded vectors of SPY with Fuzzy ART
2- For each class train a back propagation Neural Network

Data preparation for Fuzzy ART:

The complement code implemented by Fuzzy ART requires the data to be standardized, so as a first step, we rescale SPY in the interval]0,1[

0.25: RESCALED S&P500

After embedding the above, using the S chart procedure averaged at 1020 data points, we classify the embedded vectors with Fuzzy-ART and obtain 2 classes. In our case, class 1 had enough data in the last elements of the embedded vectors in order to train an ANN, while with class 2 we were unlucky, and had no enough data for a network, so we proceed with training a Back Propagation Network for the vectors classified under category 1 and the result is as follows:



0.26: 3-D PLOT OF S&P500 VECTORS OF ART CLASS 1

Measurement	AR(4) 1 step ahead	Embedded vectors multiple steps ahead	**Classified Embedded Vectors, Class 1**
RMSE	1.91	4.94	**0.0258**
Percentage split; Test-Train	70%	70%	**70%**
Instances	451	457	**168**

TABLE 0.6: RMSE COMPARISON, BACK PROPAGATION ANN, AR(4) PROCESS ASSUMED VS EMBEDDED VECTORS - PROPOSED MODEL

As can be noted from the above table, the RMSE performance was enhanced, and the surface of the embedded vectors is smoother.

1.11.4.**Chaos Game based visual observations**

Random Data Set: Consider the procedure previously described for producing the Sierpinski triangle in the Chaos Game, and the following initial labeled triangle A(1,2), B(3,4) and C(5,6) :

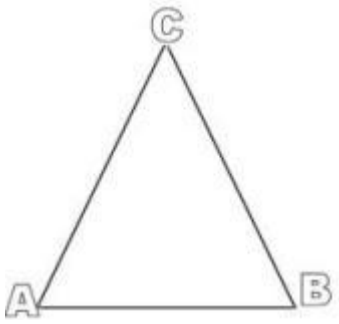


FIGURE 0.27: INITIAL LABELING FOR THE TRIANGE IFOR THE GAME OF CHAOS

Suppose we have a random series X=(x1,x2,…,xn) where xi is an independent and identically distributed random variable in the interval]0,1[. If xi<1/3 then the fair dice's result is either one or two and our vertex is A, similarly, if xi>2/3 then our vertex is C, otherwise our vertex is B. Such a procedure applied on the data of a random time series of 3000 points resulted in the following:

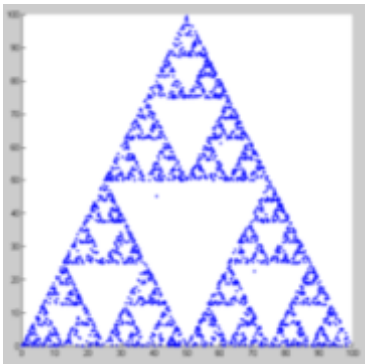
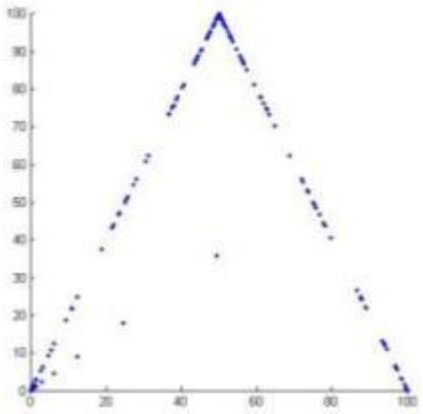


FIGURE 0.28: SIERPINSKI TRIANGLE GENERATED WITH 3000 I.I.D. RANDOM NUMBERS

S&P500 rescaled daily close price: Now, instead of generating i.i.d. random variables, we introduce to the chaos game the S&P500 time series for daily close price since 29/01/1993 till 14/05/2010, after rescaling the data in the interval]0,1[

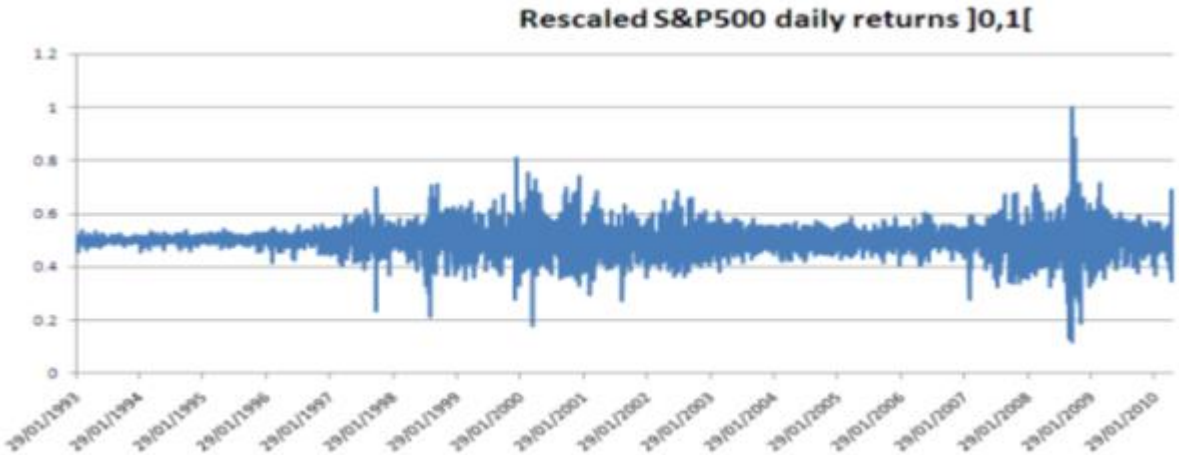
FIGURE 0.29: RESCALED CLOSE PRICE FOR S&P500

The result of the above is the following:

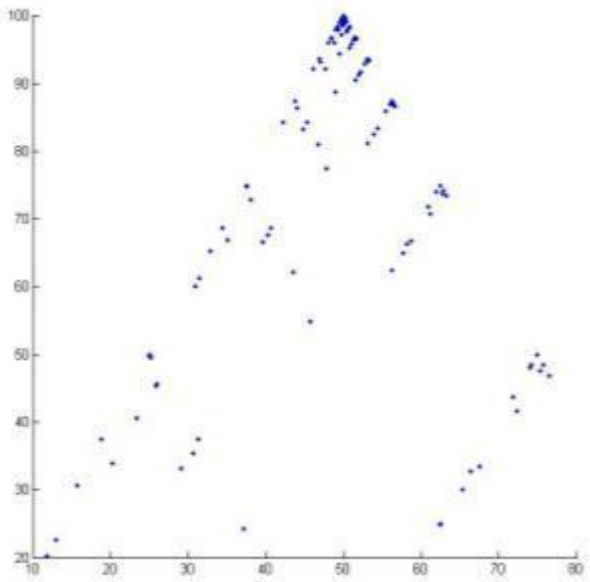


0.30: SIERPINSKI TRIANGLE FOR S&P500 CLOSE PRICE DATA

Daily returns of S&P500 rescaled: Now we introduce to the chaos game, the daily returns of the previously used S&P500 close price series, rescaled again in the range]0,1[.



0.31: RESCALED DAILY RETURNS FOR S&P500



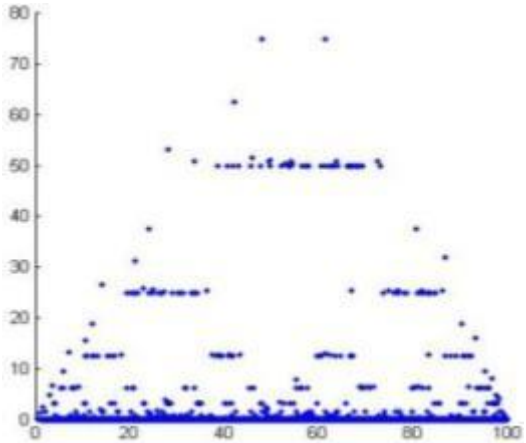
0.32: SIERPINSKI TRIANGLE FOR S&P500 DAILY RETURNS

The literature states that 10000 points are needed to construct the Sierpinski triangle, for which the Fractal Dimension D=log(3)/log(2) = 1.585. In a quantitative randomness test using the chaos game, the fractal dimension of the obtained triangle should be compared to the actual D of the full Sierpinski triangle. In this experiment which is meant to be only a warming-up in this work, we analysis qualitative and thus limited to visual observations.

S&P500 direction: In this last experiment using the chaos game, we shall transform the S&P500 (X) series in hand into a -1, +1 and 0 series (Y), in the following manner:

- if X(t+1) > X(t) then Y(t) = +1 and the vertex is A
- If X(t+1) < X(t) then Y(t) = -1 and the vertex is B
- If X(t+1)=X(t) then Y(t) = 0 and the vertex is C

The purpose of this transformation is simply to test for randomness in the direction of the index, in this manner, we have three states corresponding respectively to the three vertices of the triangle.



0.33: SIERPINSKI TRIANGLE FOR S&P500 DAILY RETURNS DIRECTION

As can be visually noted from the above, the incomplete Sierpinski triangles produced by the data, shows a weak random process when compared with a Gaussian random time series.

Analysis

From the above experiments, we can note that embedding is a powerful tool when analyzing time series. There is no clean cut theory to define the criteria for a good embedding scheme. This fact leaves some questions open for discussion: Is there one or more embedding scheme or classes of schemes, better than others? What if all embeddings are correct, and they are only expectation oriented. The embedding provided in this work, by the S chart, is strictly empirical, and more than other methods, lacks theoretical justifications. However, it provides a simple visual tool, to tell, how much, visually, portions of the data are similar. This is somehow analogous to what a chartist look at in the data. While doing this work, we have tried to find some characteristic exponent that measures the degree of similarity in the data. One idea to calculate such a measure was the following:

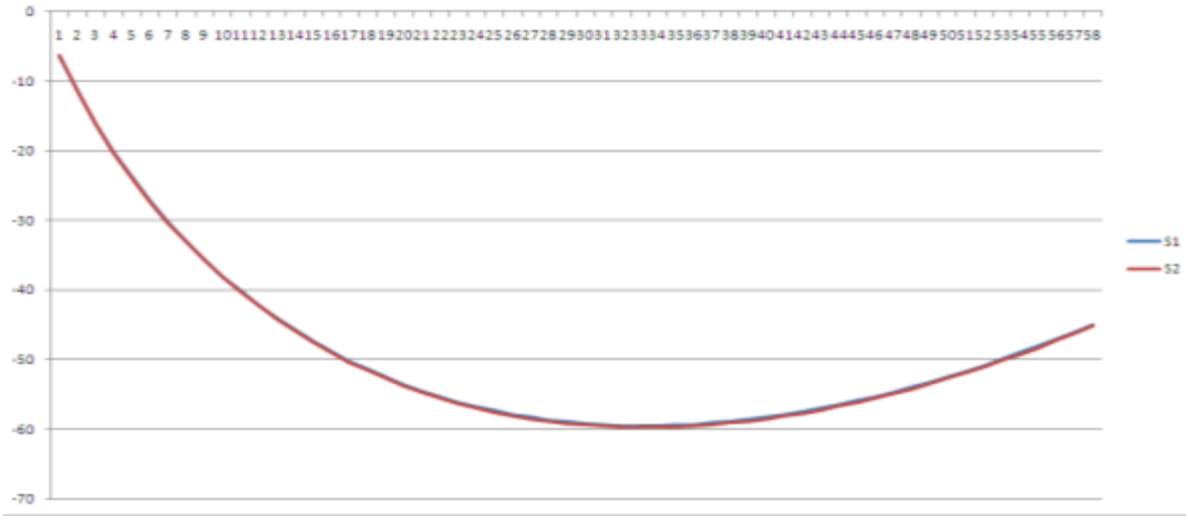
1- We calculate the Minimum of the S chart of a gives Series of length n
2- We then calculate respectively the S1 and S2 chart data for each split given by the Min(S)
3- Consider ln|S2|=k*ln|S1|
4- Then k is the slope of the log/log plot of S1, S2.

If the 2 splits are perfectly repeating cycles, then S1 = S2 and k=1, for series where S1<>S2 the deviation of k from 1 would measure the "degree of similarity". Such an approach would suggest that the predictability of a series, even under the random walk hypothesis becomes higher as k approaches 1.

Let us consider the first 100 points of S&P500 and the two splits provided by Min(S), previously calculated:

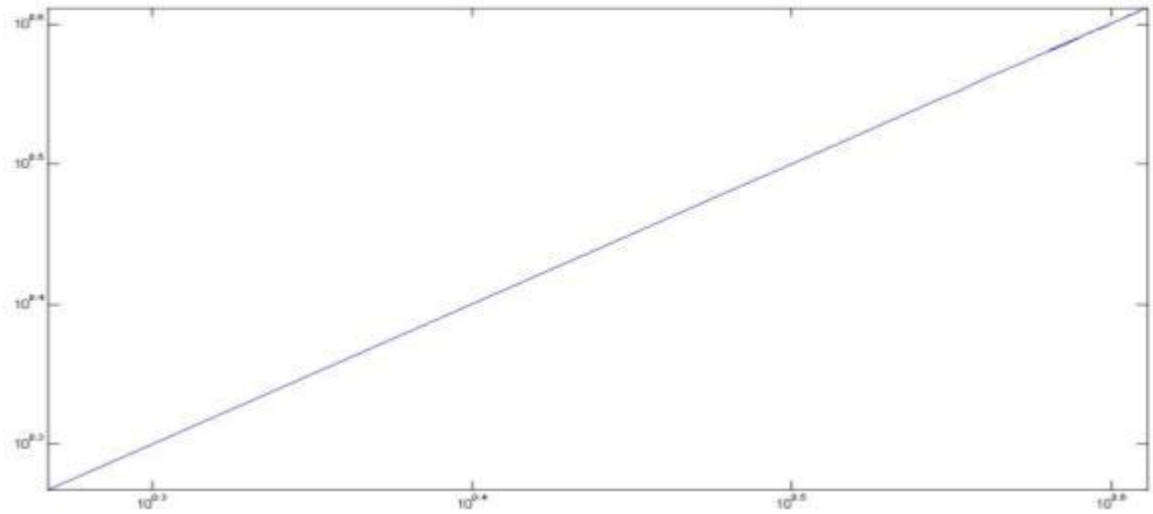
0.1: FIRST 100 POINTS FOR S&P500, 2 SPLITS COMPARED

And the comparison of their respective S chart:



0.2: S CHARTS COMPARISON OF S&P500 2 SPLITS FOR 100 POINTS

The log/log plot of S1 to S2 looks as follows:

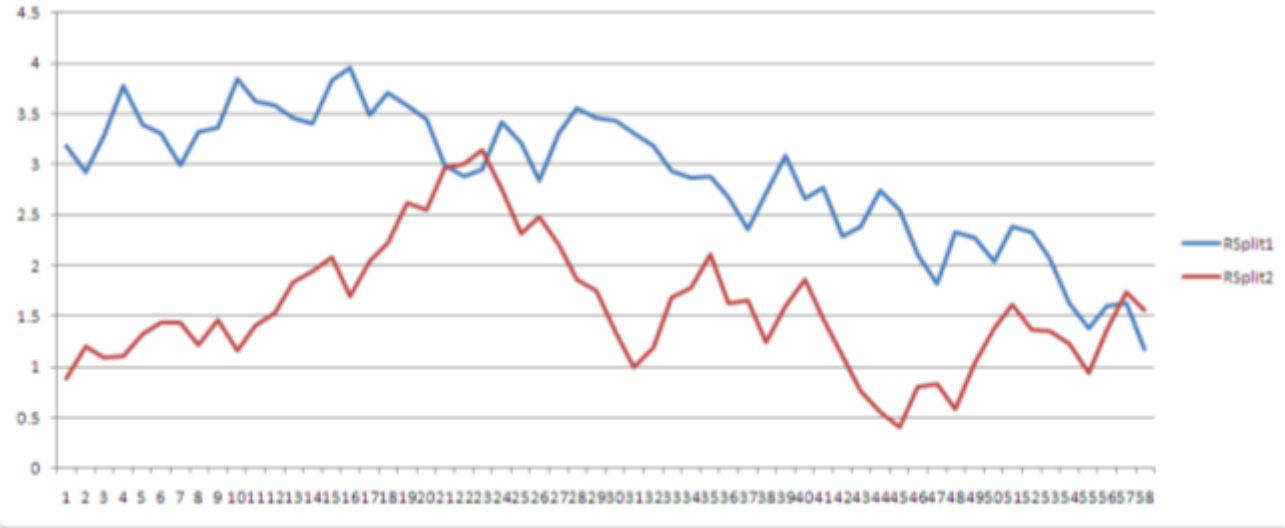


0.3: LOG/LOG PLOT FOR S CHARTS OF THE 2 SPLITS FOR S&P500 FIRST 100 POINTS

By regressing ln(S2) on ln(S1) the slope k= 1.001697669

Ds= |1-k| = 0.001698 where Ds is suggested to reflect the "degree of similarity"

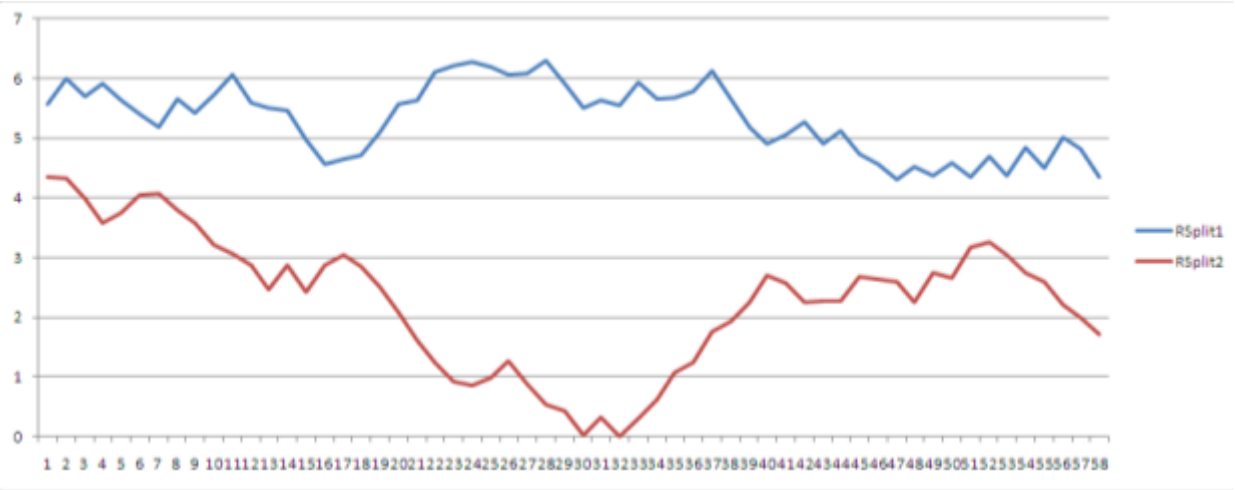
To observe a visual significance of the slope k, we calculated k for many randomly generated series and observed the following:



0.4: RANDOM TIME SERIES-1; 2 SPLITS FIRST 100 POINTS

	K	|1-K|
Random	1.02949987	0.0295
S&P500	1.001697669	0.001698

TABLE 0.1: K EXPONENT FOR RANDOM SERIES 1



0.5: RANDOM TIME SERIES-2; 2 SPLITS FIRST 100 POINTS

	K	|1-K|
Random	0.938069891	0.06193
S&P500	1.001697669	0.001698

TABLE 0.2: K EXPONENT FOR RANDOM SERIES 2

2

0.6: RANDOM TIME SERIES-3; 2 SPLITS FIRST 100 POINTS

	K	|1-K|
Random	0.990682052	0.009318
S&P500	1.001697669	0.001698

TABLE 0.3: K EXPONENT FOR RANDOM SERIES 3

0.7: RANDOM TIME SERIES-3; 2 SPLITS FIRST 100 POINTS

	K	|1-K|
Random	1.000897942	0.000898
S&P500	1.001697669	0.001698

TABLE 0.4: K EXPONENT FOR RANDOM SERIES 4

From the above four sample, one can note that, in the first two sample (Fig numbers) the value of |1-K| was higher than in the next two samples, and consequently, the visual similarity between the splits was more exhibited in the 2nd samples.

Moreover, from our predictive model, we notice that, when the data is split at the point where |K-1| approaches zero, a classification of the embedded vectors would yield better predictions. In this work, only qualitative evidence was provided however, quantitative analysis on the assumptions are left for future work.

Conclusion and future work

1.12. Conclusion

In this work, we have introduced a simple method, that splits a time series in visually similar parts. We were expecting this phenomena to be explained with use of the Hurst exponent. The Hurst exponent for the splits provided by our method, was within the expected value of a random series of the same length. Visually, however, the phenomena keeps being intriguing and so, we have proceeded shown an experiment that uses embedding of a time series, without the use of the available empirical methods. We have shown as well, how the predictive model based on the classification of the embedded vectors, yielded better results.

Using the chaos game, we have observed a weak random process in the S&P500 index, when compared to a randomly generated time series. In the analysis section, we have tried to calculate a measurement that quantifies the degree of similarity in portions of random time series.

The question, whether the markets follow a random walk or not, is still hard to answer. However, there is a visual evidence that even random time series exhibit some self similar properties to a certain degree; as observed in the calculation of k in the analysis section.

A final observation is due. Imagine looking at a pine tree, the left and right branching of the tree are not equal, however, they do exhibit a curious similarity:

1- A tree at first look seems a triangle
2- At each level of leaves, observing the details, the leaves at left and those at right are almost symmetrical, but different, and unique.
3- Is there an exponent, or a process governing the Non-Symmetry but similarity in both sides of the tree? (to extend this is more complex, in the 3-D of the tree?)
4- Would the Hurst exponent be of more value in analyzing "symmetrical branches"
5- Chaos theory may answer some of these questions
6- Now imagine a time series and a method that can divide similarities in it. Is there a Global relation between the growth of the tree and the time series' similarities?
7- Think of the tree top as the first baby-tree, it always looks like a triangle, however, the sides have different shapes in detail. Both sides represent the level of growth of that tree. Observe those 2 sides in one dimension, they will look indeed like embedded data from a time series.
8- Now unfold the levels of the tree to One-Dimension, it will look like a complete time series. Seemingly random.
9- Suppose we expose the resulting time series to Picasso, and then to Jesse Laurinston Livermore, would Picasso embed the series to get a tree? And would Livermore guess the future direction of the series?

As a conclusion, it seems that embedding has more to offer should ones analyze better the context of the system. In the case of a tree, no available parametric method would yield a tree when the

data is embedded, yet, the series is that of tree branching. Which suggests again, local randomness and global determinism, based on some evolution heuristic.

1.13. Limitation

As noted from the experiment on the predictive model, prior to training the Back Propagation ANN, the classification of the embedded vectors might yield classes that lack enough data suitable to train a network. This, in some cases, might result in being able to predict the n^{th} step but not the $(n-x)^{th}$ step.

1.14. Future work

In future work, in order to rationalize the qualitative observations provided here, we hope to perform the following:

1- Design a Monte Carlo simulation, with the hope to statistically analyze the occurrence of values close to zero for |1-k| in randomly generated time series. We hope to provide analysis with v-statistics.
2- From the above experiments and analysis, we have good reasons to believe that the value |1-k| would be suitable to calculate the optimal embedding dimension.
3- During the course of this work, many of the above has been started, but is however, still experimental.
4- Moreover, during the course of this work, a clustering approach, based on the frequency of the features encountered was developed, but is however still experimental as well, and is left for future work.
5- Finally, the next quest, would continue to investigate on the nature of randomness and chaos theory and their relation to the cognitive process.

References

[1]- Aristotle, Politics: BOOK I – PART X
[2]- Peter Forsyth. An introduction to Computational Finance Without Agonizing Pain, 2008
[3]- http://en.wikipedia.org/wiki/Jesse_Lauriston_Livermore , Wikipedia 2010
[4]- Edgar E. Peters. Fractal Market Analysis, Willey & Sons, INC., 1994
[5]- Zoran Vojinovic, Vojislav Kecman and Rainer Seidel, A Data Mining Approach to Financial Time Series Modelling and Forecasting, John Wiley & Sons, Ltd., 2001
[6]- Predrag Cvitanovic et.al, Classical And Quantum Chaos, 2002
[7]- http://en.mimi.hu/stockmarket/chartist.htm
[8]- http://dictionary.bnet.com/definition/chartist.html
[9]- http://en.wikipedia.org/wiki/Efficient-market_hypothesis, Wikipedia 2010
[10]- Paul Cootner, The Random Character of Stock Market Prices, 1964.
[11]- Burton G. Malkiel, Random Walk Down Wall Street, W. W. Norton & Company
[12]- http://en.wikipedia.org/wiki/Brownian_motion, Wikipedia 2010
[13]- http://en.wikipedia.org/wiki/Neural_network, Wikipedia 2010
[14]- Haykin Simon, Neural Networks, A Comprehensive Foundation, Prentice Hall, 1991
[15]- Arbib, Michael A. (Ed.) (1995). The Handbook of Brain Theory and Neural Networks.
[16]- Warren McCulloch and Walter Pitts in , Logical Calculus of Ideas Immanent in Nervous Activity, 1943
[17]- Hebb, Donald O. , The Organization of Behavior. New York: Wiley, 1949
[18]- Carpenter and Grossberg(1987)
[19]- Carpenter,Grossberg, and Rosen, (1991)
[20]- Dr. Hasan Fleyeh, Neural Networks lecture notes, 2009
[21]- Andrew Nicola Edmonds, Time series prediction using supervised learning and tools from chaos theory, 1996 – http://www.scientio.com
[22]- M. B. Kennel, R. Brown, and H. D. I. Abarbanel, Determining embedding dimension for phase-space reconstruction using a geometrical construction, Phys. Rev. A 45, 3403 (1992).
[23]- Temujin Gautama, Danilo P. Mandicy and Marc M. Van Hullez, A DIFFERENTIAL ENTROPY BASED METHOD FOR DETERMINING THE OPTIMAL EMBEDDING PARAMETERS OF A SIGNAL
[24]- Bio Qian, Khaled Rasheed, Hurst Exponent And Financial Market Predictability

Appendix A

1.15. Calculation of the Hurst Exponent, MATLAB CODE

```
function [RSAverage, LevelVectorsArray, H,eH] =
MyHurstExponent(zSeries,MinSegment)
    %n = length(x);
    %mean = sum(x)/n;
    %stdev = sqrt(sum((x-mean).^2/n));
%clear all;
%load SpyClose.mat;
%DataSet
%%%load SpyClose.mat;
%X=data(:,2);
% X(3888,2) = 98.1;
%Hurst Exponent Estimation
% Min Segment Length
%m = 10;
%m=IVector(zE,1);
%m=zE;

X=zSeries;
m=MinSegment;
% Full data signal length
[N,Xc] = size(X);
% Lowest level Vectors' count
lvc = ceil(N/m);
% Total Levels Count
LevelVectorsCount = lvc;
l=1;
%Set Variables for level 1 Containing the whole set
Level = 1;
%The LevelVectorsArray has: Column1: level number
%                           Column2: Vectors Count in level
%                           Column3: Length of each vector
%                           Column4: Total Capacity of Vectors
%                           Column5: Last Vector's Length (due to
%                           fractions)
%                           Column6: Last Vector's Fraction =
%                           LastVectorLenght/LengthPerVector
%                           Column7: = Fractional Vectors Count =
%                           (VectorsCount-1) + LastVector's Fraction

    LevelVectorsArray(Level,1) = Level;
    LevelVectorsArray(Level,2) = LevelVectorsCount;
    LevelVectorsArray(Level,3) = ceil(N/LevelVectorsCount);
    LevelVectorsArray(Level,4) = LevelVectorsArray(Level,3)*
LevelVectorsArray(Level,2);
    LevelVectorsArray(Level,5) = N- (LevelVectorsArray(Level,3) *
(LevelVectorsArray(Level,2)-1));
    LevelVectorsArray(Level,6) =
LevelVectorsArray(Level,5)/LevelVectorsArray(Level,3);
    LevelVectorsArray(Level,7) = (LevelVectorsArray(Level,2)-1) +
LevelVectorsArray(Level,6);
%Set Variables for levels > 1 Containing the segments
while ceil(LevelVectorsCount) ~= 1
    l = l*2;
```

```
    Level = Level + 1;
    LevelVectorsCount = ceil(N/m/l);
    LevelVectorsArray(Level,1) = Level;
    LevelVectorsArray(Level,2) = LevelVectorsCount;
    LevelVectorsArray(Level,3) = ceil(N/LevelVectorsCount);
    LevelVectorsArray(Level,4) = LevelVectorsArray(Level,3)*
LevelVectorsArray(Level,2);
    LevelVectorsArray(Level,5) = N- (LevelVectorsArray(Level,3) *
(LevelVectorsArray(Level,2)-1));
    LevelVectorsArray(Level,6) =
LevelVectorsArray(Level,5)/LevelVectorsArray(Level,3);
    LevelVectorsArray(Level,7) = (LevelVectorsArray(Level,2)-1) +
LevelVectorsArray(Level,6);

end
% Total levels for Which to compute R/S: TotalLevels
[TotalLevels,Variables] = size(LevelVectorsArray);
%For each level
for i = 1:TotalLevels
   %if it is not the level containing the Full Data
   if ((LevelVectorsArray(i,2)) > 1 )
   %For each Vector in a level except the last vector
    for j = 1:LevelVectorsArray(i,2)-1;
        %Get the length of the vector/segment
        LV = LevelVectorsArray(i,3);
        %Define the segment involved
        zSegment=X((j-1)*LV+1:j*LV);
        %1- Compute the SmM = zSegment-Mean
        SmM = zSegment-mean(zSegment);
        %2-  Compute the Cumulative Sum of SmM = zCumSum
        zCumSum = cumsum(SmM);
        %3-  Compute the Range R of the segment = Max(zComSum)  - Min(zCumSum)
        R=max(zCumSum)-min(zCumSum);
        %4- Compute S, the standard deviation of the zSegment
        S=std(zSegment);
        %5- Compute the rescaled range of zSegment = R/S
        RS=R/S;
        % Save the R/S in the LevelVectors Array for each vector from
        % Culumn7 and up
        LevelVectorsArray(i,7+j)=RS;
    end
    else
       j=0;
   end
        %Compute the above for the LAST segment vector in the Level (since
        %it might be different from others
            j=j+1; % increment j by one to index the last segment
            %Get length of the last vector
            LV = LevelVectorsArray(i,5);
            %Define the current segment
            zSegment = X((j-1)*LV+1:N);
            %1- Compute the SmM = zSegment-Mean
            SmM = zSegment-mean(zSegment);
            %2-  Compute the Cumulative Sum of SmM = zCumSum
            zCumSum = cumsum(SmM);
            %3-  Compute the Range R of the segment = Max(zComSum)  -
Min(zCumSum)
            R=max(zCumSum)-min(zCumSum);
            %4- Compute S, the standard deviation of the zSegment
            S=std(zSegment);
            %5- Compute the rescaled range of zSegment = R/S
            RS=R/S;
```

```
            % Save the R/S in the LevelVectors Array for each vector
            LevelVectorsArray(i,7+j)=RS;
            %RSAverage(i,1) = mean(LevelVectorsArray(i,8:7+j));  % Average R/S
            RSAverage(i,1) = (sum(LevelVectorsArray(i,8:7+j-1)) +
LevelVectorsArray(i,7+j)) /LevelVectorsArray(i,2);
            RSAverage(i,2) = LevelVectorsArray(i,3); % Observations per Segment
            RSAverage(i,3) = log2(RSAverage(i,2)); % Log2 Observations
            RSAverage(i,4) = log2(RSAverage(i,1)); % Log2 R/S

            % Calculation pf Peters equation for expected value of R/S in a
            % random time series
            t=RSAverage(i,2);
            p1=((t-0.5)/t)*(sqrt(t*pi/2)^(0.5));
            p2=0;
            for r = 1:t-1
                p2=p2+sqrt((t-r)/r);
            end

            ExpectedRS=p1*p1;
            RSAverage(i,5)=ExpectedRS;
            RSAverage(i,6)=RSAverage(i,1);
            RSAverage(i,7)=log2(ExpectedRS);
            rX(i,1) = RSAverage(i,3);  %Log2 Observations
            rY(i,1) = RSAverage(i,4);  %Log2 R/S
            rY1(i,1)=RSAverage(i,7); %Log2 Expected R/S

end
%Compute the Hurst exponent by Regressing rY on rX
%linear regression
H = regress(rY,rX,0.05);
eH = regress(rY1,rX,0.05);
```

1.16. Monte Carlo Simulation for Hurst Exponent, MATLAB CODE

```
%Experiment: Try H exponent on different segments of SPY

clear all;
load SpyClose.mat;
StartPoint=1;
SegmentLength = 1500;
xSPY=data(:,2) ;
lx=MyLogReturn(xSPY);
ld=length(lx);
totalSegments = ceil(ld/SegmentLength);

for i=1:totalSegments-1
    zHSpy(i,1)=i;  %Segment Number
StartPoint=1+(i-1)*SegmentLength;
zHSpy(i,2)=StartPoint; %Start Point
EndPoint=i*SegmentLength;
zHSpy(i,3)=EndPoint; %End Point
xD=lx(StartPoint:EndPoint) ;
[RSSpy,LevelsSpy,zHSpy(i,4),zHSpy(i,5)] = MyHurstExponent(xD,2);
end

%Monte Carlo Simulation for Random Series' HURST
```

```
%Generate a Random Series
%************************************
% TOTALLY RANDOM
%************************************
%a vector of lx random values from a normal distribution of mean 0 and standard
devation 1:
%initialize the randomizer
for Simulation = 1:100
    MonteCarlo(Simulation,1)=Simulation;
    MonteCarlo(Simulation,2)=SegmentLength;
rand('twister',sum(100*clock));
z=randn(ld,1);
%Hurst
for i=1:totalSegments-1
StartPoint=1+(i-1)*SegmentLength;
EndPoint=i*SegmentLength;
xD=z(StartPoint:EndPoint) ;
[RSRandom,LevelsRandom,zHSpy(i,6),zeHRandom] = MyHurstExponent(xD,2);  %Random
Series H exponent in column 6 to compare with Peters
%zHSpy(i,7)=mean(zHSpy(:,6))-1.96*std(zHSpy(:,6)); % Min H
%zHSpy(i,8)=mean(zHSpy(:,6))+1.96*std(zHSpy(:,6)); % Max H
end
MonteCarlo(Simulation,3)=mean(zHSpy(:,6));  %mean of the simulation
MonteCarlo(Simulation,4)=zeHRandom; %expected hurst for random segment
MonteCarlo(Simulation,5)=std(zHSpy(:,6)); %Standard deviation

end
MeanSim(1,1)=Simulation; %Total number of simulations
MeanSim(1,2)= SegmentLength; %Segment Length
MeanSim(1,3)= mean(MonteCarlo(:,3)); %Mean of Hurst simulation
MeanSim(1,4)= zeHRandom; % Expected Hurst (peters)
MeanSim(1,5)=mean(MonteCarlo(:,5)); %Standard Deviation
zHSpy(1,7)=mean(MonteCarlo(:,3))-1.96*mean(MonteCarlo(:,5)); % Min H
zHSpy(1,8)=mean(MonteCarlo(:,3))+1.96*mean(MonteCarlo(:,5)); % Max H
%MeanSim(1,5)=std(MonteCarlo(:,3)); %Standard Deviation
%zHSpy(1,7)=mean(MonteCarlo(:,3))-1.96*MeanSim(1,5); % Min H
%zHSpy(1,8)=mean(MonteCarlo(:,3))+1.96*MeanSim(1,5); % Max H
[zHr,zHc] = size(zHSpy);
for j=1:zHr
    if (zHSpy(j,4)<zHSpy(1,7)) || (zHSpy(j,4)>zHSpy(1,8))
        zHSpy(j,9)=1;
    else
        zHSpy(j,9)=0;
    end
end
```

1.17. S Chart and series Embedding, MATLAB code

```
% Embedding Dimensions extrapolation by Cumulative Sums
% Calculates an nxn Matrix for all
% CumSum(log(CumSum(CumSum(x)/Av(CumSum(x)))
function
[EmbeddedSpace,MinIndex,MinValue,MinLineVal,MinLineIndex,Sdata,SLineData,EXPdata
,zValueAtMin]= MyEmbeddedDimension(zSeries,AverageIndex)
%clear all;
%load SpyClose.mat;
%X=data(:,2);  % Original Time Series
```

```
X=zSeries;
Lx = length(X);
% LiSe=(1:Lx); %Compare the exponent wrt the perfect line
%   Xl = LiSe';
 Xl = sort(X); % Compare the exponent wrt the Sorted Data
%SMatrix = zeros(Lx,Lx);
%IVector=zeros(Lx,3); %Index of min  :  Value Of Min  :  X Value in time series
%Cumulative Sum of X

Y=cumsum(X);
Yl=cumsum(Xl);
%Cumulative Sum of Y
Z=cumsum(Y);
Zl=cumsum(Yl);
i = AverageIndex;
% Z/Mean(Z)
if i>0
M=Z/mean(Z(1:i));
Ml=Zl/mean(Zl(1:i));
else
    M=Z;
    Ml=Zl;
end
% log M
L=log(M);
Ll=log(Ml);
%Cumulative Sum of L
Sdata=cumsum(L);
SLineData=cumsum(Ll);
[MinValue,MinIndex]=min(Sdata);  % zMin is the minimum value and Ind is the
index
[MinLineVal,MinLineIndex]=min(SLineData);
for j=1:Lx
EXPd(j) = log(Sdata(j))/log(SLineData(j));
end
EXPdata=EXPd';
%Now here we embed the chart
zValueAtMin=X(MinIndex);
m =MinIndex;
    TotalDataPerRow=ceil(Lx/m);
EmbeddedSpace = zeros(m,ceil(Lx/m));
for ii=1:ceil(Lx/m)-1   %For Each Column (i=column Number)
    StartPeriod=(ii-1)*m + 1;   %Start point of a column
    EndPeriod=StartPeriod + m-1; %End point of a column
x1=X(StartPeriod:EndPeriod);

xData=x1;
EmbeddedSpace(:,ii)=xData;
end
%Content of last column
tlc=Lx-(m*(ceil(Lx/m)-1));
    StartPeriod=(ii)*m + 1;   %Start point of a column
    EndPeriod=StartPeriod + tlc -1; %End point of a column
    x1=X(StartPeriod:EndPeriod);
    xData=x1;
EmbeddedSpace(1:length(xData),ii+1)=xData;
```

1.18. Back Propagation Neural Network, MATLAB code

```
%************************************************************************
%
%************************************************************************

%************************************************************************
% What To run (1:run , 0: do not run
clear
   RunOpen = 0;
   RunHigh = 0;
   RunLow = 0;
   RunClose = 1;
   % The bellow is not relevant to forex or stock, it was run on a random
   % time series only to see the difference in results when the series is
   % highly chaotic (i tried italian lottery, since lottery is highly chaotic
and random )
   % this needs another mat file
   RunSen = 0;
   PredVal = 1;
   AllSix=1;

% End what to run
%************************************************************************
% LR=0.5  1 layer 3P  = 55%
%

lRate = 0.5 ;   %0.5 seems good
zEpochs = 20000 ;


%i tried with PCS but the results are not better, an more, dimensionality
%is not a problem here
MatlabPca = 0;
MaxFrac = 0.05; %0.15:2 features  0.10:3 features 0.05: 4 features
nFeatures =8;
%************************************************************************
%************************************************************************

%*


load eur-us1005.mat  %eur-us1005.mat has a total of 697 patterns
[dRow, dCol] = size(data);
StartFrom = 1;
trainDim =364;  %this is to set the training set dimension
testDim=157; %this is to set the testing set dimension
zFeatures=1;
FeatureStart = 1;
Mult=1;  %this is a multiplier used to enhance teh precision, since the NN will
approximate the values, in forx we are interested in minimum price change
sometimes to 4th or 5th decimal point precision
data = data * Mult;  %original data multiplied by MULT
% TRAIN AND TEST DATA
TrainSet = data(FeatureStart:zFeatures,StartFrom:StartFrom+trainDim);
TestSet = data(FeatureStart:zFeatures, StartFrom+trainDim+1:dCol);
```

```
% TRAIN TARGETS - OPEN - HIGH - LOW - CLOSE
OpenTrainTarget = data(zFeatures+1,StartFrom:StartFrom+trainDim);
HighTrainTarget = data(zFeatures+2,StartFrom:StartFrom+trainDim);
LowTrainTarget = data(zFeatures+3,StartFrom:StartFrom+trainDim);
CloseTrainTarget = data(zFeatures+4,StartFrom:StartFrom+trainDim);
% TEST TARGETS - OPEN - HIGH - LOW - CLOSE
OpenTestTarget = data(zFeatures+1,StartFrom+trainDim+1:dCol);
HighTestTarget = data(zFeatures+2,StartFrom+trainDim+1:dCol);
LowTestTarget = data(zFeatures+3,StartFrom+trainDim+1:dCol);
CloseTestTarget = data(zFeatures+4,StartFrom+trainDim+1:dCol);

if RunSen == 1
    SenTrainTarget = data(zFeatures+PredVal,StartFrom:StartFrom+trainDim);
    SenTestTarget = data(zFeatures+PredVal,StartFrom+trainDim+1:dCol);
end
cc=0;
          P = TrainSet;
          T = TestSet;
          [xRow, xCol] = size(P);


% Set Networks Parameters
%*******************************
if RunSen == 1

    tGoal = 0.0000001; % average square error
    tGrad = 0.0001; %GRADIENT
NetSen = newff(minmax(P),SenTrainTarget,[13 26] ,{'logsig','logsig','purelin'});
NetSen.trainParam.epochs = zEpochs;
NetSen.trainParam.lr = lRate;
NetSen.divideFcn = '';
NetSen.trainParam.max_fail = inf;
NetSen.trainParam.min_grad = tGrad;
NetSen.trainParam.mu_max = inf;
NetSen.trainParam.goal = tGoal;
NetSen = train(NetSen,P,SenTrainTarget);
NetSenTrain = sim(NetSen, P);
NetSenTest= sim(NetSen, T);
plot(NetSenTest/Mult,'r');
hold;
plot (SenTestTarget/Mult,'g');
legend('Predicted', 'Real');
nTestResult = NetSenTest/Mult;
nTestTarget =SenTestTarget/Mult;
[nRow,nCol] = size(nTestResult);
PerTest = 0;
for m = 1:nCol
    if round(nTestResult(m)) == round(nTestTarget(m))
        PerTest = PerTest+1;
    end

end
CorrectPer = PerTest * 100 /nCol;
disp(CorrectPer);
Comp = [nTestResult ; nTestTarget];

end
```

```
%*********************************
%*********************************
%*********************************
if RunOpen == 1
NetOpen = newff(minmax(P),OpenTrainTarget,[9 18]
,{'logsig','logsig','purelin'});
NetOpen.trainParam.epochs = zEpochs;
NetOpen.trainParam.lr = lRate;
NetOpen.divideFcn = '';
NetOpen.trainParam.max_fail = inf;
NetOpen.trainParam.min_grad = 0.0001;
NetOpen.trainParam.mu_max = inf;
NetOpen.trainParam.goal = tGoal;
NetOpen = train(NetOpen,P,OpenTrainTarget);
NetOpenTrain = sim(NetOpen, P);
NetOpenTest= sim(NetOpen, T);
plot(NetOpenTest/Mult,'r');
hold;
plot (OpenTestTarget/Mult,'g');
legend('Predicted Open', 'Real Open');
end
%*********************************
%*********************************
if RunHigh ==1
NetHigh = newff(minmax(P),HighTrainTarget,[9 18]
,{'logsig','logsig','purelin'});
NetHigh.trainParam.epochs = zEpochs;
NetHigh.trainParam.lr = lRate;
NetHigh.divideFcn = '';
NetHigh.trainParam.max_fail = inf;
NetHigh.trainParam.min_grad = 0.0001; %Gradient
tGoal = 0.0001; % average square error
NetHigh.trainParam.mu_max = inf;
NetHigh.trainParam.goal = tGoal;
NetHigh = train(NetHigh,P,HighTrainTarget);
NetHighTrain = sim(NetHigh, P);
NetHighTest= sim(NetHigh, T);
plot(NetHighTest/Mult,'r');
hold;
plot (HighTestTarget/Mult,'g');
legend('Predicted High', 'Real High');
end
%*********************************
%*********************************
%*********************************
if RunLow == 1
NetLow = newff(minmax(P),LowTrainTarget,[9 18] ,{'logsig','logsig','purelin'});
NetLow.trainParam.epochs = zEpochs;
NetLow.trainParam.lr = lRate;
NetLow.divideFcn = '';
NetLow.trainParam.max_fail = inf;
NetLow.trainParam.min_grad = 0.00001;
tGoal = 0.001; % average square error
NetLow.trainParam.mu_max = inf;
NetLow.trainParam.goal = tGoal;
NetLow = train(NetLow,P,LowTrainTarget);
NetLowTrain = sim(NetLow, P);
NetLowTest= sim(NetLow, T);
plot(NetLowTest/Mult,'r');
hold;
plot (LowTestTarget/Mult,'g');
legend('Predicted Low', 'Real Low');
```

```
end
%*********************************
%*********************************
if RunClose == 1
NetClose = newff(minmax(P),CloseTrainTarget,1 );
NetClose.trainParam.epochs = zEpochs;
NetClose.trainParam.lr = lRate;
NetClose.divideFcn = '';
NetClose.trainParam.max_fail = inf;
NetClose.trainParam.min_grad = 0.00001;
tGoal = 0.001; % average square error
NetClose.trainParam.mu_max = inf;
NetClose.trainParam.goal = tGoal;
NetClose = train(NetClose,P,CloseTrainTarget);
NetCloseTrain = sim(NetClose, P);
NetCloseTest= sim(NetClose, T);
plot(NetCloseTest/Mult,'r');
hold;
plot (CloseTestTarget/Mult,'g');
legend('Predicted Close', 'Real Close');
end
%*********************************
```

1.19. Monte Carlo Simulation for the suggested exponent K

```
%Monte Carlo simulation for K
clear all;
load SPYclose;
xSpy = data(:,2);
Spy100 = xSpy(1:100);
[EmbeddedSpy,MinIndexspy,MinValuespy,MinLineValspy,MinLineIndexspy,Sdataspy,SLin
eDataspy,EXPdataspy,zValueAtMinspy]= MyEmbeddedDimension(Spy100,100);
%Build a vector of length Double the MinIndexValue
fHs1 = xSpy(1:2*MinIndexspy,1);
%Construct the embeding and the S chart for the built vector
[EmbeddedfHs1,MinIndexfHs1,MinValuefHs1,MinLineValfHs1,MinLineIndexfHs1,SdatafHs
1,SLineDatafHs1,EXPdatafHs1,zValueAtMinfHs1]= MyEmbeddedDimension(fHs1,100);
%Separate the S Vector in 2 equal parts
Ss1 = SdatafHs1(1:MinIndexfHs1);
Ss2 = SdatafHs1(MinIndexfHs1+1:2*MinIndexfHs1);
LogSs1 = log(abs(Ss1));
LogSs2 = log(abs(Ss2));
Ks1=regress(LogSs2,LogSs1);
Ks2=regress(LogSs1,LogSs2);

%Generate a Random Series
%*************************************
% TOTALLY RANDOM
%*************************************
%a vector of 120 random values from a normal distribution of mean 0 and standard
devation 1:
%initialize the randomizer
%================================================
for MC = 1:5
for zSim =1:1000
rand('twister',sum(1000*clock));
z=randn(200,1);
sZ = cumsum(z);
%Make the series look like a price series
sZ=sZ+ abs(min(sZ)) + 10;
```

```
[sZ,i]=shuffle(sZ,1);
[EmbeddedsZ,MinIndexsZ,MinValuesZ,MinLineValsZ,MinLineIndexsZ,SdatasZ,SLineDatas
Z,EXPdatasZ,zValueAtMinsZ]= MyEmbeddedDimension(sZ,100);
%Build a vector of length Double the MinIndexValue
fH1 = sZ(1:2*MinIndexsZ,1);
%Construct the embeding and the S chart for the built vector
[EmbeddedfH1,MinIndexfH1,MinValuefH1,MinLineValfH1,MinLineIndexfH1,SdatafH1,SLin
eDatafH1,EXPdatafH1,zValueAtMinfs1]= MyEmbeddedDimension(fH1,100);
%Separate the S Vector in 2 equal parts
S1 = SdatafH1(1:MinIndexfH1);
S2 = SdatafH1(MinIndexfH1+1:2*MinIndexfH1);
LogS1 = log(abs(S1));
LogS2 = log(abs(S2));
K1=regress(LogS2,LogS1);
K2=regress(LogS1,LogS2);

%K1=regress(S1,S2);
%K2=regress(S2,S1);
SumK1K2 = K1+K2;
DifK1K2 = K1-K2;
Simulation(zSim,1) = zSim;
Simulation(zSim,2)= K1;
Simulation(zSim,3)= K2;
Simulation(zSim,4)= SumK1K2;
Simulation(zSim,5)= DifK1K2;
Simulation(zSim,6)= abs(1-K1);
end
StatsData(MC,1)=mean(Simulation(:,3)); %mean |K1|
StatsData(MC,2)=std(Simulation(:,3)); %stdv |K1|
end
StatsData(MC+1,1)=mean(StatsData(1:MC,1)); %Mean of the simulations
StatsData(MC+1,2)=mean(StatsData(1:MC,2)); %Mean of the Standard Deviation
StatsData(MC+2,1)=std(StatsData(1:MC,1)); % Standard Deviation of mean of all
simulations
StatsData(MC+3,1)=StatsData(MC+1,1) - 1.95*StatsData(MC+1,2);
StatsData(MC+3,2)=StatsData(MC+1,1) + 1.95*StatsData(MC+1,2);
%StatsData(MC+4,1)=abs(1-StatsData(MC+3,1));
%StatsData(MC+4,2)=abs(1-StatsData(MC+3,2));
```