Model Based Multi Criteria Decision Making Methods for Prediction of Time Series Data

Ahmed Salih Ibrahim

Submitted to the
Institute of Graduate Studies and Research
in partial fulfilment of the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University
January 2014
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering.

Prof. Dr. Işik Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Asst. Prof. Dr. Mehmet Bodur
Supervisor

Examining Committee

1. Asst. Prof. Dr. Adnan Acan ______________________
2. Asst. Prof. Dr. Mehmet Bodur ______________________
3. Asst. Prof. Dr. Ahmet Ünveren ______________________

ABSTRACT

Financial forecasting is a difficult task due to the intrinsic complexity of the financial system, in this research the estimation of the stock exchange prices is targeted using the five-year time series data of prices. The objective of this work is to use an intelligence techniques and mathematical techniques to create a model, that has the ability to predict the future price of a stock market index, then decide throughout the k-means clustering with majority voting, which one of those prediction techniques is the best. It is a multi-decision making in order to find the best predictive method. The proposed method combines multiple methods to have higher prediction accuracy and higher profit/risk ratio. The forecasting techniques, namely, Radial Basis Function (RBF) combined with Self-organizing map, Nearest Neighbour (K-Nearest Neighbour) methods, and Autoregressive Fractionally Integrated Moving Average (ARFIMA) are implemented in forecasting the future price of a stock market index based on its historical price information, and the best forecast of these three methods is decided by majority voting after k-means clustering. The experimentation was performed on data obtained from the London Stock Exchange. The data used was a series of past closing prices of the Share Index. The results showed that the proposed decision method provides better prediction than forecasts of the three techniques.

Keywords: Forecasting, SOM-RBF, K-Nearest Neighbour, ARFIMA, Decision-making.

ÖZ

Finansal sistemlerin iç karmaşası nedeniyle finansal tahmin zor bir iştir. Bu araştırmada beş yıllık zaman serisi verisini kullanarak hisse senedi fiyatlarının tahmini amaçlanmaktadır. Çalışmanın amacı hisse senetlerinin gelecekte fiyatını çeşitli matematiksel ve yapay ussal tahmin yöntemleri kullanarak bulup, ardından, k-ortalama öbekleme yöntemi ile hangi tahmin yönteminin daha başarılı olduğuna karar vermektir. Böylece oluşturulan çoklu karar verme mekanizması her durum için en iyi tahmin yöntemini bulur. Tahmin yöntemleri olarak Kendinden Düzenli Radyal Baz Fonksiyonu (RBF) en yakın komşu (K-Nearest Neighbour) metodu, ve Atoregressif Oranlı Tümlevsel Gezer Ortalama (ARFIMA) metodları kullanılarak beş yıllık zaman serisinden gelecekteki değer tahmin edilmiş ve üçü arasında en iyi tahmin eden metoda k-ortalama öbekleyici ve çoğunluk oyu kullanarak karar verilmiştir. Deneyler Londra Hisse Senetleri Borsasından alınan beş yıllık günluk kapanış veri üzerinde denenmiştir. Sonuçlar önerilen yöntemin seçtiği tahminin, kullanılan her üç yöntemin tahminden daha başarılı olduğunu göstermektedir.

To the most fascinating persons that I met in my life …

Dad & Mom

ACKNOWLEDGEMENTS

The words fail to express my gratitude to Dr. Mehmet Bodur for his Supervision and guidance from the very early stage of this research. And, without his guide, this work will never see the light.

I would like to extend my thanks to all the staff and instructors of the Department of Computer Engineering at Eastern Mediterranean University who helped me during the study.

I would like to take this opportunity to thanks my dear family who has been always patient and supportive during my studies.

Finally, my deepest thank goes to my dear friends for their support during the study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\emptyset_t$	The set of available information.
t	Time
P_t	The stock price at time *t*
ϵ_t	The forecast error at time *t*
E	The expected value operator
x_t	Actual price
r_t	Daily returns series
y_t^m	Vectors (pieces) of information
T	The number of observation on the training period
m	The embedding dimension of the time series
/ρ/	The absolute (Euclidian) correlation
$\hat{\alpha}_m$	Coefficients derived from the estimation of a linear model.
$\mathcal{X}$	High-dimension continuous input space of neurons.
$\mathcal{A}$	Low-dimension discrete space of neurons.
$\mathcal{N}$	Neurons.
W	Weights.
$\|\| \cdot \|\|$	Denotes the Euclidean distance.
$\alpha(t)$	Learning rate.
$h(i^*, i; t)$	The weighting function
$r_i(t)$	The coordinates of the neurons i in the output array
$\sigma(t)$	The radius of the neighbourhood function at time *t*.
$x^{in}(t)$, $x^{out}(t)$	The portions of the weight (prototype) vector

$s(t)$	The signal sample transmitted at the time step$t$.
$\hat{s}(t)$	An estimation of the transmitted sampled signal at time *t*.
$y(t)$	The corresponding channel output
p	The order of the equalizer
γ	The radius (or spread) of the function.
g_i	The gate output of expert filters.
L	The lag operator
ε_t	Errors variables sampled from a normal distribution.
$\delta/(1-\sum\varphi_i)$	The drift process.
$\Phi(L)$	The autoregressive polynomial.
$\Theta(L)$	The moving average polynomial.
p & q	Integers.
d	Real.
$(1-L)^d$	Fractional difference operator.
μ_t	Mean.
γ_k	Covariance function.
c	Finite nonzero constant.
$\mathcal{S}_j$	Coefficients.

LIST OF ABBREVIATIONS

EMH	Efficient market hypothesis
RBF	Radial basis function
SOM	Self organizing map
VQTAM	Vector-Quantized Temporal Associative Memory
KRBF	Local radial basis function
K-Nearest Neighbour	Nearest Neighbour Method
ARFIMA	Autoregressive Fractionally integrated moving average
LSE	London Stock Exchange
AR	Autoregressive
MA	Moving Average
USD	United States Dollar
Matlab	A software package for mathematical operations, Math Works, Inc., R2013b
LND	London stock market
GBP	British Pound

Chapter 1

INTRODUCTION

The forecasting of the stock market became an important financial project that attracts the researchers' attention in recent years. It includes a theory that past values of available information has some predictive relationship with the market future prices [1]. The prototypes of such information involves an economic variables such as interest rates and exchange rates, industry specific information such as growth rates of industrial production, and company specific information such as income statements.

The stock markets' studies focus on two areas, namely testing the stock market efficiency and modelling stock prices. Efficiency of the stock market is defined clearly by the concept called the efficient market hypothesis (EMH). Different modelling techniques have been used for trying to model the prices of the stock market index. These techniques have been focused on two areas of forecasting, namely technical analysis and fundamental analysis.

Technical analysis considers that market activity discovers significant new information and understanding the psychological factors that responsible for influencing the stock price in order to forecast future prices and trends. This approach is based on the assumption that the price is a reflection of the crowd in action, it is try to predict the future price movements according to the crowd psychology when moves between panic, fear, and pessimistic on one hand and confidence, excessive optimistic, and greed [6]. There are many techniques that fall under this category of analysis, the most well-known being the moving average (MA).

Fundamental analysis is a very effective way to predict the economic conditions and focuses on money policy, government policy when dealing with economy, and economic indicators such as G.D.P (gross domestic product), exports, imports and any work inside the business cycle framework. It used mathematical methods include vector auto-regression (VAR) which is a multivariable modelling technique.

The study focused on the forecasting of the London stock market index using artificial intelligence techniques radial basis function (RBF) and nearest neighbour (K-Nearest Neighbour), and the mathematical method Auto Regressive Fractionally Integrated Moving Average (ARFIMA). The study involves an assumption which is the predictions can be made based on stock price data alone. RBF, K-Nearest Neighbour and ARFIMA systems were used to attempt to predict the share index of the London stock market. Furthermore, the three techniques' predictive power also compared.

1.1 Efficient Market Hypothesis

As mentioned by Fama [2], which states that all available information that obtained by the market, affecting the current stock values. And the traders are able to make trades based on it. Therefore, it is not possible to predict the future prices index since they are already known the reflecting about the stocks. The efficient market will instantaneously control the prices of stock markets according to the news, which comes to the market in a random way [3]. And it is not impossible to forecast the future prices with results, which are better than random [5]. Many studies in this area provide evidence that the stock market is not completely efficient [7, 8, 9, 10, 11] nor do stock prices follow a random walk [12].

There are three types of EMH [4]:-

- **Weak-Form Efficiency** - this form state that the past information is fully incorporated with the current price and does not have any predictive power. This means that predicting the future returns of an asset based on technical analysis is impossible.
- **Semi-Strong Form Efficiency** - this form state that any public information is fully incorporated in the current price of an asset. Public information includes the past prices and the information that have been recorded in a company's financial statements, dividends announcement, the financial report of company's competitor, and any economic factors.
- **Strong Form Efficiency** - this form state that the current price incorporate any information, both public and private. This means that no market actor is able to derive profits continuously even when trading with information that is not already public knowledge.

1.2 Random Walk Hypothesis

This hypothesis states that all stock prices follow a random walk and do not follow any patterns or trends. It is initially theorized by a French economist Louis Bachelier in (1900), however it came back into main stream economics in (1965) in Eugene Fama's article "Random Walks in Stock Market Prices" [13].

1.3 Trading Strategies

Some investors, who are looking to manage their decisions by a set of rules, in order to eliminate any trades influenced by emotional response or feeling, will use a trading strategy, which lays out a set of rules about when to enter or exit a trade. The usually set of rules may be very easy like follows interactions between moving averages or very complicated like sort selling, where man things need to be satisfied before making the decision.

1.4 Motivation

The most important motivation for trying to forecast the stock prices of the market is financial gain. The ability to create a mathematical model that can forecast the direction of the stock prices in the future would make the owner of the model very wealthy. Thus, researchers, investors and investment professionals are always attempting to find a stock market model that would make a higher return.

1.5 Research Hypotheses

For this research, the following hypotheses have been identified:

1. Neural networks can be used to forecast the future prices of the stock market.
2. Mathematical methods can be used to forecast the future prices of the stock market.
3. Self-organizing map (SOM) combined with neural network can be used to forecast the future stock market prices of the stock Market.
4. Cluster the forecasted future prices of each method for a training period
5. Determine the best method for each cluster by majority voting.

1.6 Objective of the Report

The first objective of this work is to apply self-organizing map (SOM) combined with radial basis function network (RBF), ARIMA and nearest neighbour (K-Nearest Neighbour) to forecast the share index of the London stock market. The second objective is to compare the accuracy of the prediction results of each tool **(**according to their results**).** Then cluster the prediction results that came out from those methods and decide which one of them is the best predictive method throughout decision-making procedures. However, it must be stated that the objective of this report is not to develop a guideline for investing on assets in the stock market. This work should be regarded as a decision-making support tool when deciding to predict the market.

1.7 Structure of the Thesis

A brief description of the structure of the thesis contains the following items: Chapter 1 is an introduction to the market-forecasting problem. Chapter 2 explains the stock market data, Chapter 3 describes the Prediction methods, and Chapter 4 introduces the proposed decision making algorithm by k-means clustering. Finally, Chapter 5 concludes the thesis.

Chapter 2

DATA AND STOCK MARKET

This chapter aims to introduce the reader to the stock market and it’s data, the process of fixing the missing days in the stock market, then covers the forecasting in general definition and it’s objects, and finally discuss the previous techniques of forecasting and their effects in the history of market forecasting.

2.1 London Stocks Market

The London Stock Exchange (L.S.E), is a stock exchange which is located in Paternoster Square close to street Paul's Cathedral in the City of London in the United Kingdom. In the Dec. (2011), the Exchange had a market capitalization of 3.266$ trillion, making it number four among the largest stocks exchange in the world (and the largest in Europe) [17].

2.1.1 The Market Index

A market index defined as a statistical measure of the changes in the shares of the market [16]. The fluctuations of the market are represented by the index of that market. With the growing importance of the shares market in different societies, indices like the LSE All Share, DJIA, Nikkei and NASDAQ. Composite have grown to become a part of everyday vocabulary. The index used as a measure of the performance of the market and the change in the price of the index represents the proportional change of the shares included in the index.

2.2 The Dataset

A time series data is a sequence of the obtained data, which are ordered by time with equally spaced time intervals such as daily or hourly like air temperature. The datasets generated in many areas and the analysis of it has wide programmes in controlling process, economic prediction, marketing, social studies, medical science etc. the analysis of data uses programmed approaches to extract information and understand the properties of a physical system that generates the time series data.

In stock markets, the trading volume (index) is an important part of the technical analysis theories. Recently, some researchers have discovered evidences that the relation between trading volume and price fluctuation is obviously strong.

In this thesis, the 5 years datasets will be use, which consists of circadian closing prices of London stock market. The dataset's period contains the trading days starting from (1st January 2008 to 31st December 2012). It has collected from the actual data accessible on the website of *finance.yahoo.com/* [28]. The data Contains missing days, and those days will effect on the forecasting results, so the data have been processed using methods of data compensation.

2.2.1 Pre-processing of Data

Data pre-processing is a data mining technique that includes the transformation process of raw data into an understandable format. The real-world data may be inconsistent, incomplete, or lacking in certain trends, and may contain several errors inside. Data pre-processing is an ensured method for solving these problems [39]. In this research, forecasting of the market prices required a recorded datasets with daily closing prices of the stock market of London, and we decide to use a data starting from the 1st-

January, of (2008) to 31st-December, (2012), which was obtained from the financial data accessible on *finance.yahoo.com/* [28]. As we mentioned before, missing vector means no recorded data is available in that day, and missing value means several vectors are missing from the daily record. And this is because the stock exchange does not open all the days of the year. So, the missing vectors and values may have some changes on the features of datasets when it used for prediction purposes.

There are methods to rebuild the missing data values inside the range of a time series. The linear interpolation method is the most commonly used for solving the missing data, which is done by taking the weighted mean of the values of the before missing day and after it.

For example, in the following table (Table 1), it contains a set of values corresponding to an index; but the 3rd value of *F* is missing.

Table 1: Example of Missing Value

Y	F(y)
1	0
2	0.9093
3	-
4	-0.7568
5	-0.2794

Interpolation method will provide an average value by using the estimation function at middle points, from both before and after values. The function is:

$$f(3) = (f(2) + f(4))/2 \qquad (2.1)$$

In Table 2, we have a sample of the real (raw) closing prices of London stock market from 06/08/2008 to 11/08/2008 with missing days at 9-10/08/2008.

Table 2: Real Data of London Stock Market

Days	Closing Price
06-08-2008	10738.49214
07-08-2008	10699.20075
08-08-2008	10690.217
11-08-2008	10643.0269

After data was pre-processed by linear interpolation, we obtained the following table:

Table 3: Real Data After Pre-processing

Days	Closing price
06-08-2008	10738.49214
07-08-2008	10699.20075
08-08-2008	10690.217
09-08-2008	10674.486967
10-08-2008	10658.756933
11-08-2008	10643.0269

Figure 1 and Figure 2 show daily close price and returns of London stock market after the data of stock market was pre-processed.

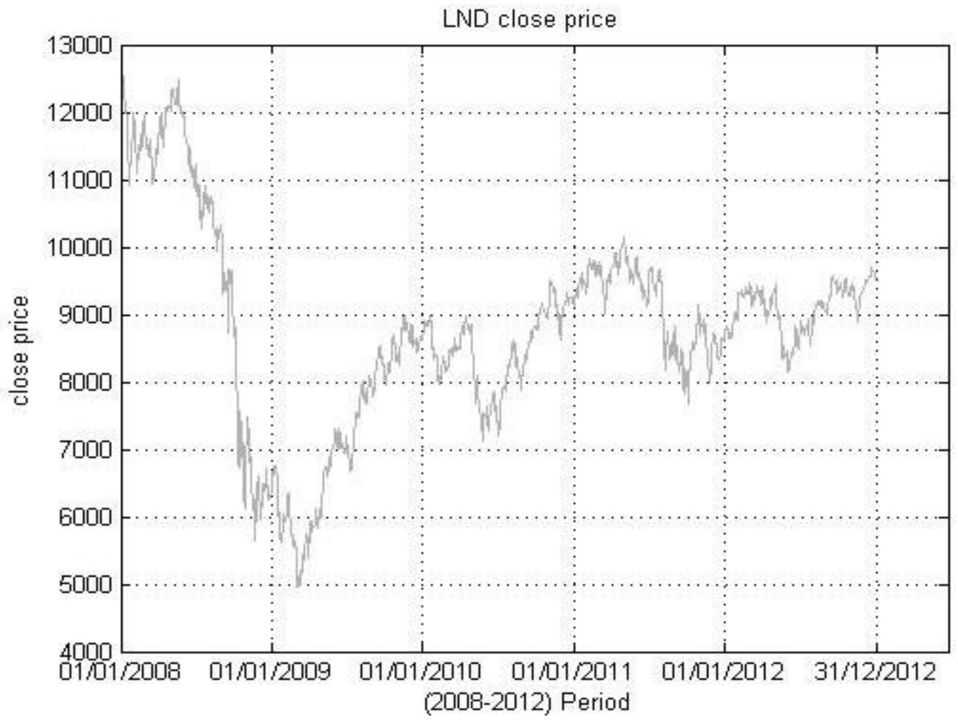

Figure 1: Closing Prices of London Stock Market

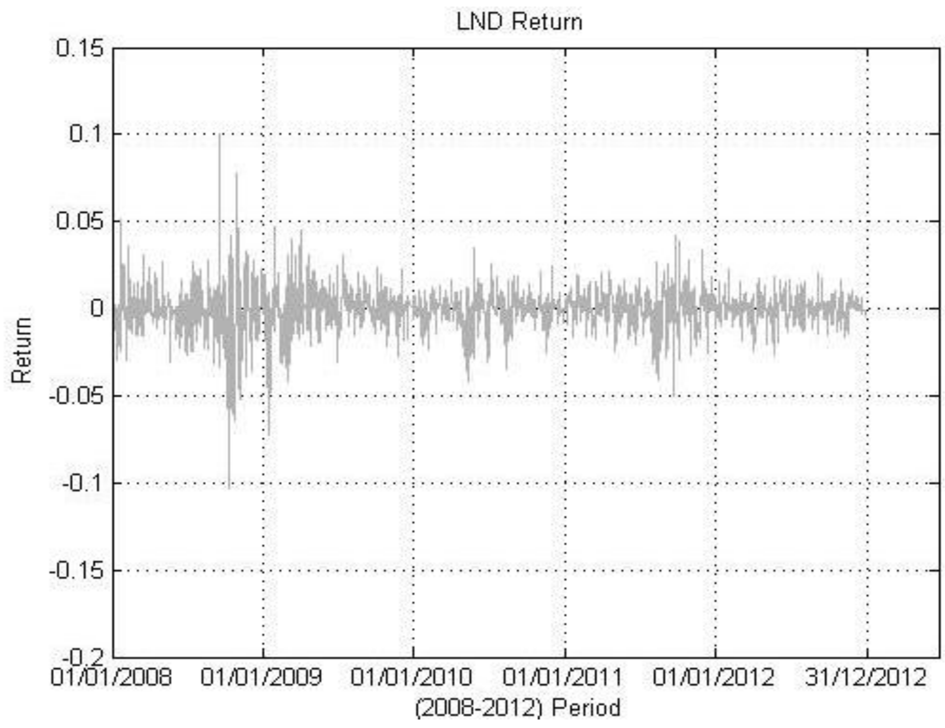

Figure 2: Returns of London Stock Market

2.3 Forecasting

The core component of the analysis of market is the market prediction. It explain the future numbers, properties, and trends in the target market [29]. The main objective of forecasting the stock markets is for decision makers to make better decisions.

2.3.1 Reality Checks

The forecasting results of the market index must always be a subject to a reality check. When we have forecasts, we need to derive a way to test it for reality. In this case, if the total market is worth some estimate, you might check with traders who sold products to this market in some given years to see whether their results check with your forecasts. You might look for macro-economic factors to confirm the relative size of this market compared to other markets with similar properties [29].

2.4 Modelling Stock Prices or Returns

It means finding the best predictive model that can represent the problem based on the given information. In addition, when the historically information is available, it will makes it possible to generate a statistical predictive model based on data. Many forecasting techniques were proposed to forecast the future event based on past observations. And, here we will presents the some models that were published previously, used currently, and still under research.

(a) Smoothing

Smoothing methods are used to determine the average value around which the data is fluctuating. There are two examples of this type of approach, which are the moving average and exponential smoothing. Moving averages is created by summing up the data series and divide it by the total number of observations [41]. Exponential smoothing is created by taking the value of weighted average of the previous observations, where the weights are decreasing with the age of the past observations.

(b) Curve Fitting

The graphical history of datasets sometimes exhibits properties, patterns, and they repeating themselves over time. Curve fitting or data mining, is about drawing the conclusions based on past available information. When apply it to an investment procedure or trading strategy, the historical results, showed that such conclusions do not had a true values they are implemented [42].

(c) Linear Modelling

The predicting process with linear regression models have shown their usefulness in predicting the returns both in developed and developing markets [43]. And it could predict the direction of market prices 50 percent of the time, according to establishment of random walk hypothesis, which assumes that to get the best results in forecasting the future price of the market index; it should be based on the current price itself.

(d) Non-linear Modelling

In recent years, when they discover the non-linear movements in the financial stock markets, it become one of the important things that attract many researchers and financial analysts [44]. And the most important part, is the mounting evidence that the distribution of stock returns is good to represent by a mixture of normal distributions (e.g. see Ryden, Terasvirta [45] and the references therein).

The neural networks are able to perform the non-linear modelling without any prior knowledge about the relationship between input and output variables. Because of that, there has been a growing interest to apply the artificial intelligence techniques in order to capture the future stock characteristics, see [46, 47, 48, 49], for previous work on stock predictions. Among many nonlinear models, the forecasters prefer to use the artificial neural networks as a non-parametric regression method [50]. Because it has the ability to establish relationships in areas where mathematical knowledge of the analysed time series is absence and difficult to rationalize [51].

2.5 Forecasting Studies

In 1969, Reid [15], Newbold, and Granger [16] examined a large number of time series to determine their accuracy in post-sample forecasting. However, the first idea about comparing a large number of time series was conducted by Makridakis and Hibon [14]. Their major conclusion was that the developed statistical methods do not need to be an outperform simple methods such as exponential smoothing. The conclusion was strongly criticized, because it does not be similar to the conventional wisdom. Makridakis, continuing his studies by launching M-competition (predicting competition arranged by Spyros Makridakis where the errors of forecasting methods were compared for forecasts of a variety of economic time series) to respond to criticisms. The results of the M-competition were not different from the earlier conclusions made by Makridakis and Hibon [14].

Chapter 3

PREDICTION METHODS

The chapter aims to introduce to the reader the four forecasting techniques that were used in the study, and explain each one of them in details and formulas, those prediction techniques are used to predict the stock market index. Then choose the best predictive one throughout clustering and decision-making procedures.

3.1 Nearest Neighbour Method

The nearest neighbourhood method is defined as a non-parametric class of regression. Moreover, the main idea behind this approach is that the time series copies its own characteristics along the time. In other statements, past vectors of information of the time series have symmetry with a last vector of available information on *t+1*. Such way of detecting the patterns in the datasets characteristics is the main goal for the similarity between K-Nearest Neighbour algorithm and other analysis techniques. The way that the K-Nearest Neighbour works on is totally different from the normal method like the ARIMA. The ARIMA is try to detect the statistical patterns between the observations according to their locations in time. For the K-Nearest Neighbour, such location does not play any role and it is not important, since the goal is to locate similar vectors of information, without need to know where they have been located in time. Ignore all the mathematical forms, the main idea of the K-Nearest Neighbour method is to detect the dynamic non-linearity of similarity in the datasets.

Next, we will describe the method of K-Nearest Neighbourhood. For a detailed review of the process see [30, 31].

3.1.1 The K-Nearest Neighbour with Correlation

The correlation case for the K-Nearest Neighbour algorithm works according to the following steps:

1) The first step is to determine the start of the training period and divide it into different vectors (pieces) y_t^m with size *m*, where *t* is equal to *m*. The value of *T* means the number of reviews (observations) in the trained datasets. The term *m* is assign to be the embedding dimension of the datasets. The last available vector of information before the prediction of the observations will called it y_T^m, and the other vectors will be defined as y_i^m.

2) The second step is to choose the most similar *k* pieces to y_T^m. In correlation technique, the searching process of the *k* vectors is done by taking the highest value of 'ρ', which is defines the absolute of the (Euclidian correlation) between y_i^m and y_T^m .

3) When the detecting process of *k* pieces is done, each one of them with *m* reviews, it will be needed to understand the construction way of the forecasts on *t+1* by using the *k* vectors. There are many ways could be used here, like using an average or tricube functions [18]. The method contains the calculation of the following formula [19].

$$\hat{y}_{T+1} = \hat{\alpha}_0 + \hat{\alpha}_1 y_{T-1} + \hat{\alpha}_2 y_{T-2} + \cdots + \hat{\alpha}_m y_{T-m} \tag{3.1}$$

Where $\hat{\alpha}_0, \hat{\alpha}_1 .. \hat{\alpha}_m$ represent the coefficients values, which were captured by estimating the linear-model with corporated variables as y_{i+1} and from the explanatory variables such as $y_{i_r}^m = (y_{i_r}, y_{i_{r-1}}, \cdots, y_{i_r-m+1})$ where *r* is from 1 to *k*. and to try to understand such regression, (3.1) will be present in a matrix way, in next expression.

$$\begin{bmatrix} y_{i_1+1} \\ y_{i_2+1} \\ y_{i_3+1} \\ \vdots \\ y_{i_k+1} \end{bmatrix} = \hat{\alpha}_0 + \hat{\alpha}_1 \begin{bmatrix} y_{i_1} \\ y_{i_2} \\ y_{i_3} \\ \vdots \\ y_{i_k} \end{bmatrix} + \hat{\alpha}_2 \begin{bmatrix} y_{i_1-1} \\ y_{i_2-1} \\ y_{i_3-1} \\ \vdots \\ y_{i_k-1} \end{bmatrix} + \dots + \hat{\alpha}_{m-1} \begin{bmatrix} y_{i_1-m+1} \\ y_{i_2-m+1} \\ y_{i_3-m+1} \\ \vdots \\ y_{i_k-m+1} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \vdots \\ \varepsilon_k \end{bmatrix} \tag{3.2}$$

It is necessary to notice that the K-Nearest Neighbour method is non-temporal. The value of y_{i_k+1} will represent the observations (reviews) in one period-ahead from the vectors chosen by the correlation method. The value of y_{i_k-m} will represent the first *k* chosen vectors, while the term y_{i_k} represent the last terms of each chosen vector. It is simple to notice that the number of explanatory time series on Equation (3.2) is *m*, and each one of them have *k*.

The term $\hat{\alpha}_1$, in Equation (3.2), is the attached coefficient to the last review of the chosen sets and $\hat{\alpha}_2$ is the attached coefficient to all the second last reviews (observations) of all *k* series. Those coefficients are remaining until they reach the first review of all chosen *k* series. The values of the coefficients in Equation (3.2) are detected with the minimum summation of the error $(\sum_{i=1}^{k} \varepsilon_k^2)$, the steps 1-3 are executed in a main loop until all forecasts on *t+1* are created.

3.1.2 The K-Nearest Neighbour with Absolute Distance

The absolute distance of the K-Nearest Neighbour is much easier than the one with the correlations. Its steps are:

1) The first step is to determine the start of the training period and divide it into different vectors (pieces) y_t^m with size *m*, where *t* is equal to *m*. The value of *T*

means the number of reviews (observations) in the trained datasets. The term m is assign to be the embedding dimension of the datasets. The last available vector of information before the prediction of the observations will called it y_T^m, and the other vectors will be defined as y_i^m.

2) The second step is to choose the most similar *k* pieces to y_T^m. In absolute distance technique, the searching process of the *k* vectors is done by taking the lowest summation of distances between the vectors y_i^m, and y_T^m.

3) When the detecting process of *k* pieces is done, each one of them with *m* reviews (observations), it will be needed to understand the construction way of the forecasts on *t+1* by using the *k* vectors. The absolute distance techniques easily verify the observation ahead of the chosen *k* neighbours and take the mean value of them.

The steps 1-3 are executed in a main loop until that all forecasts on *t+1* have been created. And we should demonstrate that the absolute distance does not use any kind of local regression.

3.2 Radial Basis Function Method

In this section we will proposed a model based on a combination of a radial basis function (RBF) and a self-organizing map (SOM). The results shown that the proposed SOM-RBF is more specified in modelling and forecasting the time datasets. The proposed model is applied to stock market data.

3.2.1 Self-Organizing Map Based on Adaptive Filtering

The Self-Organizing Map (SOM) is one of the common learning tools, which learn from a high-dimension continuous input space like $\mathcal{X}$ to low-dimension discrete space like $\mathcal{A}$ of $\mathcal{N}$ neurons, which are ordered in fixed form like a two dimension array. The map $i^*(x): \mathcal{X} \to \mathcal{A}$ is determined by the weight matrix $W = (w_1, w_2, \dots, w_q)$, $W_i \in \mathbb{R}^p \subset \mathcal{X}$, the weight vectors (which is a set of real values) is assigned to each input vector $x(t) \in \mathbb{R}^p \subset \mathcal{X}$, a neuron $i^*(t) = \arg min_{\forall i} \left||x(t) - w_i(t)|\right|, i^*(t) \in \mathcal{A}$, where $||\cdot||$ denotes to the Euclidean distances between the weights and input vector, and *t* represent a discrete time step corresponding with the iteration of the algorithm. The vector of weights of the current winning neuron as well as the vectors of weights of the neurons in its neighbourhood are changed by the following equation:

$$W_i(t+1) = W_i(t)h(i^*, i; t)[x(t) - W_i(t)], \qquad (3.3)$$

where $0 < \alpha(t) > 1$, is the rate of learning and $h(i^*, i; t)$ is the function of weights which limit the neighbourhood of the winning neuron. Usually the choice for this function $h(i^*, i; t)$ is given by the Gaussian function:

$$h(i^*, i: t) = \exp(-\frac{\left||r_i(t) - r_{i^*}(t)|\right|^2}{2\sigma^2(t)}), \qquad (3.4)$$

where $r_i(t)$ and $r_{i^*}(t)$ are, respectively, represent the coordinates of the neuron i and neuron i^* in the output matrix, and $\sigma(t) > 0$ determines the radius of the neighbourhood function in time *t*. The variables $\alpha(t)$ and $\sigma(t)$ should both disappears with time to ensure that the convergence of the weights vector will reach the steady states. And it is given by:

$$\alpha(t) = \alpha_0 (\frac{\alpha_T}{\alpha_0})^{(t/T)} \; and \; \sigma(t) = \sigma_0 (\frac{\sigma_T}{\sigma_0})^{(t/T)}, \qquad (3.5)$$

where α_0 (σ_0) and α_T (σ_T) are, respectively, the initial and final values of (t) $(\sigma(t))$. Changing the weights is continuous until a steady state of global ordering of the vector of weights has been reached. After that, we can say that the map has converged.

Despite of the simple work of SOM, it has been applied to a various problems [20, 21, 22]. The use of the SOM for function approximation purposes become more famous and common among the researchers in the last years, especially in the fields of the datasets forecasting since the year of 1989 [23].

3.2.2 VQTAM Model

The VQTAM method is a temporally domain of a SOM-based associative memory technique which has been used by many researchers in order to learns the static of input/output mappings, especially in robotic areas [24]. The input value $x(t)$, into SOM method at time $t,$ is divided into two parts. The first one, which is called $x^{in}(t) \in \mathbb{R}^q$, will contains the data about the dynamic mappings' input in order to be learned. The second part, which is called $x^{out}(t) \in \mathbb{R}^q$, carries the data about the desired outputs of those mappings. The dimension of the weight vector of neuron $i, w_i(t)$ has some changes. And these changes are represented as in the following:

$$x(t) = \begin{pmatrix} x^{in}(t) \\ x^{out}(t) \end{pmatrix} and\ w_i(t) = \begin{pmatrix} w^{in}(t) \\ w^{out}(t) \end{pmatrix} \qquad (3.6)$$

Where $x^{in}(t) \in \mathbb{R}^q$ and $x^{out}(t) \in \mathbb{R}^q$ are, respectively, the parts of the weight vectors, which is responsible for storing information about input/output of the desired mappings. According to the chosen variables to be build, the vector $x^{in}(t)$ and the vector $x^{out}(t)$ can use the SOM for learning forward or inverse mapping. In equalization task, we are looking for, set $p > 1$ & $q = 1$, in order to apply the following definitions:

$$x^{in}(t) = [y(t)y(t-1) \ldots y(t-p+1)]^T \qquad (3.7)$$

$$x^{out}(t) = s(t) \qquad (3.8)$$

Where $s(t)$, is the transmitted sampled signal in time $t, y(t)$ is the responsible channel for carrying the outputs, p represent the order of the equalization, and T represent the transposing vectors. Throughout learning process, the winning neurons in time step t is detected based on $x^{in}(t)$:

$$i^*(t) = \arg\min_{i\in\mathcal{A}}\{\left|\left|x^{in}(t) - w_i^{in}(t)\right|\right|\} \tag{3.9}$$

In order to update the weight vector, both $x^{in}(t)$ and $x^{out}(t)$ are used:

$$w_i^{in}(t+1) = w_i^{in}(t) + \alpha(t)h(i^*, i{:}\, t)[x^{in}(t) - w_i^{in}(t)] \tag{3.10}$$

$$w_i^{out}(t+1) = w_i^{out}(t) + \alpha(t)h(i^*, i{:}\, t)[x^{out}(t) - w_i^{out}(t)] \tag{3.11}$$

Where $0 < \alpha(t) > 1$, is the learning rate and $h(i^*, i; t)$ represent varying Gaussian neighbourhood function as in Equation (3.4). In simple statements, the learning rule in Equation (3.10) plays as a quantizer of the vector of the preserving topology in the input’s space and the rule in Equation (3.11) have the same actions in the output’s space of the learning mappings. As the training process continuing, the SOM learn to associate the vector of the input prototypes w_i^{in} with the corresponding vector of output w_i^{out}. The associated memory of self-organizing map performed by the VQTAM could be used as a function approximator. To be more specific, when the SOM completes its training, its output $z(t)$ for a new input vectors is derived from the codebooks vector that have been learned $w_{i^*}^{out}$, like follows:

$$z(t) \equiv w_{i^*}^{out} \tag{3.12}$$

Where $w_{i^*}^{out}$ = $[w_{1,i^*}^{out}\ w_{2,i^*}^{out}\ \ldots w_{q,i^*}^{out}\ ]^{\ T}$,is the vector of weights of the current winning neuron $i^*(t)$. According to the channel equalization mission we are looking for, set $q = 1$. Thus, the outputs of the VQTAM equalizer will be a scaled version of Equation (3.12), given by:

$$z(t) = \hat{s}(t) = w_{1,i}^{out}(t) \tag{3.13}$$

Where $\hat{s}(t)$ is a derived transmitted sampled signal at time step *t*. It could requires many neurons to generates a small estimated values of error: $(t) = s(t) - z(t) = s(t) - w_{1,i^*}^{out}(t)$, when approximating proceeds in mappings process.

3.2.3 Building Local-RBF as a Filter from the VQTAM

The VQTAM model itself can be used as a function approximator. However, as we mentioned before, it is a quantizer method for essential the vectors, and it may requires a huge number of neurons in order to reach to the generalization accuracy. To improve SOM's performances in forecasting, we propose a simple RBF model based on a trained VQTAM.

3.2.4 A local RBF Model

Let us assume that the trained VQTAM has q number of neurons, a general RBF network contains a q Gaussian function and *M* outputs (number of neurons) could be used for building over a learned input/output vector of codebooks, w_i^{in} and w_i^{out} , like the following:

$$z(t) = \frac{\sum_{i=1}^{q} w_i^{out}\, G_i(x^{in}(t))}{\sum_{i=1}^{q} G_i(x^{in}(t))} \tag{3.14}$$

where $z(t) = [z_1(t)\; z_2(t) \ldots z_M(t)]^T$, represent the output vector, $w_i^{out} = [w_{1,i}^{out}\;\; w_{2,i}^{out} \ldots w_{M,i}^{out}]^T$ will represent the vector of the connecting weights, which is responsible to connect the *i*th basis function with the *M* unit of outputs, and $G_i(x^{in}(t))$ is defined as the responding basis functions with the current vectors of input, $x^{in}(t)$:

$$G_i\left(x^{in}(t)\right) = \exp\left(-\frac{\left\|x^{in}(t)-w_i^{in}\right\|^2}{2\gamma^2}\right) \tag{3.15}$$

Where w_i^{in}, represent the core of *i*th basis functions, and $\gamma > 0$ is the radius (spreads). Notice that in Equation (3.14), if all vectors of the q codebooks will be used to derive the associated outputs. We will refers the RBF as the Global-RBF (GRBF) model, and ignore the local nature of the Gaussian functions.

However, in Local-RBF, which we are interested in, only a small part of the vector of the input space is used to derive the outputs of the mapping for each vector of inputs. In the subject of the VQTAM architectures, localized of the model means that it needs only $1 < K < q$ variables for setting up the basis functions' centres and the hidden to the output vectors of weights of a RBF structure. For this purposes, our suggestion is to use the vector of the prototypes of the first K winning neurons $\{i_1^*(t), i_2^*(t), \dots, i_K^*(t)\}$. Thus, the derived output is now given by:

$$z(t) = \frac{\sum_{K=1}^{K} w_{1,i_K^*}^{out}\, G_{i_K^*}(x^{in}(t))}{\sum_{K=1}^{K} G_{i_K^*}(x^{in}(t))} \tag{3.16}$$

We refer to the Local-RBF as the KRBF structure. Usually when we train two-phases of RBF, the centres of the basis function are first ones whom defined by clustering of the x^{in} vector through (example: the K-means clustering) [25]. Then the weights of hidden to output layer should calculated by using the L.M.S rules (or the pseudo inverse methods) [26, 27]. In the KRBF model, it needs only one phase of learning, where the clustering of the self-organizing map is implemented on the pairs of input/output together $\{x^{in}(t), x^{out}(t)\}$.

3.3 ARIMA

The **A**uto **R**egressive **I**ntegrated **M**oving **A**verage (ARIMA) model forecasts the future values in a datasets by taking a linear combination of its own past values, past errors (also called shocks or innovations), and current values. Box & Jenkins first popularized the (ARIMA) approach, and they refer it as Box and Jenkins model. Box and Tiao (1975) [32] made some discussion about the model of the general transfer function which is employed by the (ARIMA) procedure, When an ARIMA model involves another dataset and dealing with it as input variables, the ARIMA model will called as ARIMAX model. Pankratz (1991) [33] defines the ARIMAX model as dynamic regression.

3.3.1 The Three Stages of ARIMA Modelling

The analysis studies that applied on (ARIMA), separate it into three phases, according to the phases mentioned by Box and Jenkins (1976) [34]. The IDENTIFY, ESTIMATE, and FORECAST words will represent those phases, which are described below:

1. In the IDENTIFY phase - detect the datasets and choose the ARIMA models for it. In the IDENTIFY phase the datasets will be read, may differencing them, and calculates the inverse, partial, and cross autocorrelations.
2. In the ESTIMATE phase - detect the ARIMA model in order to fit the datasets which have been specified in the IDENTIFY phase, and derives the parameters of that model.
3. In the FORECAST phase – predicts the future values of the datasets and create a confidence interval for those predicted values from the ARIMA model, which was generated by the previous ESTIMATE phase.

In the econometrics studies and statistics studies, specifically in the datasets studies, an (ARIMA) model is fitted to time series data either to better understand the data or to predict future points in the series (forecasting). If the time-series is pretended to have a long-range dependence, then *d* parameter could be represented by a non-integer value, and that is what we called it an autoregressive fractionally integrated moving average model (ARFIMA).

3.3.2 Autoregressive Fractionally Integrated Moving Average (ARFIMA)

The ARFIMA (p, d, q) model is used here as a statistical analysis tool on dataset y_t with long memory:

$$\Phi(L)(1-L)^d(y_t-\mu_t)=\Theta(L)\varepsilon_t,\ t=1,\ldots,T. \tag{3.17}$$

Where ε_t is the past values, $\Phi(L)=(1-\phi_1L-\cdots-\phi_pL^p)$ is the autoregressive polynomial, $\Theta(L)=(1+\theta_1L+\cdots+\theta_qL^q)$ is the moving average polynomial in the lag operator L *(also called a backward-shift operator);* and the μ_t will represent the mean value of the time series y_t, p and q are integers, *d* is real, and $(1-L)^d$ is the fractional difference operator, which is determined by the following expansion:

$$(1-\mathrm{L})^{\mathrm{d}}=\sum_{\mathrm{k}=0}^{\infty}\frac{\Gamma(\mathrm{k}-\mathrm{d})L^k}{\Gamma(-d)\Gamma(k+1)} \tag{3.18}$$

Where $\Gamma(.)$ represent the gamma function, the ARMA-part of the model it will be invertible and stationary when all roots of $\Theta(z)=0$ and $\Phi(z)=0$ are outside of the unit circle. If the stability condition is reached, then an ARMA (p, q) model will have a MA (∞) representation. In addition, $\Theta(z)=0$ and $\Phi(z)=0$ do not have common roots. We say that:

$$z_t = y_t - \mu_t \tag{3.19}$$

If $d \in (-0.5\,, 0)$, the process is said to be exhibit intermediate memory (anti-persistence) or long-range negative dependence. And if $d \geq 0.5$, the process is non-stationary and have an infinite variance. If $d = 0$, the process is short memory. The process z_t is covariance stationary if $d < 0.5$, see [35]. The auto covariance function γ_k of an ARFIMA (p, d, q) process disappears hyperbolically: $\gamma_k \sim ck^{2d-1}, k \to \infty$, where c denotes a finite nonzero constant. We propose that $d > -1$, which makes the process z_t invertible, see [36].

The effecting of the past values of disturbance process follows a geometric lag, damping off to non-zero values quickly, so we should have a MA(q) model, which has the an enough memory of exactly periods, in order to make the effect of the moving average dies off. The convergence of the mean squared error of the AR (∞) is based on one-step-ahead of forecasting, $MSE(\hat{z}_{t|T})$, to the innovation variance σ_ε^2 as $T \to \infty$. The AR(∞) represented by z_t , is defined as:

$$z_t = \sum_{j=1}^{\infty} \pi_j z_{t-j} + \varepsilon_t \tag{3.20}$$

3.4 Clustering

Clustering, also called ‘segmentation’, which have been used to cluster the huge volume of the time series, and the goal from this process is to find a harmonic groups of data, which could be used in different analysis and studies. Such a requirements of this type of processing makes it necessary in a wide range of subjects, especially in the communities of sciences, medicine and marketing, where the human factors plays an important role but it will be not easy to understand. In clustering, the specific dependent variables are absence, and it is not easy to compare two kinds of clustering objectively [37].

3.4.1 Definition of Clustering

Clusters are combinations of objects that have similar properties, which are separated from objects that have totally different properties (resulting in internal homogeneity and external heterogeneity) [37]. There are two kinds of clustering, hierarchical clustering and the non-hierarchical one. (See Figure 3).

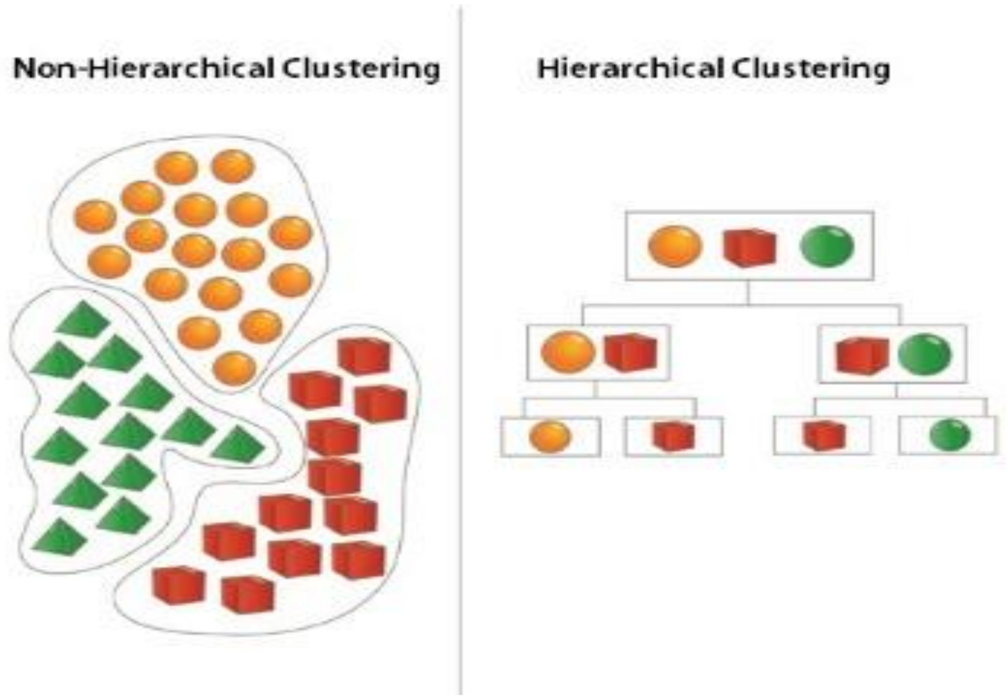

Figure 3: Types of Clustering.

3.4.2 K-means Clustering

The main goal of the K-means clustering is to partition m variables into k groups (or clusters) where each variables follows the nearest cluster and according to the minimum value of average. K-means clustering usually start by a single cluster and the average of data will represent the centre. This cluster is then divided into 2 parts, and the average of the new clusters also trained. If the determined number of clusters is not with a power of two, then the nearest one is chosen. Then the clusters with the least important are eliminated and the remaining clusters are again input to the training process in order to get the final clusters [38].

The main steps of the k-means clustering algorithm are:

1. Select *k*, which represent the number of clusters (groups).
2. Select the starting point of *k* for using it as an initial estimator to derive the centroids of the clusters.
3. Test all values in the loaded datasets and assigning them to the centroid.
4. When each value is followed a certain cluster, recompute the new *k* centroids.
5. The steps 3 and 4 are repeatedly executed until no value changes its centroid, or until the maximum number of observations through the time series is done.

Before the clustering procedure can be applied, actual data samples (i.e., indexes) are collected from the stock markets. Then normalize the data throughout normalization process, which is a method for organizing data elements in a database into tables, to avoid the duplication of data, insert, delete and update anomaly.

3.4.3 Decision-making Procedure

Every decision-making process produces a final choice [40]. The output can be an action or an opinion of choice. Therefore, the decision-making is one of the daily activities of any human being. However, when it comes to business fields, the decision-making it will be a habit and a process as well. Effective and successful decisions will make the company profit more, and in the other hand, the unsuccessful ones make it losses. Therefore, corporate decision-making process is the most critical process in any organization.

In the decision making phase in our research, we choose the results of four prediction techniques denoted by R^*, then inter those values in k-means clustering to organize them in a set of groups, in our study we choose five clusters (by experiment) which means that we will have five groups of data and of course five centroids.

$$R^* = \begin{bmatrix} r^*_{1,m1} & r^*_{1,m2} & r^*_{1,m3} & r^*_{1,m4} \\ \vdots & \vdots & \vdots & \vdots \\ r^*_{n,m1} & r^*_{n,m2} & r^*_{n,m3} & r^*_{n,m4} \end{bmatrix} \quad (3.21)$$

Since we have prediction results of two years, then we got a730 rows by four methods, where *n*=730, *m1, m2, m3 and m4* are represent the four prediction techniques, and $\boldsymbol{r}^*$ is the predicted value. After this process, calculate the estimated error *E* for each value in each row, in order to have 730 rows by four of error values, which represent the error of each value in each method. According to the following equation:

$$E = |\, R - R^*| \quad (3.22)$$

Where *R* represent a730 values from the original dataset, repeated for four methods, by applying Equation (3.22), we got:

$$E = \begin{bmatrix} e_{1,m1} & e_{1,m2} & e_{1,m3} & e_{1,m4} \\ \vdots & \vdots & \vdots & \vdots \\ e_{n,m1} & e_{n,m2} & e_{n,m3} & e_{n,m4} \end{bmatrix} \quad (3.23)$$

Where e represent the error value between the original and predicted value, then calculate the minimum error in each row of *E*.

$$E_{min} = \min_{n}(E) \quad (3.24)$$

By the end of k-means clustering, we will have a number of values following a certain centroid, for determining the values of each centroid; calculate the absolute squared distance between each value in each row of R^* and the five centroids according to the Equation (3.25). And the centroid with the minimum distance is the one who represent that row of values.

$$C_{r^*,i} = \min\{||r_1^* - r_{c1}||, ||r_1^* - r_{c2}||, ||r_1^* - r_{c3}||, \dots, ||r_1^* - r_{c5}||\} \quad (3.25)$$

Where *i= (1, 2, 3… 5),* after the end of this process, we will have a new column of centroids, where each value represent the cluster centre of row. Now, this column of centroids is also represent as a cluster centre of error rows *E*, and in the whole 730 rows of error values, the five centroids will certainly repeated, so we should count for each centroid ($r_{c1} \dots r_{c5}$), how many it has been repeated with the four methods. And by the end of this step we will got a number of error's rows for each centroid, like the following example:

$$r_{c1} = \begin{array}{c} m1 \\ m2 \\ m3 \\ m4 \end{array} \begin{bmatrix} r_{c1,1} & \cdots & r_{c1,n} \\ r_{c1,1} & \cdots & r_{c1,n} \\ r_{c1,1} & \cdots & r_{c1,n} \\ r_{c1,1} & \cdots & r_{c1,n} \end{bmatrix} \quad (3.26)$$

Calculate mean of errors for each method inside the cluster centroids, then find the minimum one, which will tell us which one of those methods is the winner inside that cluster. And continue with this process for all centroids in order to find the minimum mean of each prediction method inside that cluster.

Then, decide which one is the best predictive method according to the majority voting (how many it wins inside the clusters). And to verify our decision, we will calculate the Normalized Mean Squared Error (NMSE) after this process and it should be less than the NMSE’s for each method.

Chapter 4

FORECASTING AND DECISION-MAKING

Finding a forecasting technique that has the ability to predict the future prices of stocks with good accuracy is the subject of this research. As we explained previously, this work focused on building three computational intelligence models, which have the ability to predict the future index of stock market. Moreover, this chapter will covers the results of those prediction techniques, starting with Radial basis function, K-nearest neighbourhood then ARFIMA, and focuses on the results of the decision-making procedure, which according to it we will decide which method is the best predictive one.

4.1 SOM-based RBF

As described in Chapter 2, we take five years of data of London stock market starts with 2008 to 2012, the data was pre-processed to fill the missing days. After that, it will be ready to inter to the RBF model. Next step was the creation of the RBF model, which is done by determining the parameters of the model like the window size of 20 (length of the time window) which represent a dimension of the input vector, were used for the extraction of the features that were used to train and test the models.

4.1.1 Running the RBF Program for the Experiment

The implementation of the experiment was conducted in a MATLAB^{TM} environment. The RBF model is created by the following steps:

- **Creating a Network**
    1. Select the number of inputs
    2. Select the number of hidden nodes
    3. Select the activation functions
- **Training the Network**
    1. Select the optimization algorithm
    2. Input the training and target data
    3. Choose the number of epochs for training
    4. Train the network
- **Testing the Network**

    The trained network ability is then evaluated by testing the model using the testing data set.

4.1.2 Experiment Results of RBF

The results of the experiment are presented in a graphical form and in a table that compares the prediction accuracies. The accuracy of the model constructed with four functions to select, is more than all the other models. The graphs in (figure 4) shows that RBF is able to follow the trend of the target prices.

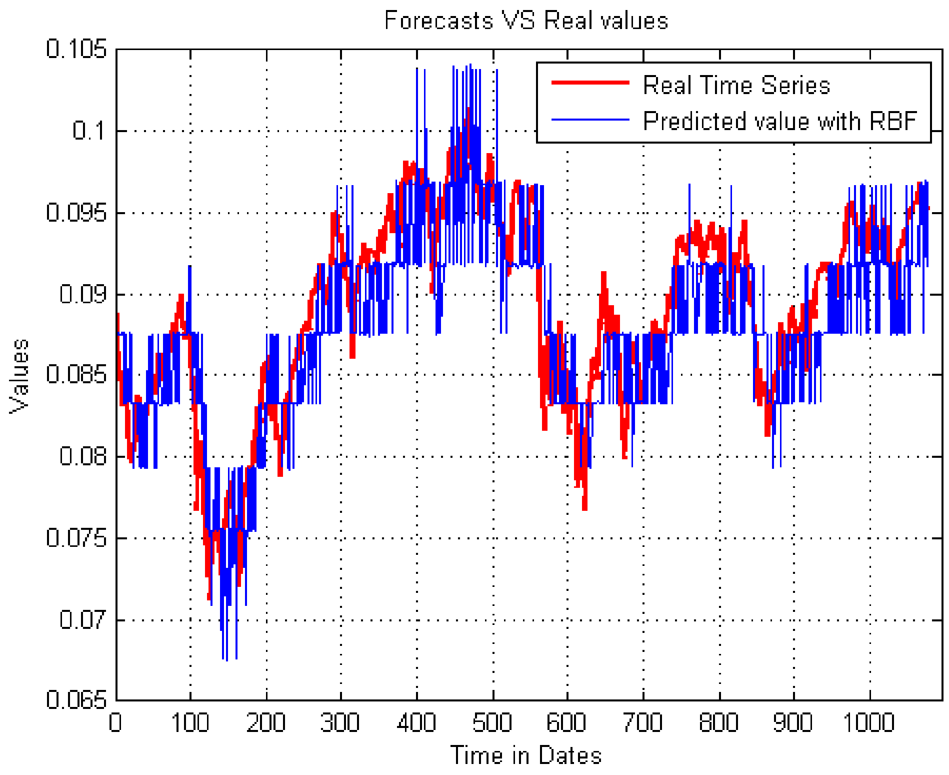

Figure 4: Forecast 3 Years (2010-2012) of LND with RBF Method

The Normalized Mean Squared Error have been calculated from estimated variance of the residuals ($e(t)$) and the estimated variance of the sequence of distorted samples, which is equal to 0.3309.

In Figure 5, we can notice the estimated error $e(t)$ of the forecasting the prices of London stock market.

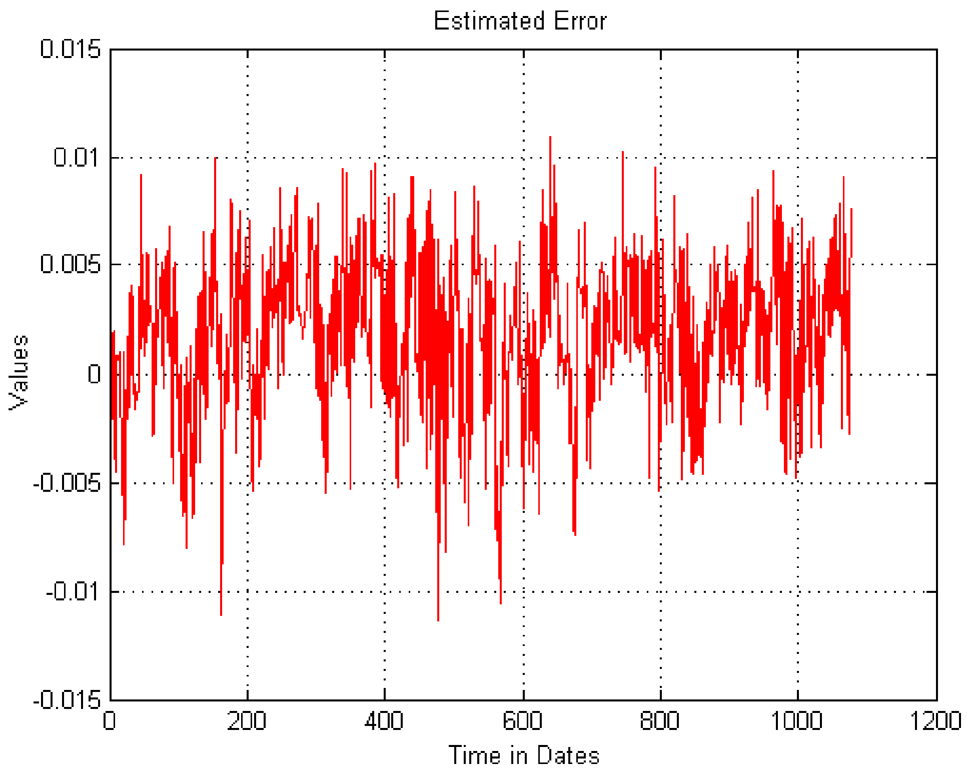

Figure 5: The Estimated Error of RBF During Forecasting of LND

Table 4 contains one week of the actual price, predicted one, and the estimated error values as a sample of London stock market.

Table 4: Actual Price, Forecasted Price and Error with RBF Method

Date	Actual price x100000	Predicted price x100000	Estimation error
01/01/2010	0.086206	0.087513	0.001274395
02/01/2010	0.086863	0.087541	-0.00057307
03/01/2010	0.087521	0.087668	-0.00161044
04/01/2010	0.088178	0.087486	-0.00187231
05/01/2010	0.088835	0.087522	-0.00235116
06/01/2010	0.089067	0.087622	-0.00289427
07/01/2010	0.088795	0.087563	-0.00232480

4.2 Nearest Neighbour Method

Chose a five years of data of London stock market, starts with 2008 to 2012, the data was pre-processed to fill the missing days. Then, the data will inter to the (K-Nearest Neighbour) model. Inside the nearest neighbour, we select the first two years of data as training set, to train the algorithm; Next step was the creation of the K-Nearest Neighbour model. Which is done by initiating the parameters of the model:

- Determine the length of training data set.
- Determine the Size of histories (embedding dimension) which represent a dimension of the input vector.
- Determine the number of nearest neighbours to use in the forecast's calculation.

4.2.1 Running the K-Nearest Neighbour Model

The implementation of the experiment was conducted in a MATLAB™ environment. The K-Nearest Neighbour model is created by the following steps:

- **Creating a network**
    1. Set the input data.
    2. Select the number of embedding dimension.
    3. Select the number of the nearest neighbour.
- **Training the network**
    1. Input the training and target data.
    2. Choose the number of epochs for training.
    3. Train the network.
- **Testing the network**

    The trained network ability is then evaluated by testing the model using the testing data set.

4.2.2 Experiment Results of K-Nearest Neighbour

In the following figures, the results of the experiments are presented for forecasting methods by displaying resulting errors in a graphical form and in tables to compare the prediction accuracies of each of the architectures.

Figure 6, shows the forecasts of three years of London stock market using the K-Nearest neighbourhood method with the absolute distance and correlation. This figure shows that K-Nearest Neighbour model is able to follow the trend of the target prices.

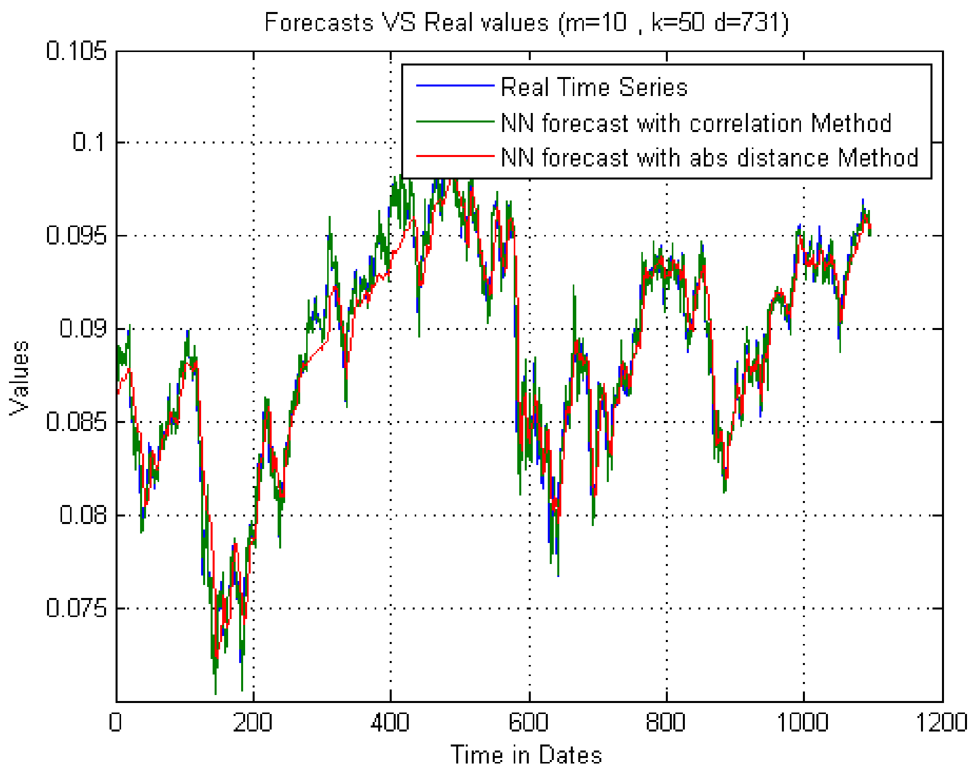

Figure 6: Forecasting of LND (2010-2012) with K-Nearest Neighbour Model

To clarify each method independently, we plot their results separately in order to notice the differences between them.

1- *Absolute Distance Method*

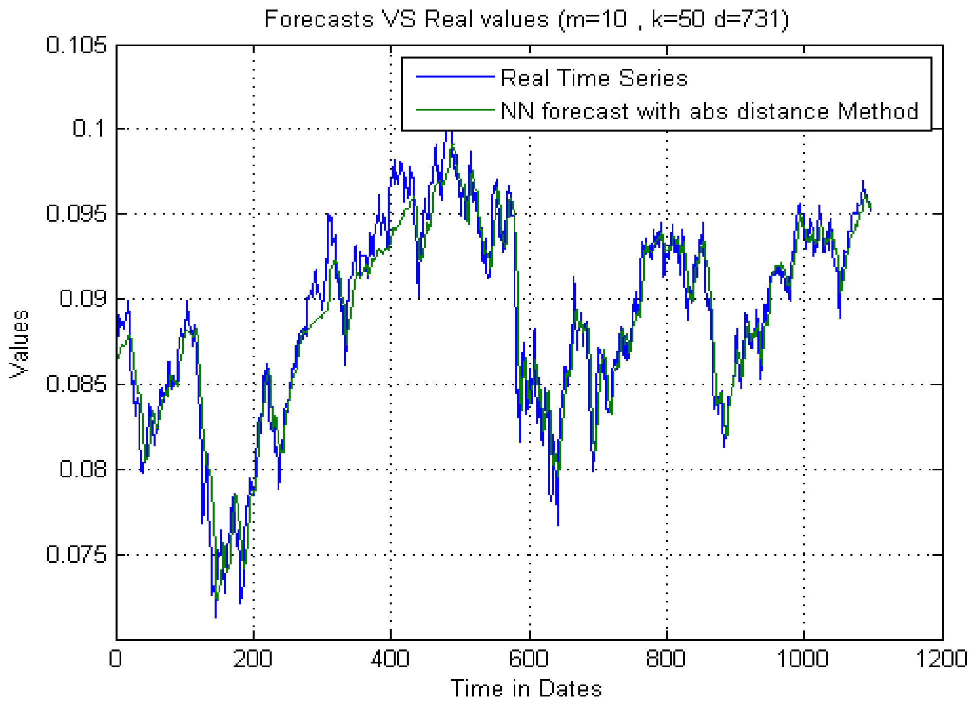

Figure 7: Forecasting of 3 Years (2010-2012) of LND Using Absolute Distance Method

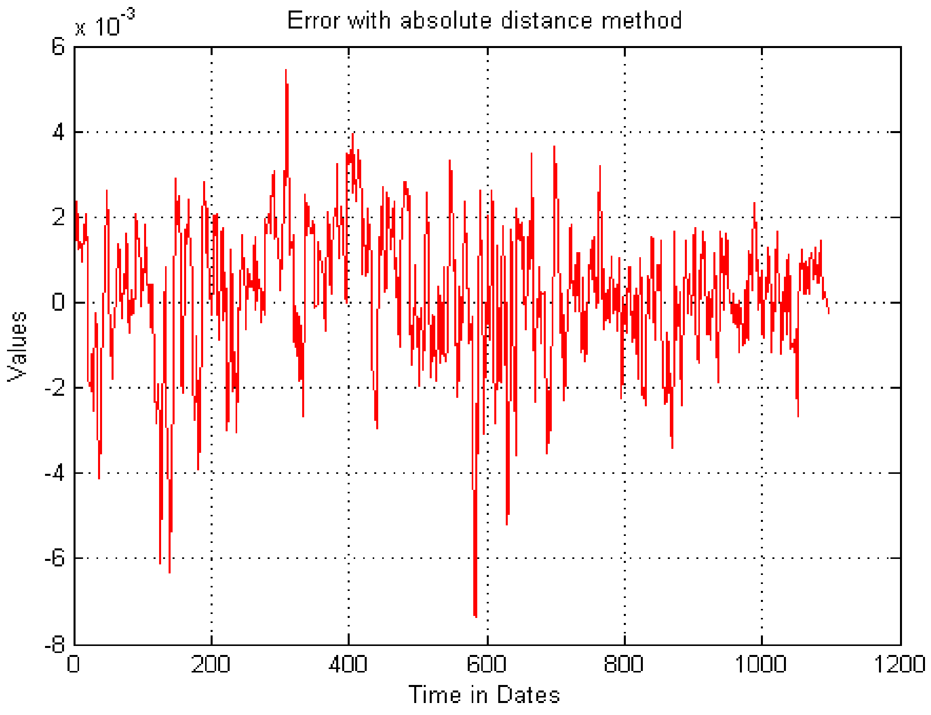

Figure 8: Estimated Error of the Forecasting with Absolute Distance Method

In Table 5, we got the prediction results from K-Nearest Neighbour with absolute distance method.

Table 5: Actual Price, Forecasted Price and Error with Absolute Distance Method

Date	Actual price X100000	Predicted price X100000	Estimated error
01/01/2010	0.086206	0.086379	0.000484
02/01/2010	0.086863	0.086377	0.001144
03/01/2010	0.087521	0.086461	0.001717
04/01/2010	0.088178	0.086549	0.002286
05/01/2010	0.088835	0.086715	0.002351
06/01/2010	0.089067	0.086982	0.001813
07/01/2010	0.088795	0.087077	0.001300

In addition, we found that the Normalized Mean Squared Error of K-Nearest Neighbour with absolute distance method is equal to 0.072822.

2- ***Correlation Method***

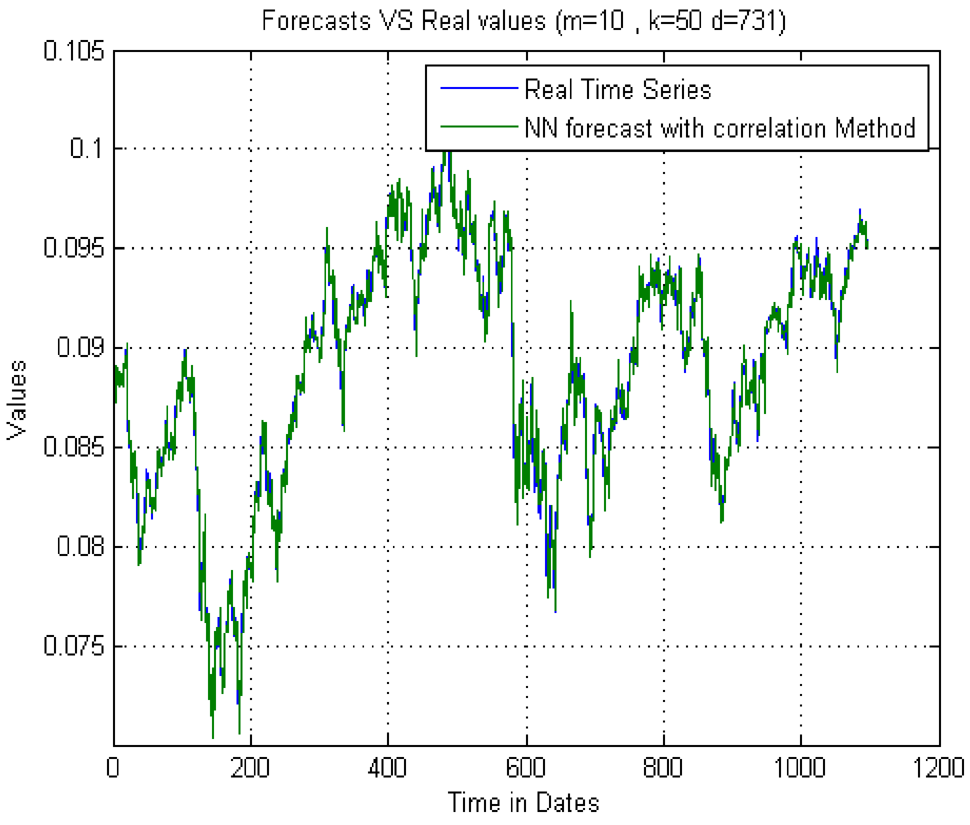

Figure 9: Forecasting of 3 Years (2010-2012) of LND Using Correlation Method

In Figure 10, we have the estimated errors of the forecasting process with correlation method, and in Table 6, we have a one week of the forecasting prices of k-Nearest Neighbour with correlation.

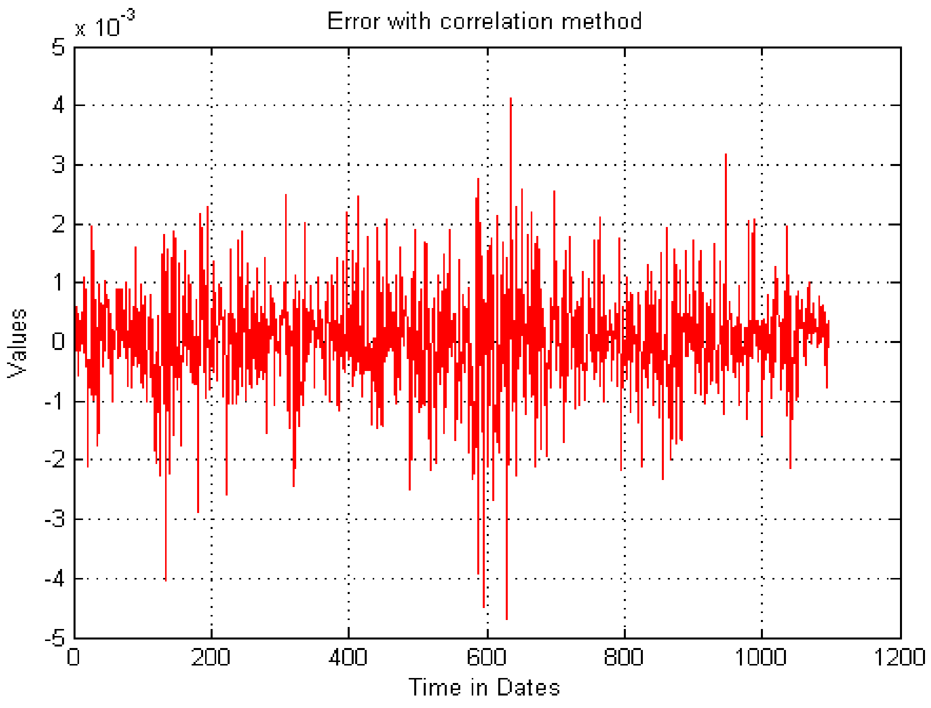

Figure 10: Estimated Error of the Forecasting Process with Correlation Method

Table 6: Actual Price, Forecasted Price and Error with Correlation Method

Date	Actual price	Predicted price	Estimated error
01/01/2010	0.086206	0.085975	0.000888611936
02/01/2010	0.086863	0.086734	0.000786535975
03/01/2010	0.087521	0.087759	0.000419161388
04/01/2010	0.088178	0.088996	-0.00016031025
05/01/2010	0.088835	0.089091	-2.4568358e-05
06/01/2010	0.089067	0.088327	0.000467787763
07/01/2010	0.088795	0.088949	-0.00057164938

And, the Normalized Mean Squared Error of K-Nearest Neighbour with correlation method is equal to 0.021962.

4.3 ARFIMA Model

Chose five years of data of London stock market starts with 2008 to 2012, then the data set was pre-processed to fill the missing days. Since the data is ready for ARFIMA. We will run the algorithm to work on it. Inside the ARFIMA, we select the first two years of data as training set, to train the algorithm; Next step is setting up the parameters of the model:

- Determine the length of training data set.
- Determine the dimension of the input vector.
- Initiate the matrix of predicted value.

4.3.1 Running the ARFIMA Model

The implementation of the experiment was conducted in a MATLAB™ environment. The ARFIMA model is created by the following steps:

- **Creating a network**
  1. Set the input data.
  2. Initiate the arfima function.
  3. Initiate the arma function.
  4. Initiate the estimate hurst exponent function.
- **Training the network**
  1. Input the data.
  2. Select the training period.
  3. Train the network.
- **Testing the network**

  The trained network ability is then evaluated by testing the model using the testing data set.

4.3.2 Experiment Results of ARFIMA

The results of the experiment whether the forecasting results or error results are presented in a graphical form and in a table, which compares the prediction accuracy.

In Figure 11, we had the forecasting of three years of London stock market using the ARFIMA method, and the figure also showed the ability of ARFIMA model to follow the trend of the target prices.

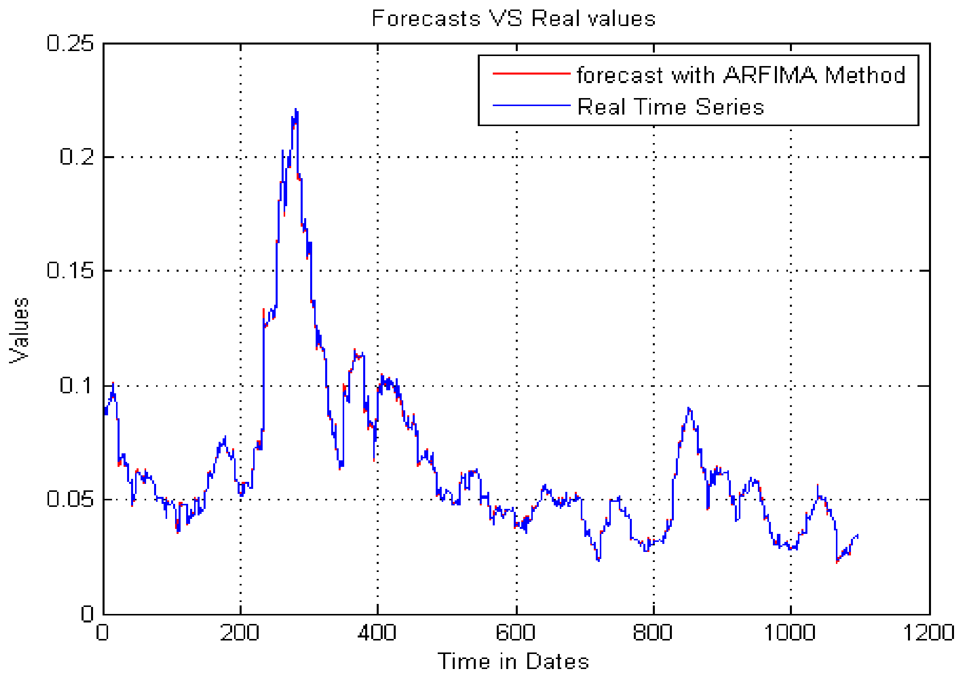

Figure 11: Forecasting of 3 Years (2010-2012) of LND with ARFIMA Method

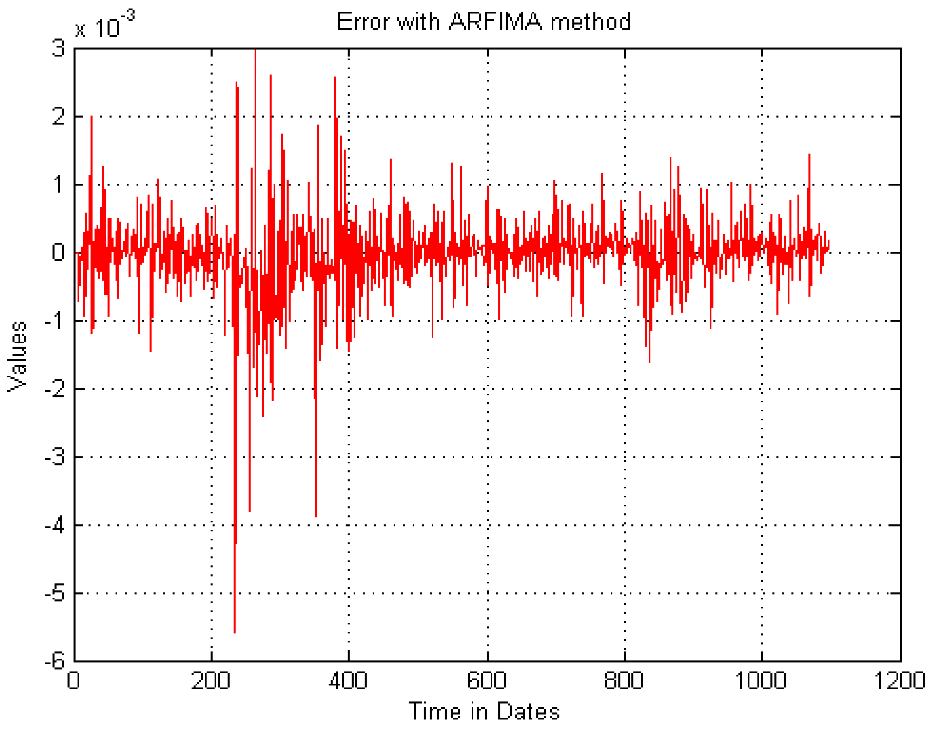

Figure 12: Estimated Error of the Forecasting Process with ARFIMA Method

In Figure 12, we had the estimated errors of forecasting process, and the Normalized Mean Squared Error of ARFIMA method is equal to 0.99084.

In Table 7, we have the actual price for one week as an example from real time series with the predicted one from ARFIMA method and its error values.

Table 7: Actual Price, Forecasted Price and Error with ARFIMA Method

Date	Actual price	Predicted price	Estimated error
01/01/2010	0.08620	0.08674	0
02/01/2010	0.08686	0.08722	0
03/01/2010	0.08752	0.09006	0
04/01/2010	0.08817	0.08713	0
05/01/2010	0.08883	0.08722	0
06/01/2010	0.08906	0.08730	0
07/01/2010	0.08879	0.09084	-0.00013

4.4 Decision-making with K-Means Clustering and Majority Voting

In this part, we will implement the formulas in section 3.4.3 in order to find the best predictive method.

In Figure 13, the result of K-means clustering showing the organizing process of two years of the predicted data, which are results from the four prediction techniques.

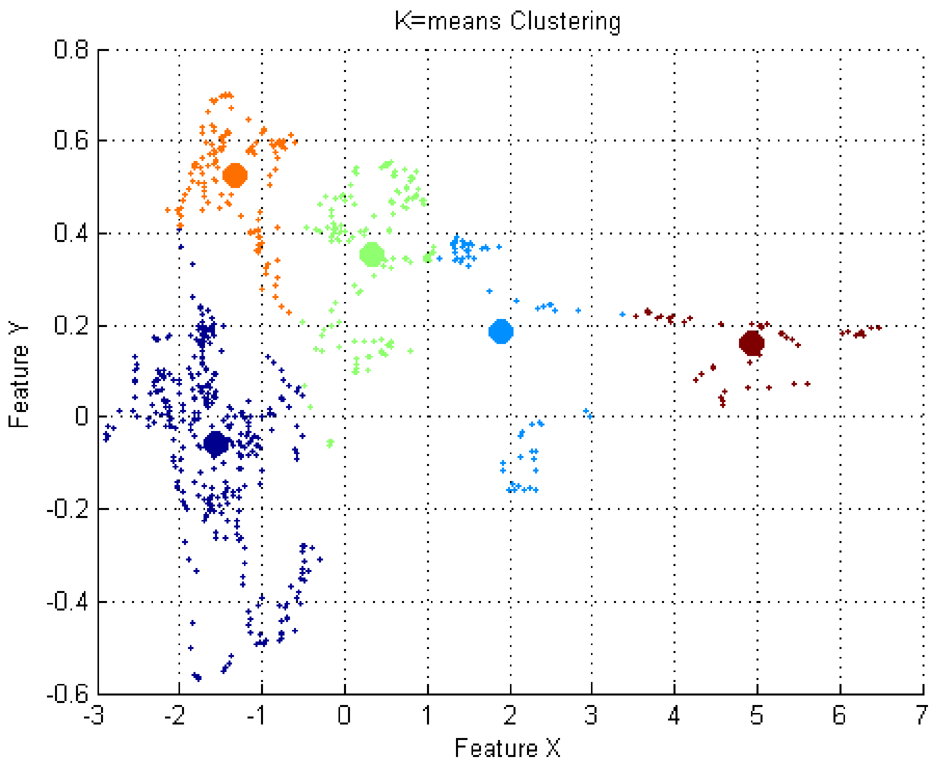

Figure 13: K-means Clustering of Predicted Data Set

From k-means clustering, we determined the centroids of the data set, and its positions which will be used later to calculate the absolute squared distances between the clustered values and its centroids. The following table shows the centroids, which are results from k-means clustering.

Table 8: The Centroids Values for Each Method

Radial Basis Function	K-Nearest Neighbour with Correlation	ARFIMA	K-Nearest Neighbour with Absolute distance
0.285713	0.35195	1.26139	0.29674
0.189153	0.21309	4.90843	0.16356
-0.16578	-0.21957	-1.2087	-0.19606
0.143105	0.22215	-1.85388	0.22430
0.402472	0.40552	-0.43465	0.38011

Since we have the real time series and the predicted values of models, calculate the estimated error of each model independently, according to the equation: $E = | R - R^*|$. After that, create a matrix of errors values and calculate the minimum error.

Table 9: A sample of the Error Values

RBF-error	Correlation-error	ARFIMA	Absolute dist.-error
0.0044784	0.0001782	0.0002975	-0.0002415
0.0006748	-7.4728e-05	-0.000342	-0.0007294
0.0051210	0.00088829	-0.000334	-9.4642e-05
0.0012205	-0.0001164	0.0002230	0.00018814
0.0054419	0.00031724	0.0001396	0.00029780
0.0015306	-0.0007832	-2.1447e-05	-0.0003005
0.0023384	0.00059860	4.31493e-06	-0.0002713

Table 10: The Sample of the Minimum Errors Considering the Best Prediction for Each Day

Minimum error
-0.0002415
-0.00072943
-0.00033496
-0.00011642
0.000139685
-0.00078321
-0.00027132

In Figure 14, we have a plot showing the estimated minimum error for two years of predicted index.

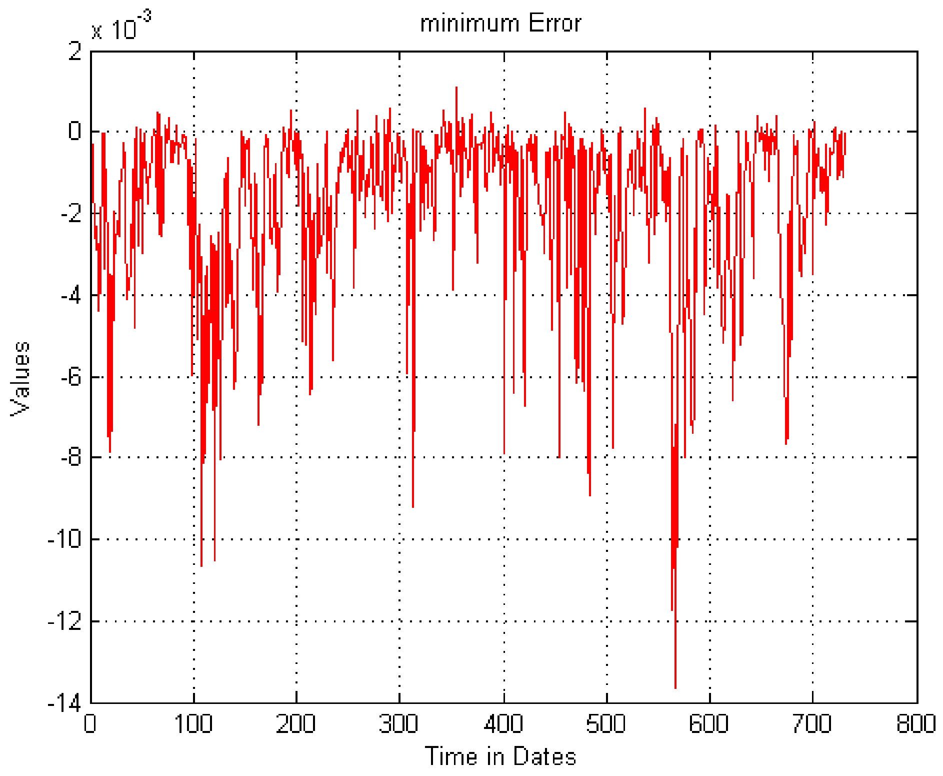

Figure 14: Estimated Minimum Error of 2 Years of Prediction

As we mentioned before in section 3.4.3, start to calculate the squared distance between the predicted value and all centroids according to formula $||r_i^* - r_{ci}||$, then select the minimum one to represent that row, in order to determine cluster centroid of each row in the error matrix. After that, start to calculate the mean error of each method for the five centroids, then find the minimum one for each cluster, as represented in Table 11.

Table 11: The Minimum Mean of Error of Each Method Inside the Clusters

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**
RBF	0.0005712	0.00060276	0.00007204	0.00037026	0.00048059
KNN_cor	0.0000398	0.00046351	0.00005150	0.00045150	0.00005063
ARFIMA	0.0000634	0.00633920	0.00053492	0.00004336	0.00083553
KNN_abs	0.0002650	0.00026179	0.00037973	0.00025329	0.00032551

From the above table, calculate the minimum value inside each cluster, which will represent the winner method inside that cluster, and according to the majority voting process, the method that wins more than the others, it will be the best predictive method.

After running the program, and by taking the majority voting from the five clusters, we found that the fourth method (Nearest Neighbour with correlation) is the winner method inside more than one cluster, which will make it the best predictive method.

By the calculation of NMSE (Normalized Mean Squared Error) inside the decision-making process, we found that it has a value less than the NMSE values for each method, and we found that the improvement in percent for error value is about 98%, which verify our procedure in order to find the best method. See Table 12.

Table 12: Comparison of the NMSE Results

Method	NMSE's
Radial basis function	0.3309
Nearest neighbour _ Cor.	0.02196
ARFIMA	0.99084
Nearest neighbour _ Abs.	0.07282
Decision-making procedure	*0.00043478*

4.4.1 Effect of Number of Clusters on the Success of Decision

In this section, we will put another result of the K-means clustering process in order to show the differences in selecting process of the number of clusters.

- **Six Clusters**

  Here in this experiment, we will use six clusters to group the forecasted results.

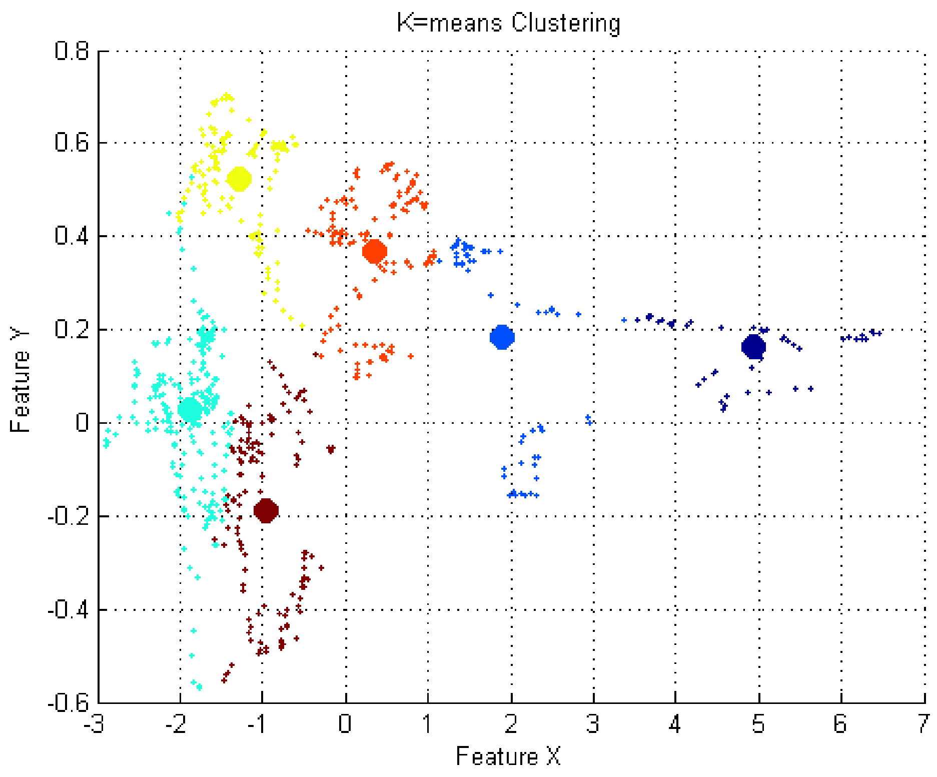

Figure 15: Experiment Results of 6 Clusters

In the following table, we have the results of calculating the minimum mean of errors value inside each cluster, which the decision is build based on them.

Table 13: The Mean of the Minimum Errors of Six Clusters

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**	**Cluster 6**
RBF	0.0005712	0.0006027	0.000072	0.0003702	0.0004805	0.00037026
KNN_cor	0.0000398	0.0004635	0.000051	0.0004515	0.0000506	0.00006515
ARFIMA	0.0000634	0.0063392	0.000534	0.0000433	0.0008355	0.00006349
KNN_abs	0.0002650	0.0002617	0.000379	0.0002532	0.0003255	0.00026479

As we noticed, when we use six clusters, the nearest neighbour with correlation is still the winner one with NMSE 0.00043664, but the ARFIMA became the second one.

- **Seven Clusters**

Here in this experiment, we will use seven clusters to group the forecasted results.

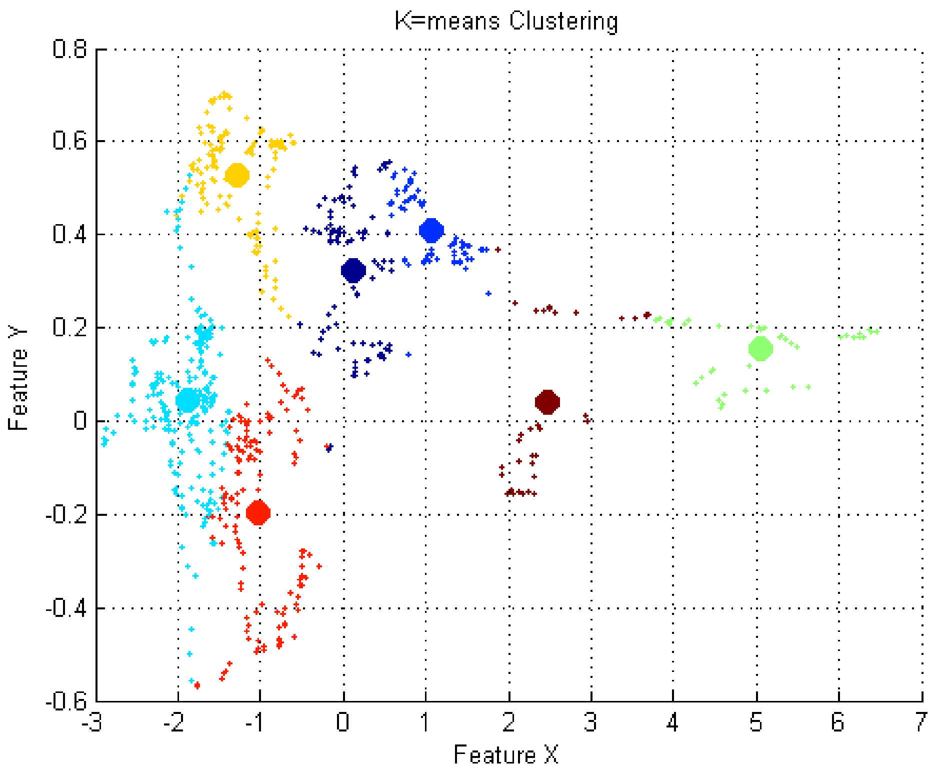

Figure 16: Experiment Results of 7 Clusters

Table 14: The Mean of the Minimum Errors of Seven Clusters

	Cluster 1	**Cluster 2**	**Cluster 3**	**Cluster 4**	**Cluster 5**	**Cluster 6**	**Cluster 7**
RBF	0.000571	0.000602	0.000072	0.000370	0.000480	0.0003626	0.000370
KNN_cor	0.000039	0.000463	0.000051	0.000451	0.000050	0.0000651	0.000510
ARFIMA	0.000063	0.006339	0.000534	0.000043	0.000835	0.0000534	0.005349
KNN_abs	0.000265	0.000261	0.000379	0.000253	0.000325	0.0002647	0.000047

In the experiment with seven clusters, the nearest neighbour with correlation is still the winner with NMSE 0.00043782, but the ARFIMA became equal to the nearest neighbour with absolute distance in the second place.

Table 15: NMSE Value for Each Group of Clusters

Number of clusters	Normalized Mean Square Error
Five clusters	0.00043478
Six clusters	0.00043664
Seven clusters	0.00043782

As a conclusion, we select the five group of clusters because it has the minimum NMSE value, and that will provide a decision-making procedure in order to find the sufficient number of groups for clustering process.

Chapter 5

CONCLUSION

Forecasting the stock price is a challenging problem than determining its direction, because building a good predictive model needs information-rich variables to be considered. A model that predicts the future behaviour of the market stock price provides a certain level of accuracy. In this thesis, four prediction techniques were applied to forecast the future price index of London stock market: Radial basis function, Nearest Neighbour with absolute distance and correlation, Autoregressive Fractionally Integrated Moving Average. This thesis proposes an approach to decide the best estimate of their future prices by majority voting on the clusters of K-means clustering.

The obtained results showed that all four techniques have considerable ability to forecast the price of the index. Another objective of the study was to determine which one of those prediction techniques is the best one in predicting the share index of market, which is done by taking the majority voting among the four techniques in order to find the most predictive method inside the market.

5.1 Future Work

The future research could focus on using macroeconomic variables, such as interest rate, GDP (gross domestic product), etc. as an input to the different models to determine if they have any predictive power and chose the best. A study can be conducted to test the effect of either the interest rate or the rate of inflation on the share index. This could help the decision makers or investors to understand how the market would behave if the interest rate increases or decreases. Another area of interest is to use the artificial intelligence techniques that can be adaptive and learn the data online. This would look like creating a methods that are capable of learning the new market patterns as they occurs in real time and still have a good predictive power.

Our future work includes the following tasks:

1. Using more economically significant features, such as oil prices, import and export values, interest rates and the growth rates of GDP (gross domestic product), to enhance the accuracy of forecasting.
2. Trying to improve the performance of the neural networks in the system using fuzzy logic and evolutionary algorithms.
3. Applying the proposed model to share price indices, interest rates and other financial time series.
4. Proposed an adaptive way in order to determine the best predictive model

REFRENCES

[1] Kolarik and Rudorfer G., “Time series forecasting using neural networks, department of applied Computer science,” Vienna University of Economics and Business Administration, no. 1090, pp. 2–6, 1997.

[2] Fama E., “The behaviour of stock market prices”, *The Journal of Business*, Vol. 38, pp. 34-105, 1965.

[3] Fama E., Foundations of finance. Portfolio Decisions and Securities Prices, New York: Basic Books, 1976.

[4] Fama E., “Efficient capital markets: a review of theory and empirical work,” Journal of Finance, vol. 25, pp. 383–417, 1970.

[5] Jensen M., “Some anomalous evidence regarding market efficiency,” Journal of Financial Economics, vol. 6, pp. 95–101, 1978.

[6] Balvers R., Cosimano T., and McDonald B., “Predicting stock returns in an efficient market,” Journal of Finance, vol. 55, pp. 1109–1128, 1990.

[7] Allan Borodin, Ran El-Yaniv and Vincent Gogan. Can We Learn to Beat the Best Stock. Journal of Artificial Intelligence Research 21, pp. 579-594, 2004.

[8] Stefan Zemke. Data Mining for Prediction. Financial Series Case. The Royal Institute of Technology, ISBN: 917283613-X, Sweden 2003.

[9] Enke D. and Thawornwong S, The adaptive selection of financial and economic variables for use with artificial neural networks. Neurocomputing 56, pp. 205-232, 2004.

[10] Lo A.W., MacKinlay A.C., Stock market prices do not follow random walks: evidence from a simple specification test, University of Pennsylvania, Review of Financial Stud. No.1, pp. 41-66, 1988.

[11] Azoff, E.M., Neural Network Time series Forecasting of financial markets. John Wiley and Sons Inc. New York, NY, USA, ISBN: 0471943568, 1994.

[12] Bachelier L., "Théorie de la spéculation", Annales Scientifiques de l'École Normale Supérieure 3 (17): pp. 21–86.

[13] Fama, Eugene F., Random Walks in Stock Market Prices. Financial Analysts Journal 21, pp. 55-59, September/October, 1965.

[14] Makridakis S. and Hibon H., "Accuracy of forecasting: An Emperical Investigation" J. Roy Statist. Soc., no. 8, pp. 69–80, 1992.

[15] Reid D.J., "A comparative study of time series prediction techniques on economic data" PhD thesis, Department of Mathematics, University of Nottingham, 1969.

[16] Newbold P. and Granger C.W.J., “Experience with forecasting univariate time series and the combination of forecasts (with discussion)” Journal of Royal Statistical Society, no. A 137, pp. 131–165, 1974.

[17] "Market highlights for first half 2010". World Federation of Exchanges. Retrieved 18 August, 2010.

[18] Fernandez, R. F., Rivero, S. S., and Felix, J. A. Nearest Neighbour Predictions in Foreign Exchange Markets. Working Paper, 05, FEDEA, 2002.

[19] Fernandez, R. F., Rivero, S. S., and Garcia, A. M. An empirical evaluation of non-linear trading rules. Working paper, 16, FEDEA, 2001.

[20] Flexer, A., on the use of self-organizing maps for clustering and visualization. Intelligent Data Analysis, 5(5), pp. 373–384, 2001.

[21] Kohonen, T. K., Oja, E., Simula, O., Visa, A., & Kangas, J., Engineering applications of the self-organizing map. Proceedings of the IEEE, 84(10), pp. 1358–1384, 1996.

[22] Oja, M., Kaski, S., & Kohonen, T., Bibliography of self-organizing map (som) papers: 1998–2001 addendum. Neural Computing Surveys, 3, pp. 1–156, 2003.

[23] Ritter, H., Martinetz, T., & Schulten, K., Topology conserving maps for learning visuo-motor-coordination. Neural Networks, 2(3), pp. 159–168, 1989.

[24] Barreto, G. A., Ara´ujo, A. F. R., & Ritter, H. J., Self-organizing feature maps for modelling and control of robotic manipulators. Journal of Intelligent and Robotic Systems, 36(4), pp. 407–450, 2003.

[25] MacQueen, J., some methods for classification and analysis of multivariate observations. In L. M. L. Cam and J. Neyman (Eds.) Proceedings of the 5th Berkeley symposium on mathematical statistics and probability, vol. 1, pp. 281–297, 1967.

[26] Principe, J. C., Euliano, N. R., & Lefebvre, W. C., Neural and adaptive systems: Fundamentals through simulations. New York, NY: JohnWiley & Sons, ISBN: 0471351679,2000.

[27] Haykin, S., Neural networks: A comprehensive foundation. Englewood Cliffs, NJ: Macmillan Publishing Company, ISBN:0023527617, 1994.

[28] http://finance.yahoo.com/ Yahoo! Finance is an Internet web site sponsored by yahoo that provides financial news and information.

[29] http://www.mplans.com/articles/what-is-a-market-forecast/#ixzz2IYVSYzoe Mplans is part of a network of Palo Alto Software sites dedicated to helping small business startups, entrepreneurs and marketers plan for business success.

[30] Farmer, D., Sidorowich, J. Predicting chaotic time series. Physical Review Letters, vol. 59, pp. 845-848, 1987.

[31] Fernandez, R. F., Rivero, S. S., and Garcia, A. M. Using nearest neighbour predictors to forecast the Spanish Stock Market. Investigaciones Económicas, vol. 21, pp. 75-91, 1997.

[32] Box, G. E. P. and Tiao, G. C., Comparison of Forecast and Actuality. University of Wisconsin, Madison, U.S.A, Appl.Statist, pp. 195-200, 1975.

[33] Pankratz, A., Forecasting with dynamic regression models. (John Wiley and Sons, New York), ISBN: 0471615285, 1991.

[34] Box, G. E. P. and Jenkins, G. M., Time Series Analysis: Forecasting and Control, Revised Edition, San Francisco: Holden Day, 1976.

[35] Hosking, J. R. M., Fractional differencing. Biometrika 68, pp. 165-176, 1981.

[36] Odaki, M., On the invertibility of fractionally differenced ARIMA processes. Biometrika 80, pp. 703-709, 1993.

[37] Stéphane, T., Data Mining and Statistics for Decision Making, First Edition. Published in 2011 by John Wiley & Sons Inc. ISBN: 9780470688298, 2011.

[38] Nanda, S.R., Mahanty, B. and Tiwari, M.K., Clustering Indian stock market data for portfolio management. Department of Industrial Engineering and Management, Indian Institute of Technology, Kharagpur, West Bengal, India, Elsevier (37), pp. 8793-8798, 2010.

[39] http://www.techopedia.com/definition/14650/data-preprocessing aim to provide insight and inspiration to IT professionals, technology decision-makers.

[40] James R., Human Error. Ashgate. ISBN: 1840141042, 1990.

[41] Box G., and Jenkins G., “Time series analysis: Forecasting and control”, Holden-Day, San Francisco, 1970.

[42] Schaer C., “Curve fitting: A pernicious illusion,” AIMA Newsletter, 2001.

[43] Campbell H. R., “Predictable risk and returns in emerging markets,” Review of Economic Studies, vol. 8, pp. 773–816, 1995.

[44] Abhyankar A., Copeland L., and Wong W., “Uncovering nonlinear structure in real-time stock market indexes: the s and p 500, the dax, the nikkei 225, and the ftse-100, j.,” Business Econ. Statist., vol. 15, pp. 1–14, 1997.

[45] Ryden T. T. T., and Asbrink S., “Stylized facts of daily return series and the hidden markov model,” Journal of Applied Econometrics, vol. 13, pp. 217–244, 1998.

[46] Patel P. and Marwala T., “Neural networks, fuzzy inference systems and adaptive-neuro fuzzy inference systems for financial decision making,” ICONIP 2006, Part III, LNCS, vol. 4234, pp. 430–439, 2006.

[47] Patel P. and Marwala T., “Forecasting closing price indices using neural networks,” IEEE conference on Systems, Man and Cybernatics October 8-11, Taipei, Taiwan, pp. 2351–2356, 2006.

[48] Van Eyden R., The Application of Neural Networks in the Forecasting of Share prices. Finance and Technology Publishing, 1996.

[49] Kim K., “Artificial neural networks with evolutionary instance selection for financial forecasting,” Expert Systems with Applications, vol. 20, pp. 519–526, 2006.

[50] Campbell H. R., “The world price of covariance risk,” Journal of Finance, vol. 46, pp. 111–157, 1991.

[51] Schwert W., “Stock returns and real activity: a century of evidence,” Journal of Finance, vol. 45, pp. 1237–1257, 1990.

APPENDIX

Matlab code

D1. Matlab code for KNN method

```
% KNN method

% This model is a time series predictor using two different methods and then plot the
forecasts versus real points

addpath('c:\Users\Black Bird\Desktop\nearest neighbour\NN_FEX\london\m_Files');

[x] = xlsread('data LND_ret.xlsx'); % Load data from file
d=731;     % Defines where to start the forecasts (and also the training data (1:d-
1))
m=10;       % Size of histories (embeding dimension)
k=50;       % Number of nearest neighbors to use in the forecast's calculation

method_1='correlation';
method_2='absolute_distance';

[OutSample_For_Corr,InSample_For_Corr,InSample_Res_Corr]=nn(x,d,m,k,method_1);

[OutSample_For_Abs,InSample_For_Abs,InSample_Res_Abs]=nn(x,d,m,k,method_2);

plot([x(d+1:end),InSample_For_Corr,InSample_For_Abs]);
xlabel('Time in Dates');
ylabel('Values');
title(['Forecasts VS Real values (m=',num2str(m),' , k=',num2str(k),'
d=',num2str(d),')']);
legend('Real Time Series','NN forecast with correlation Method','NN forecast with abs
distance Method');
grid;

%%%%% calculate the error valies %%%
[x1] = xlsread('data LND_ret_2.xlsx');
error_cor=x1-InSample_For_Corr;  % prediction error cor
error_abs=x1-InSample_For_Abs;  % prediction error abs

figure;
plot(error_cor,'r-','linewidth',1);
xlabel('Time in Dates');
ylabel('Values');
title('Error with correlation method');
grid on;
NMSE_cor=var(error_cor)/var(x1);

figure;
plot(error_abs,'r-','linewidth',1);
xlabel('Time in Dates');
ylabel('Values');
title('Error with absolute distance method');
grid on;
NMSE_abs=var(error_abs)/var(x1);

 %The main functions are:


nn.m
%{

NN -  Creates forecasts of a time series on t+1 using nearest neighbour
  algorithm.


      Usage: [OutSample_For,InSample_For,InSample_Res]=nn(x,d,m,k,method,n)

      INPUT:

              x - The time series to be forecasted (the function was
              originally made for stock price, but it accepts any kind of
              time series).

              d - the observation where the InSample forecasts will
              start. It also defines the training period of the
              algorithm. For example, if lenght(x)=500 and d=400, the
              values of 1:400 will be the training period for the
              forecasted value of 401. For the forecast of 402, the
              training period is 1:401, meaning that each time a new
              observation is available, the algorithm adds it to the
              training period. Please notes that the parameter d doesn't
```

```
                have any effect on the out of sample forecasts.

                m - Embedding dimension (size of the histories).

                k - The number of nearest neigbours to be used in the
                construction of the forecasts.

                method - Defines the method to calculate the forecasts. Can
                take values 'absolute_distance' (default) or 'correlation'.
                More details about each method can be found at word
                document.

                n - Number of outsample Forecasts. The default value is 0.

        OUTPUT:
                OutSample_For - A vector with the out of sample forecasted
                values of the time series. The length of [OutSample_For] is
                n. Notes that the out sample forecasts are build with the
                whole modeled series x.

                InSample_For  - A vector with the in sample forecasted
                values of the time series. The length of [InSample_For] is
                length(x)-d

                InSample_Res  - A vector with the out-of-sample residues
                from the in-sample forecasts.


    Ahmed Ibrahim
    Email: neo.neo700@gmail.com
    Department of Computer Engineering.
    Eastern Mediterranean University / N.Cyprus.
    Created: January/2013
    Last Update: May/2013

    This is the algorithm involved on the use of the non-linear forecast of
    asset's prices based on the nearest neighbour method, Perlin(2006). The
    rotines were build according to the work of Rodriguez, Rivero and
    Artilles (2001).


[OutSample_For,InSample_For,InSample_Res]=nn(Ibov,1000,3,50,'absolute_distance',5)


%}

function [OutSample_For,InSample_For,InSample_Res]=nn(x,d,m,k,method,n)

if (nargin<4)
    error('Its missing arguments.')
end

if d>=length(x)
    error('The value of d must be between 1 and length(x)-1')
end

if (nargin==4)
    n=0;
    method='absolute_distance';
    OutSample_For=[];
end

if (nargin==5)
    n=0;
    OutSample_For=[];
end

% Main Loop.

for v=0:length(x)-d-1;

     Series=x(1:d+v);

    [For]=nn_core(Series,m,k,method);

    InSample_For(v+1,1)=For;

    fprintf(1,['\nCalculating NN Forecast #',num2str(v+1)]);

end

disp(' ');

InSample_Res=x(d+1:length(x))-InSample_For;

if n~=0
    x2=x;
    for z=1:n
        [Out_For]=nn_core(x2,m,k,method);
        OutSample_For(z,1)=Out_For;
        x2=[x2;OutSample_For(z)];
    end
end

nn_cor
%{

DESCRIPTION:

Core function for nn.m

INPUT:
        x - Series to be modelled
        m - Embedding dimension (size of the histories)
        k - The number of nearest neigbours to be used in the construction
            of the forecasts
        method - The method to be used in the calculations


 OUTPUT:
        For_x - The one out of sample forecast of x


POURPOSE: This core function will create each forecast in nn.m

%}

function [For_x]=nn_core(x,m,k,method);


[n1,n2]=size(x);

switch method
    case 'correlation'

        %   Calculation of the correlations between the pieces of the time series.

        chunk = x(n1-m+1:n1,1);
        for i=0:n1-m-1;
            aba=corrcoef(chunk,x(n1-m-i:n1-i-1,1));
            mcorrel(n1-m-i,1)=aba(2,1);
        end;

        mcorrel2=abs(mcorrel);

        %   Find the k max abs correlation and also the piece related to such k
correlations.

        [sorted idx] = sort(mcorrel2);

        fullIdx = repmat( idx((end-k+1):end),1,m+1) + repmat([0:m],k,1);

        %   REgression to find the coefficents.

        s = x(fullIdx);

        Coefficients = regress(s(:,m+1),[ones(k,1),s(:,m:-1:1)]);

        c = ones(1,m+1);
        c(2:end) = x(n1+2 - [2:m+1]);

        For_x(1,1)=c*Coefficients;

    case 'absolute_distance'

        %   Calculation of the sum of distances between the pieces of the time
series.

        chunk = x(n1-m+1:n1,1);
        for i=0:n1-m-1;
            distance=sum(abs(chunk-x(n1-m-i:n1-i-1,1)));
            sum_distance(n1-m-i,1)=distance;
        end

        % Sort the distances and find the lowests

        [sorted idx] = sort(sum_distance,'descend');

        % Create a matrix with the indexes of the neighbors found

        fullIdx = repmat( idx((end-k+1):end),1,m+1) + repmat([0:m],k,1);

        % Grab the values of such neighbors
        s = x(fullIdx);

        % Calculate the forecast

        For_x(1,1)=mean(s(:,m+1));
end
```

D2. Matlab code for RBF method.

```
% method3_MATLAB implementation of a local SOM-based RBF time series predictor.
clear; clc; close all;

%--------------- Organize the training data ------------

[Close] = xlsread('data LND_ret_2.xlsx');  % Load the time series to be clustered

%% Building the input vectors from an univariate time series
p=20;       % Dimension of the input vector (length of the time window) 5
lap=p-1;      % Amount of overlapping between consecutive input vectors
Dw=buffer(Close,p,lap);   % Build the data vectors
if lap>0
     Dw=Dw(:,p:end)';  % Eliminate the first 'p-1' vectors with zeros)
else Dw=Dw';
end
Dw=fliplr(Dw);

Ynext=Dw(2:end,1);        % Target (one-day-ahead) values
Dw=[Ynext Dw(1:end-1,:)]; % Input regressor vectors for Ynext

Dw=Dw+0.01*randn(size(Dw));  % Add some gaussian noise to the data 0.01
         % SOM initialization and training
Mx = 16;             % Number of neurons in the X-dimension
My = 1;              % Number of neurons in the Y-dimension
msize = [Mx My];     % Size of 2-D SOM map
MASK=[0; ones(p,1)];
sMap = som_randinit(Dw, 'msize', msize,'lattice','rect','shape','sheet');
sMap = som_seqtrain(sMap,Dw,'radius',[floor(0.5*max(msize))
0],'sample_order','random','neigh','gaussian','trainlen',50);

%--------------- Organize the testing data ------------
[data_test] = xlsread('tested_dataln3.xlsx');   % Load testing time series
Dw=buffer(data_test,p,lap); % Build input data vectors

if lap>0   %0
     Dw=Dw(:,p:end)';  % get rid of the first 'p-1' vectors with zeros
else Dw=Dw';
end
Dw=fliplr(Dw);

Ynext=Dw(2:end,1);        % Target (one-day-ahead) values
Dw=[Ynext Dw(1:end-1,:)]; % Input regressor vectors for Ynext

Dw=Dw+0.01*randn(size(Dw));  % Add some gaussian noise to the data 0.01
[LEN_DATA DIM_INPUT]=size(Dw);  % Data matrix size (1 input vector per row)

K=4;     % Number of radial basis functions to select
s2=0.1;  % Spread of the basis functions 0.1
G=[];
w=[];
for t=1:LEN_DATA
    win = som_bmus(sMap,Dw(t,:),[1:K]);    % Find the K winning neurons
    for i=1:K
        act = norm(Dw(t,2:end)-sMap.codebook(win(i),2:end));
        G(i) = exp(-act*act/(2*s2*s2));   % output of the i-th basis function
        w(i) = sMap.codebook(win(i),1);   % hidden-to-output layer weight
    end
    Yhat(t) = dot(w,G)/sum(G);  % predicted value
    error(t)=Ynext(t)-Yhat(t);  % prediction error
end

% Plot target and predicted time series
figure;
plot(Ynext,'r-','linewidth',2); hold on; plot(Yhat,'b-');
xlabel('Time in Dates');
ylabel('Values');
title('Forecasts VS Real values');
legend(' Real Time Series',' Predicted value with RBF');
axis([0 1095 0.065 0.105]);
grid on;
hold off

figure;
```

```
plot(error, 'r-', 'linewidth',1);
% xlable('Time in Dates');
% ylable('Error values');
title('Estimated Error');
grid on;
% Normalized mean squared error
NMSE=var(error)/var(Ynext)
RMSE=sqrt(mean((error).^2));
```

Function1- SOM_RANDINIT Initialize a Self-Organizing Map with random values.

```
function sMap = som_randinit(D, varargin)

%SOM_RANDINIT Initialize a Self-Organizing Map with random values.

%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_randinit
%
% PURPOSE
%
% Initializes a SOM with random values.
%
% SYNTAX
%
%  sMap = som_randinit(D)
%  sMap = som_randinit(D,sMap);
%  sMap = som_randinit(D,'munits',100,'hexa');
%
% DESCRIPTION
%
% Initializes a SOM with random values. If necessary, a map struct
% is created first. For each component (xi), the values are uniformly
% distributed in the range of [min(xi) max(xi)].
%
% REQUIRED INPUT ARGUMENTS
%
%  D                  The training data.
%           (struct) Data struct. If this is given, its '.comp_names' and
%                    '.comp_norm' fields are copied to the map struct.
%           (matrix) data matrix, size dlen x dim
%
% OPTIONAL INPUT ARGUMENTS
%
%  argID (string) Argument identifier string (see below).
%  value (varies) Value for the argument (see below).
%
%% check arguments

% data
if isstruct(D),
  data_name = D.name;
  comp_names = D.comp_names;
  comp_norm = D.comp_norm;
  D = D.data;
  struct_mode = 1;
else
  data_name = inputname(1);
  struct_mode = 0;
end
[dlen dim] = size(D);

% varargin
sMap = [];
sTopol = som_topol_struct;
sTopol.msize = 0;
munits = NaN;
i=1;
while i<=length(varargin),
  argok = 1;
  if ischar(varargin{i}),
    switch varargin{i},
     case 'munits',     i=i+1; munits = varargin{i}; sTopol.msize = 0;
     case 'msize',      i=i+1; sTopol.msize = varargin{i};
                               munits = prod(sTopol.msize);
     case 'lattice',    i=i+1; sTopol.lattice = varargin{i};
     case 'shape',      i=i+1; sTopol.shape = varargin{i};
     case {'som_topol','sTopol','topol'}, i=i+1; sTopol = varargin{i};
     case {'som_map','sMap','map'}, i=i+1; sMap = varargin{i}; sTopol = sMap.topol;
     case {'hexa','rect'},          sTopol.lattice = varargin{i};
     case {'sheet','cyl','toroid'}, sTopol.shape = varargin{i};
     otherwise argok=0;
    end
  elseif isstruct(varargin{i}) & isfield(varargin{i},'type'),
    switch varargin{i}.type,
     case 'som_topol',
      sTopol = varargin{i};
     case 'som_map',
      sMap = varargin{i};
      sTopol = sMap.topol;
     otherwise argok=0;
    end
  else
    argok = 0;
  end
  if ~argok,
    disp(['(som_topol_struct) Ignoring invalid argument #' num2str(i)]);
  end
  i = i+1;
end

if ~isempty(sMap),
  [munits dim2] = size(sMap.codebook);
  if dim2 ~= dim, error('Map and data must have the same dimension.'); end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% create map

% map struct
if ~isempty(sMap),
  sMap = som_set(sMap,'topol',sTopol);
else
  if ~prod(sTopol.msize),
    if isnan(munits),
      sTopol = som_topol_struct('data',D,sTopol);
    else
      sTopol = som_topol_struct('data',D,'munits',munits,sTopol);
    end
  end
  sMap = som_map_struct(dim, sTopol);
end

if struct_mode,
  sMap = som_set(sMap,'comp_names',comp_names,'comp_norm',comp_norm);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% initialization

% train struct
sTrain = som_train_struct('algorithm','randinit');
sTrain = som_set(sTrain,'data_name',data_name);

munits = prod(sMap.topol.msize);
sMap.codebook = rand([munits dim]);

% set interval of each component to correct value
for i = 1:dim,
  inds = find(~isnan(D(:,i)) & ~isinf(D(:,i)));
  if isempty(inds), mi = 0; ma = 1;
  else ma = max(D(inds,i)); mi = min(D(inds,i));
  end
  sMap.codebook(:,i) = (ma - mi) * sMap.codebook(:,i) + mi;
end

% training struct
sTrain = som_set(sTrain,'time',datestr(now,0));
sMap.trainhist = sTrain;

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Function2- SOM_SEQTRAIN  Use sequential algorithm to train the Self-Organizing Map.
function [sMap, sTrain] = som_seqtrain(sMap, D, varargin)

%SOM_SEQTRAIN  Use sequential algorithm to train the Self-Organizing Map.

%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_seqtrain
%
% PURPOSE
%
% Trains a Self-Organizing Map using the sequential algorithm.
%
% SYNTAX
%
%  sM = som_seqtrain(sM,D);
%  sM = som_seqtrain(sM,sD);
%  sM = som_seqtrain(...,'argID',value,...);
%  sM = som_seqtrain(...,value,...);
%  [sM,sT] = som_seqtrain(M,D,...);
```

```
%
% DESCRIPTION
%
% Trains the given SOM (sM or M above) with the given training data
% (sD or D) using sequential SOM training algorithm. If no optional
% arguments (argID, value) are given, a default training is done, the
% parameters are obtained from SOM_TRAIN_STRUCT function. Using
% optional arguments the training parameters can be specified. Returns
% the trained and updated SOM and a train struct which contains
% information on the training.
%
% REFERENCES
%
% Kohonen, T., "Self-Organizing Map", 2nd ed., Springer-Verlag,
%    Berlin, 1995, pp. 78-82.
% Kohonen, T., "Clustering, Taxonomy, and Topological Maps of
%    Patterns", International Conference on Pattern Recognition
%    (ICPR), 1982, pp. 114-128.
% Kohonen, T., "Self-Organized formation of topologically correct
%    feature maps", Biological Cybernetics 43, 1982, pp. 59-69.
%
% REQUIRED INPUT ARGUMENTS
%
%  sM          The map to be trained.
%     (struct) map struct
%     (matrix) codebook matrix (field .data of map struct)
%              Size is either [munits dim], in which case the map grid
%              dimensions (msize) should be specified with optional arguments,
%              or [msize(1) ... msize(k) dim] in which case the map
%              grid dimensions are taken from the size of the matrix.
%              Lattice, by default, is 'rect' and shape 'sheet'.
%  D           Training data.
%     (struct) data struct
%     (matrix) data matrix, size [dlen dim]
%
% OPTIONAL INPUT ARGUMENTS
%
%  argID (string) Argument identifier string (see below).
%  value (varies) Value for the argument (see below).
%
%  The optional arguments can be given as 'argID',value -pairs. If an
%  argument is given value multiple times, the last one is
%  used. The valid IDs and corresponding values are listed below. The values
%  which are unambiguous (marked with '*') can be given without the
%  preceeding argID.
%
%   'mask'       (vector) BMU search mask, size dim x 1. Default is
%                         the one in sM (field '.mask') or a vector of
%                         ones if only a codebook matrix was given.
%   'msize'      (vector) map grid dimensions. Default is the one
%                         in sM (field sM.topol.msize) or
%                         'si = size(sM); msize = si(1:end-1);'
%                         if only a codebook matrix was given.
%   'radius'     (vector) neighborhood radius
%                         length = 1: radius_ini = radius
%                         length = 2: [radius_ini radius_fin] = radius
%                         length > 2: the vector given neighborhood
%                                     radius for each step separately
%                                     trainlen = length(radius)
%   'radius_ini' (scalar) initial training radius
%   'radius_fin' (scalar) final training radius
%   'alpha'      (vector) learning rate
%                         length = 1: alpha_ini = alpha
%                         length > 1: the vector gives learning rate
%                                     for each step separately
%                                     trainlen is set to length(alpha)
%                                     alpha_type is set to 'user defined'
%   'alpha_ini'  (scalar) initial learning rate
%   'tracking'   (scalar) tracking level: 0, 1 (default), 2 or 3
%                         0 - estimate time
%                         1 - track time and quantization error
%                         2 - plot quantization error
%                         3 - plot quantization error and two first
%                             components
%   'trainlen'   (scalar) training length (see also 'tlen_type')
%   'trainlen_type' *(string) is the trainlen argument given in 'epochs'
%                         or in 'samples'. Default is 'epochs'.
%   'sample_order'*(string) is the sample order 'random' (which is the
%                         the default) or 'ordered' in which case
%                         samples are taken in the order in which they
%                         appear in the data set
%   'train'     *(struct) train struct, parameters for training.
%                         Default parameters, unless specified,
%                         are acquired using SOM_TRAIN_STRUCT (this
%                         also applies for 'trainlen', 'alpha_type',
%                         'alpha_ini', 'radius_ini' and 'radius_fin').
%   'sTrain', 'som_train' (struct) = 'train'
%   'neigh'     *(string) The used neighborhood function. Default is
%                         the one in sM (field '.neigh') or 'gaussian'
%                         if only a codebook matrix was given. Other
%                         possible values is 'cutgauss', 'ep' and 'bubble'.
%   'topol'     *(struct) topology of the map. Default is the one
%                         in sM (field '.topol').
%   'sTopol', 'som_topol' (struct) = 'topol'
%   'alpha_type'*(string) learning rate function, 'inv', 'linear' or 'power'
%   'lattice'   *(string) map lattice. Default is the one in sM
%                         (field sM.topol.lattice) or 'rect'
%                         if only a codebook matrix was given.
%   'shape'     *(string) map shape. Default is the one in sM
%                         (field sM.topol.shape) or 'sheet'
%                         if only a codebook matrix was given.
%
% OUTPUT ARGUMENTS
%
%  sM          the trained map
%     (struct) if a map struct was given as input argument, a
%              map struct is also returned. The current training
%              is added to the training history (sM.trainhist).
%              The 'neigh' and 'mask' fields of the map struct
%              are updated to match those of the training.
%     (matrix) if a matrix was given as input argument, a matrix
%              is also returned with the same size as the input
%              argument.
%  sT (struct) train struct; information of the accomplished training
%
%% Check arguments

error(nargchk(2, Inf, nargin));  % check the number of input arguments

% map
struct_mode = isstruct(sMap);
if struct_mode,
  sTopol = sMap.topol;
else
  orig_size = size(sMap);
  if ndims(sMap) > 2,
    si = size(sMap); dim = si(end); msize = si(1:end-1);
    M = reshape(sMap,[prod(msize) dim]);
  else
    msize = [orig_size(1) 1];
    dim = orig_size(2);
  end
  sMap   = som_map_struct(dim,'msize',msize);
  sTopol = sMap.topol;
end
[munits dim] = size(sMap.codebook);

% data
if isstruct(D),
  data_name = D.name;
  D = D.data;
else
  data_name = inputname(2);
end
D = D(find(sum(isnan(D),2) < dim),:); % remove empty vectors from the data
[dlen ddim] = size(D);                % check input dimension
if dim ~= ddim, error('Map and data input space dimensions disagree.'); end

% varargin
sTrain = som_set('som_train','algorithm','seq','neigh', ...
         sMap.neigh,'mask',sMap.mask,'data_name',data_name);
radius     = [];
alpha      = [];
tracking   = 1;
sample_order_type = 'random';
tlen_type  = 'epochs';

i=1;
while i<=length(varargin),
  argok = 1;
  if ischar(varargin{i}),
    switch varargin{i},
     % argument IDs
     case 'msize', i=i+1; sTopol.msize = varargin{i};
     case 'lattice', i=i+1; sTopol.lattice = varargin{i};
     case 'shape', i=i+1; sTopol.shape = varargin{i};
     case 'mask', i=i+1; sTrain.mask = varargin{i};
     case 'neigh', i=i+1; sTrain.neigh = varargin{i};
     case 'trainlen', i=i+1; sTrain.trainlen = varargin{i};
     case 'trainlen_type', i=i+1; tlen_type = varargin{i};
     case 'tracking', i=i+1; tracking = varargin{i};
     case 'sample_order', i=i+1; sample_order_type = varargin{i};
     case 'radius_ini', i=i+1; sTrain.radius_ini = varargin{i};
     case 'radius_fin', i=i+1; sTrain.radius_fin = varargin{i};
```

```
      case 'radius',
       i=i+1;
       l = length(varargin{i});
       if l==1,
         sTrain.radius_ini = varargin{i};
       else
         sTrain.radius_ini = varargin{i}(1);
         sTrain.radius_fin = varargin{i}(end);
         if l>2, radius = varargin{i}; tlen_type = 'samples'; end
       end
      case 'alpha_type', i=i+1; sTrain.alpha_type = varargin{i};
      case 'alpha_ini', i=i+1; sTrain.alpha_ini = varargin{i};
      case 'alpha',
       i=i+1;
       sTrain.alpha_ini = varargin{i}(1);
       if length(varargin{i})>1,
         alpha = varargin{i}; tlen_type = 'samples';
         sTrain.alpha_type = 'user defined';
       end
      case {'sTrain','train','som_train'}, i=i+1; sTrain = varargin{i};
      case {'topol','sTopol','som_topol'},
       i=i+1;
       sTopol = varargin{i};
       if prod(sTopol.msize) ~= munits,
         error('Given map grid size does not match the codebook size.');
       end
       % unambiguous values
      case {'inv','linear','power'}, sTrain.alpha_type = varargin{i};
      case {'hexa','rect'}, sTopol.lattice = varargin{i};
      case {'sheet','cyl','toroid'}, sTopol.shape = varargin{i};
      case {'gaussian','cutgauss','ep','bubble'}, sTrain.neigh = varargin{i};
      case {'epochs','samples'}, tlen_type = varargin{i};
      case {'random', 'ordered'}, sample_order_type = varargin{i};
      otherwise argok=0;
     end
   elseif isstruct(varargin{i}) & isfield(varargin{i},'type'),
     switch varargin{i}(1).type,
      case 'som_topol',
       sTopol = varargin{i};
       if prod(sTopol.msize) ~= munits,
         error('Given map grid size does not match the codebook size.');
       end
      case 'som_train', sTrain = varargin{i};
      otherwise argok=0;
     end
   else
     argok = 0;
   end
   if ~argok,
     disp(['(som_seqtrain) Ignoring invalid argument #' num2str(i+2)]);
   end
   i = i+1;
 end

 % training length
 if ~isempty(radius) | ~isempty(alpha),
   lr = length(radius);
   la = length(alpha);
   if lr>2 | la>1,
     tlen_type = 'samples';
     if     lr> 2 & la<=1, sTrain.trainlen = lr;
     elseif lr<=2 & la> 1, sTrain.trainlen = la;
     elseif lr==la,        sTrain.trainlen = la;
     else
       error('Mismatch between radius and learning rate vector lengths.')
     end
   end
 end
 if strcmp(tlen_type,'samples'), sTrain.trainlen = sTrain.trainlen/dlen; end

 % check topology
 if struct_mode,
   if ~strcmp(sTopol.lattice,sMap.topol.lattice) | ...
     ~strcmp(sTopol.shape,sMap.topol.shape) | ...
     any(sTopol.msize ~= sMap.topol.msize),
     warning('Changing the original map topology.');
   end
 end
 sMap.topol = sTopol;

 % complement the training struct
 sTrain = som_train_struct(sTrain,sMap,'dlen',dlen);
 if isempty(sTrain.mask), sTrain.mask = ones(dim,1); end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %% initialize
```

```
M        = sMap.codebook;
mask     = sTrain.mask;
trainlen = sTrain.trainlen*dlen;

% neighborhood radius
if length(radius)>2,
  radius_type = 'user defined';
else
  radius = [sTrain.radius_ini sTrain.radius_fin];
  rini = radius(1);
  rstep = (radius(end)-radius(1))/(trainlen-1);
  radius_type = 'linear';
end

% learning rate
if length(alpha)>1,
  sTrain.alpha_type ='user defined';
  if length(alpha) ~= trainlen,
    error('Trainlen and length of neighborhood radius vector do not match.')
  end
  if any(isnan(alpha)),
    error('NaN is an illegal learning rate.')
  end
else
  if isempty(alpha), alpha = sTrain.alpha_ini; end
  if strcmp(sTrain.alpha_type,'inv'),
    % alpha(t) = a / (t+b), where a and b are chosen suitably
    % below, they are chosen so that alpha_fin = alpha_ini/100
    b = (trainlen - 1) / (100 - 1);
    a = b * alpha;
  end
end

% initialize random number generator
rand('state',sum(100*clock));

% distance between map units in the output space
%  Since in the case of gaussian and ep neighborhood functions, the
%  equations utilize squares of the unit distances and in bubble case
%  it doesn't matter which is used, the unitdistances and neighborhood
%  radiuses are squared.
Ud = som_unit_dists(sTopol).^2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Action

update_step = 100;
mu_x_1 = ones(munits,1);
samples = ones(update_step,1);
r = samples;
alfa = samples;

qe = 0;
start = clock;
if tracking >  0, % initialize tracking
  track_table = zeros(update_step,1);
  qe = zeros(floor(trainlen/update_step),1);
end

for t = 1:trainlen,

  % Every update_step, new values for sample indeces, neighborhood
  % radius and learning rate are calculated. This could be done
  % every step, but this way it is more efficient. Or this could
  % be done all at once outside the loop, but it would require much
  % more memory.
  ind = rem(t,update_step); if ind==0, ind = update_step; end
  if ind==1,
    steps = [t:min(trainlen,t+update_step-1)];
    % sample order
    switch sample_order_type,
     case 'ordered', samples = rem(steps,dlen)+1;
     case 'random',  samples = ceil(dlen*rand(update_step,1)+eps);
    end

    % neighborhood radius
    switch radius_type,
     case 'linear',        r = rini+(steps-1)*rstep;
     case 'user defined', r = radius(steps);
    end
    r=r.^2;        % squared radius (see notes about Ud above)
    r(r==0) = eps; % zero radius might cause div-by-zero error

    % learning rate
    switch sTrain.alpha_type,
     case 'linear',        alfa = (1-steps/trainlen)*alpha;
     case 'inv',           alfa = a ./ (b + steps-1);
     case 'power',        alfa = alpha * (0.005/alpha).^((steps-1)/trainlen);
     case 'user defined', alfa = alpha(steps);
    end
  end

  % find BMU
  x = D(samples(ind),:);                 % pick one sample vector
  known = ~isnan(x);                     % its known components
  Dx = M(:,known) - x(mu_x_1,known);     % each map unit minus the vector
  [qerr bmu] = min((Dx.^2)*mask(known)); % minimum distance(^2) and the BMU

  % tracking
  if tracking>0,
    track_table(ind) = sqrt(qerr);
    if ind==update_step,
      n = ceil(t/update_step);
      qe(n) = mean(track_table);
      trackplot(M,D,tracking,start,n,qe);
    end
  end

  % neighborhood & learning rate
  % notice that the elements Ud and radius have been squared!
  % (see notes about Ud above)
  switch sTrain.neigh,
  case 'bubble',    h = (Ud(:,bmu)<=r(ind));
  case 'gaussian',  h = exp(-Ud(:,bmu)/(2*r(ind)));
  case 'cutgauss',  h = exp(-Ud(:,bmu)/(2*r(ind))) .* (Ud(:,bmu)<=r(ind));
  case 'ep',        h = (1-Ud(:,bmu)/r(ind)) .* (Ud(:,bmu)<=r(ind));
  end
  h = h*alfa(ind);

  % update M
  M(:,known) = M(:,known) - h(:,ones(sum(known),1)).*Dx;

end; % for t = 1:trainlen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Build / clean up the return arguments

if tracking, fprintf(1,'\n'); end

% update structures
sTrain = som_set(sTrain,'time',datestr(now,0));
if struct_mode,
  sMap = som_set(sMap,'codebook',M,'mask',sTrain.mask,'neigh',sTrain.neigh);
  tl = length(sMap.trainhist);
  sMap.trainhist(tl+1) = sTrain;
else
  sMap = reshape(M,orig_size);
end

return;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% subfunctions

%%%%%%%%
function [] = trackplot(M,D,tracking,start,n,qe)

  l = length(qe);
  elap_t = etime(clock,start);
  tot_t = elap_t*l/n;
  fprintf(1,'\rTraining: %3.0f/ %3.0f s',elap_t,tot_t)
  switch tracking
   case 1,
   case 2,
    plot(1:n,qe(1:n),(n+1):l,qe((n+1):l))
    title('Quantization errors for latest samples')
    drawnow
   otherwise,
    subplot(2,1,1), plot(1:n,qe(1:n),(n+1):l,qe((n+1):l))
    title('Quantization error for latest samples');
    subplot(2,1,2), plot(M(:,1),M(:,2),'ro',D(:,1),D(:,2),'b.');
    title('First two components of map units (o) and data vectors (+)');
    drawnow
  end
  % end of trackplot



Function3- SOM_BMUS Find the best-matching units from the map for the given vectors.
function [Bmus,Qerrors] = som_bmus(sMap, sData, which_bmus, mask)

%SOM_BMUS Find the best-matching units from the map for the given vectors.
```

```
%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_bmus
%
% PURPOSE
%
% Finds Best-Matching Units (BMUs) for given data vector from a given map.
%
% SYNTAX
%
%  Bmus = som_bmus(sMap, sData)
%  Bmus = som_bmus(..., which)
%  Bmus = som_bmus(..., which, mask)
%  [Bmus, Qerrs] = som_bmus(...)
%
% DESCRIPTION
%
% Returns the indexes and corresponding quantization errors of the
% vectors in sMap that best matched the vectors in sData.
%
% By default only the index of the best matching unit (/vector) is
% returned, but the 'which' argument can be used to get others as
% well. For example it might be desirable to get also second- and
% third-best matching units as well (which = [1:3]).
%
% A mask can be used to weight the search process. The mask is used to
% weight the influence of components in the distance calculation, as
% follows:
%
%   distance(x,y) = (x-y)' diag(mask) (x-y)
%
% where x and y are two vectors, and diag(mask) is a diagonal matrix with
% the elements of mask vector on the diagonal.
%
% The vectors in the data set (sData) can contain unknown components
% (NaNs), but the map (sMap) cannot. If there are completely empty
% vectors (all NaNs), the returned BMUs and quantization errors for those
% vectors are NaNs.
%
% REQUIRED INPUT ARGUMENTS
%
%   sMap              The vectors from among which the BMUs are searched
%                     for. These must not have any unknown components (NaNs).
%            (struct) map struct
%            (matrix) codebook matrix, size munits x dim
%
%   sData             The data vector(s) for which the BMUs are searched.
%            (struct) data struct
%            (matrix) data matrix, size dlen x dim
%
% OPTIONAL INPUT ARGUMENTS
%
%   which    (vector) which BMUs are returned,
%                     by default only the best (ie. which = [1])
%            (string) 'all', 'best' or 'worst' meaning [1:munits],
%                     [1] and [munits] respectively
%   mask     (vector) mask vector to be used in BMU search,
%                     by default sMap.mask, or ones(dim,1) in case
%                     a matrix was given
%
% OUTPUT ARGUMENTS
%
%   Bmus     (matrix) the requested BMUs for each data vector,
%                     size dlen x length(which)
%   Qerrors  (matrix) the corresponding quantization errors,
%                     size equal to that of Bmus
%

error(nargchk(1, 4, nargin));  % check no. of input args is correct

% sMap
if isstruct(sMap),
  switch sMap.type,
   case 'som_map', M = sMap.codebook;
   case 'som_data', M = sMap.data;
   otherwise, error('Invalid 1st argument.');
  end
else
  M = sMap;
end
[munits dim] = size(M);
if any(any(isnan(M))),
  error ('Map codebook must not have missing components.');
end

% data
if isstruct(sData),
  switch sData.type,
   case 'som_map', D = sData.codebook;
   case 'som_data', D = sData.data;
   otherwise, error('Invalid 2nd argument.');
  end
else
  D = sData;
end
[dlen ddim] = size(D);
if dim ~= ddim,
  error('Data and map dimensions do not match.')
end

% which_bmus
if nargin < 3 | isempty(which_bmus) | any(isnan(which_bmus)),
  which_bmus = 1;
else
  if ischar(which_bmus),
    switch which_bmus,
     case 'best', which_bmus = 1;
     case 'worst', which_bmus = munits;
     case 'all', which_bmus = [1:munits];
    end
  end
end

% mask
if nargin < 4 | isempty(mask) | any(isnan(mask)),
  if isstruct(sMap) & strcmp(sMap.type,'som_map'),
    mask = sMap.mask;
  elseif isstruct(sData) & strcmp(sData.type,'som_map'),
    mask = sData.mask;
  else
    mask = ones(dim,1);
  end
end
if size(mask,1)==1, mask = mask'; end
if all(mask == 0),
  error('All components masked off. BMU search cannot be done.');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% action

Bmus = zeros(dlen,length(which_bmus));
Qerrors = Bmus;

% The BMU search involves calculating weighted Euclidian distances
% to all map units for each data vector. Basically this is done as
%   for i=1:dlen,
%     for j=1:munits,
%       for k=1:dim,
%         Dist(j,i) = Dist(j,i) + mask(k) * (D(i,k) - M(j,k))^2;
%       end
%     end
%   end
% where mask is the weighting vector for distance calculation. However, taking
% into account that distance between vectors m and v can be expressed as
%   |m - v|^2 = sum_i ((m_i - v_i)^2) = sum_i (m_i^2 + v_i^2 - 2*m_i*v_i)
% this can be made much faster by transforming it to a matrix operation:
%   Dist = (M.^2)*mask*ones(1,d) + ones(m,1)*mask'*(D'.^2) - 2*M*diag(mask)*D'
%
% In the case where there are unknown components in the data, each data
% vector will have an individual mask vector so that for that unit, the
% unknown components are not taken into account in distance calculation.
% In addition all NaN's are changed to zeros so that they don't screw up
% the matrix multiplications.

% calculate distances & bmus

% This is done a block of data at a time rather than in a
% single sweep to save memory consumption. The 'Dist' matrix has
% size munits*blen which would be HUGE if you did it in a single-sweep
% operation. If you _want_ to use the single-sweep version, just
% set blen = dlen. If you're having problems with memory, try to
% set the value of blen lower.
blen = min(munits,dlen);

% handle unknown components
Known = ~isnan(D);
W1 = (mask*ones(1,dlen)) .* Known';
D(find(~Known)) = 0;
unknown = find(sum(Known')==0); % completely unknown vectors

% constant matrices
WD = 2*diag(mask)*D';   % constant matrix
dconst = ((D.^2)*mask); % constant term in the distances
```

```
i0 = 0;
while i0+1<=dlen,
  % calculate distances
  inds = [(i0+1):min(dlen,i0+blen)]; i0 = i0+blen;
  Dist = (M.^2)*W1(:,inds) - M*WD(:,inds); % plus dconst for each sample

  % find the bmus and the corresponding quantization errors
  if all(which_bmus==1), [Q B] = min(Dist); else [Q B] = sort(Dist); end
  if munits==1, Bmus(inds,:) = 1; else Bmus(inds,:) = B(which_bmus,:)'; end
  Qerrors(inds,:) = Q(which_bmus,:)' + dconst(inds,ones(length(which_bmus),1));
end

% completely unknown vectors
if ~isempty(unknown),
  Bmus(unknown,:) = NaN;
  Qerrors(unknown,:) = NaN;
end

Qerrors = sqrt(Qerrors);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Function4- SOM_DIVIDE Divides a dataset according to a given map.

```
function [V,I]=som_divide(sMap, D, inds, mode)

%SOM_DIVIDE Divides a dataset according to a given map.
%
% [V,I]=som_divide(sMap, sData, [inds], [mode])
%
% ARGUMENTS ([]'s are optional)
%
% sMap (struct or matrix) map struct or codebook (size munits x dim)
% sData (struct or matrix) data struct or matrix (size N x dim )
% [inds] From which map units should the local data sets
% be constructed. Interpretation depends on mode
% argument. By default [1:munits].
% 'class': (vector) munits x 1 matrix of class numbers
% 'index': (vector) K x 1 vector of map node indexes
% 'index': (matrix) K x k matrix of map node subscripts
% [mode] (string) 'index' or 'class', if inds is a vector of length
% munits, default is 'class', otherwise 'index'.
% RETURNS
%
% If mode == 'index'
% V (matrix) data vectors hitting the specified nodes (size K x dim)
% I (vector) corresponding data row indexes (size K x 1)
%
% If mode == 'class' (this can be used after using som_select)
% V (cell array) V{K} includes vectors whose BMU has class number
% K in the input matrix 'coord'. Note that
% values of K below 1 are ignored.
% I (cell array) corresponding data indexes in the cell array
%
% NOTE: if the same node is specified multiple times, only one
% set of hits is returned.
%
%%%% Init & Check %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

error(nargchk(0, 4, nargin)) % check if no. of input args is correct

% map
if isstruct(sMap),
  msize = sMap.topol.msize;
  dim = size(sMap.codebook,2);
else
  msize = [size(sMap,1) 1];
  dim = size(sMap,2);
end
munits = prod(msize);

% data
if isstruct(D), D=D.data; end

% inds
if nargin<3, inds = 1:munits; end
isvec = prod(size(inds))==length(inds);

% mode
if nargin<4,
  if isvec & length(inds)==munits,
    mode = 'class';
  else
    mode = 'index';
  end
end
```

```
%%% Action & Build output according to the mode string output

if ~isvec, inds = som_sub2ind(msize,inds); end

bmus=som_bmus(sMap,D);

switch mode
 case 'index'
  I=find(ismember(bmus,inds));
  V=D(I,:);
 case 'class'
  K=max(inds); % classes
  V = cell(K,1);
  I = cell(K,1);
  for i=1:K,
    N_ind=find(inds == i); % indexes of the units of class i
    I{i}=find(ismember(bmus,N_ind)); % data indexes
    V{i}=D(I{i},:);
  end
end
```

Function5- SOM_EUCDIST2 Calculates matrix of squared euclidean distances between set of vectors or map, data struct

```
function d=som_eucdist2(Data, Proto)

%SOM_EUCDIST2 Calculates matrix of squared euclidean distances between set of vectors
or map, data struct
%
% d=som_eucdist2(D, P)
%
% d=som_eucdist(sMap, sData);
% d=som_eucdist(sData, sMap);
% d=som_eucdist(sMap1, sMap2);
% d=som_eucdist(datamatrix1, datamatrix2);
%
% Input and output arguments ([]'s are optional):
% D (matrix) size Nxd
% (struct) map or data struct
% P (matrix) size Pxd
% (struct) map or data struct
% d (matrix) distance matrix of size NxP
%
% IMPORTANT
%
% * Calculates _squared_ euclidean distances
% * Observe that the mask in the map struct is not taken into account while
% calculating the euclidean distance
%
% See also KNN, PDIST.
% Version 2.0beta Johan 291000

%% Init %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if isstruct(Data);
  if isfield(Data,'type') & ischar(Data.type),
    ;
  else
    error('Invalid map/data struct?');
  end
  switch Data.type
   case 'som_map'
    data=Data.codebook;
   case 'som_data'
    data=Data.data;
  end
else
  % is already a matrix
  data=Data;
end

% Take prototype vectors from prototype struct

if isstruct(Proto),

  if isfield(Proto,'type') & ischar(Proto.type),
    ;
  else
    error('Invalid map/data struct?');
  end
  switch Proto.type
   case 'som_map'
    proto=Proto.codebook;
   case 'som_data'
    proto=Proto.data;
  end
else
```

```
  % is already a matrix
  proto=Proto;
end

% Check that inputs are matrices
if ~vis_valuetype(proto,{'nxm'}) | ~vis_valuetype(data,{'nxm'}),
  error('Prototype or data input not valid.')
end

% Record data&proto sizes and check their dims
[N_data dim_data]=size(data);
[N_proto dim_proto]=size(proto);
if dim_proto ~= dim_data,
  error('Data and prototype vector dimension does not match.');
end

% Calculate euclidean distances between classifiees and prototypes
d=distance(data,proto);

%%%% Classification %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function d=distance(X,Y);

% Euclidean distance matrix between row vectors in X and Y

U=~isnan(Y); Y(~U)=0;
V=~isnan(X); X(~V)=0;
d=abs(X.^2*U'+V*Y'.^2-2*X*Y');

Function6- SOM_MAP_STRUCT Create map struct.
function sMap = som_map_struct(dim, varargin)

%SOM_MAP_STRUCT Create map struct.
%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_map_struct
%
% PURPOSE
%
% Creates a self-organizing map structure.
%
% SYNTAX
%
%  sM = som_map_struct(dim)
%  sM = som_map_struct(...,'argID',value,...);
%  sM = som_map_struct(...,value,...);
%
% DESCRIPTION
%
% Creates a self-organizing map struct. The struct contains the map
% codebook, labels, topology, information on normalization and training,
% as well as component names and a name for the map. The obligatory
% parameter is the map dimension. Most of the other fields can be
% given values using optional arguments. If they are left unspecified,
% default values are used.
%
%  Field         Type         Size / default value (munits = prod(msize))
%  ------------------------------------------------------------------------
%   .type        (string)     'som_map'
%   .name        (string)     'SOM date'
%   .codebook    (matrix)     rand(munits, dim)
%   .topol       (struct)     topology struct, with the following fields
%     .type         (string)  'som_topol'
%     .msize        (vector)  size k x 1, [0]
%     .lattice      (string)  'hexa'
%     .shape        (string)  'sheet'
%   .labels      (cellstr)    size munits x m, {''; ''; ... ''}
%   .neigh       (string)     'gaussian'
%   .mask        (vector)     size dim x 1, [1; 1; ...; 1]
%   .trainhist   (cell array) size tl x 1, []
%   .comp_names  (cellstr)    size dim x 1, {'Variable1', 'Variable2', ...}
%   .comp_norm   (cell array) size dim x 1, {[], [], ... []}
%
% '.type' field is the struct identifier. Do not change it.
% '.name' field is the identifier for the whole map struct
% '.codebook' field is the codebook matrix, each row corresponds to one unit
% '.topol' field is the topology of the map. This struct has three fields:
%   '.msize' field is the dimensions of the map grid. Note that the
%         matrix notation of indeces is used.
%   '.lattice' field is the map grid lattice
%   '.shape' field is the map grid shape
% '.labels' field contains the labels for each of the vectors. The ith row
%         of '.labels' contains the labels for ith map unit. Note that
%         if some vectors have more labels than others, the others are
%         are given empty labels ('') to pad the '.labels' array up.
% '.neigh' field is the neighborhood function.
% '.mask' field is the BMU search mask.
% '.trainhist' field contains information on the training. It is a cell
%         array of training structs. The first training struct contains
%         information on initialization, the others on actual trainings.
%         If the map has not been initialized, '.trainhist' is empty ([]).
% '.comp_names' field contains the names of the vector components
% '.comp_norm' field contains normalization information for each
%         component. Each cell of '.comp_norm' is itself a cell array of
%         normalization structs. If no normalizations are performed for
%         the particular component, the cell is empty ([]).
%
% REQUIRED INPUT ARGUMENTS
%
%  dim    (scalar) Input space dimension.
%
% OPTIONAL INPUT ARGUMENTS
%
%  argID (string) Argument identifier string (see below).
%  value (varies) Value for the argument (see below).
%
%  The optional arguments are given as 'argID',value -pairs. If the
%  value is unambiguous (marked below with '*'), it can be given
%  without the preceeding argID. If an argument is given value
%  multiple times, the last one is used.
%
%   'mask'       (vector) BMU search mask, size dim x 1
%   'msize'      (vector) map grid size, default is [0]
%   'labels'     (string array / cellstr) labels for each map unit,
%                 length=prod(msize)
%   'name'       (string) map name
%   'comp_names' (string array / cellstr) component names, size dim x 1
%   'comp_norm'  (cell array) normalization operations for each
%                 component, size dim x 1. Each cell is either empty,
%                 or a cell array of normalization structs.
%   'lattice'   *(string) map lattice, 'hexa' or 'rect'
%   'shape'     *(string) map shape, 'sheet', 'cyl' or 'toroid'
%   'topol'     *(struct) topology struct, sets msize, lattice and shape
%   'som_topol','sTopol' = 'topol'
%   'neigh'     *(string) neighborhood function, 'gaussian', 'cutgauss',
%                 'ep' or 'bubble'
%
% OUTPUT ARGUMENTS
%
%  sMap (struct) the map struct

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% default values
sTopol     = som_set('som_topol','lattice','hexa','shape','sheet');
neigh      = 'gaussian';
mask       = ones(dim,1);
name       = sprintf('SOM %s', datestr(now, 1));
labels     = cell(prod(sTopol.msize),1);
for i=1:length(labels), labels{i} = ''; end
comp_names = cell(dim,1);
for i = 1:dim, comp_names{i} = sprintf('Variable%d', i); end
comp_norm  = cell(dim,1);

% varargin
i=1;
while i<=length(varargin),
  argok = 1;
  if ischar(varargin{i}),
    switch varargin{i},
      % argument IDs
     case 'mask',       i=i+1; mask = varargin{i};
     case 'msize',      i=i+1; sTopol.msize = varargin{i};
     case 'labels',     i=i+1; labels = varargin{i};
     case 'name',       i=i+1; name = varargin{i};
     case 'comp_names', i=i+1; comp_names = varargin{i};
     case 'comp_norm',  i=i+1; comp_norm = varargin{i};
     case 'lattice',    i=i+1; sTopol.lattice = varargin{i};
     case 'shape',      i=i+1; sTopol.shape = varargin{i};
     case {'topol','som_topol','sTopol'}, i=i+1; sTopol = varargin{i};
     case 'neigh',      i=i+1; neigh = varargin{i};
      % unambiguous values
     case {'hexa','rect'}, sTopol.lattice = varargin{i};
     case {'sheet','cyl','toroid'}, sTopol.shape = varargin{i};
     case {'gaussian','cutgauss','ep','bubble'}, neigh = varargin{i};
     otherwise argok=0;
    end
  elseif isstruct(varargin{i}) & isfield(varargin{i},'type'),
    switch varargin{i}(1).type,
     case 'som_topol', sTopol = varargin{i};
     otherwise argok=0;
    end
  else
    argok = 0;
  end
  if ~argok,
    disp(['(som_map_struct) Ignoring invalid argument #' num2str(i+1)]);
  end
  i = i+1;
end

% create the SOM
codebook = rand(prod(sTopol.msize),dim);
sTrain = som_set('som_train','time',datestr(now,0),'mask',mask);
sMap = som_set('som_map','codebook',codebook,'topol',sTopol,...
                         'neigh',neigh,'labels',labels,'mask',mask,...
                         'comp_names',comp_names,'name',name,...
                         'comp_norm',comp_norm,'trainhist',sTrain);
```

Function7- SOM_SET Create and check SOM Toolbox structs, give values to their fields.

```
function [sS, ok, msgs] = som_set(sS, varargin)

%SOM_SET Create and check SOM Toolbox structs, give values to their fields.
%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_set
%
% PURPOSE
%
% Create and set values for fields of SOM Toolbox structs (except
% visualization struct). Can also be used to check the validity of structs.
%
% SYNTAX
%
%  sMap   = som_set('som_map');
%  sData  = som_set(sData);
%  sNorm  = som_set(...,'field',contents,...);
%  [sTopol,ok]       = som_set(sTopol,...);
%  [sTrain,ok,msgs] = som_set('som_train',...);
%
% DESCRIPTION
%
% The function is used to create and set values for fields of SOM
% Toolbox structs, except visualization structs. The given values are
% first checked for validity, and if they are not valid, an error
% message is returned. The function can also be used to check the
% validity of all the fields of the struct by supplying a struct as
% the first and only argument.
%
% NOTE: Using SOM_SET to create structures does _not_ guarantee that the
% structs are valid (try e.g. sM = som_set('som_map'); som_set(sM)). The
% initial values that the function gives to the fields of the structs are
% typically invalid. It is recommended that when creating map or data
% structs, the corresponding functions SOM_MAP_STRUCT and SOM_DATA_STRUCT
% are used instead of SOM_SET. However, when giving values for the fields,
% SOM_SET tries to guarantee that the values are valid.
%
% If a string is given as the first argument, the corresponding
% structure is first created and the field-content pairs are then
% applied to it.
%
% There can be arbitrarily many field-contents pairs. The pairs
% are processed sequentially one pair at a time. For each pair,
% the validity of the contents is checked and the corresponding
% items in the returned 'ok'-vector and 'msgs'-cellstring are set.
% - if the contents is ok, the status is set to 1 and message to ''
% - if the contents is suspicious, status is set to 1, but a
%   message is produced
% - if the contents is invalid, status is set to 0 and an error
%   message is produced. The contents are _not_ given to the field.
% If there is only one output argument, the status and messages
% for each pair are printed to standard output.
%
% The different field-contents pairs have no effect on each other.
% If a field is given a value multiple times, the last valid one
% stays in effect.
%
% In some cases, the order of the given fields is significant.
% For example in the case of 'som_map', the validity of some fields,
% like '.comp_names', depends on the input space dimension, which is
% checked from the '.data' field (dim = size(sD.data,2) to be specific).
% Therefore, the '.data' field (or '.codebook' field in case of map
% struct) should always be given a value first. Below is a list of
% this kind of dependancies:
%
% som_map:    'comp_names', 'comp_norm', 'msize', 'topol.msize',
%             'labels' and 'mask' depend on 'codebook'
%             new value for 'codebook' should have equal size to the old
%             one (unless the old one was empty)
% som_data:   'comp_names' and 'comp_norm' depend on 'data'
%             new value for 'data' should have equal dimension (size(data,2))
%            as the old one (unless the old one was empty)
%
% KNOWN BUGS
%
% Checking the values given to som_grid struct has not been
% implemented. Use SOM_GRID function to give the values.
%
% REQUIRED INPUT ARGUMENTS
%
%  sS          The struct.
%     (struct) A SOM Toolbox struct.
%     (string) Identifier of a SOM Toolbox struct: 'som_map',
%              'som_data', 'som_topol', 'som_norm' or 'som_train'
%
% OPTIONAL INPUT ARGUMENTS
%
%  field     (string) Field identifier string (see below).
%  contents  (varies) Value for the field (see below).
%
%  Below is the list of valid field identifiers for the different
%  SOM Toolbox structs.
%
%  'som_map' (map struct)
%    'codebook'    : matrix, size [munits, dim]
%    'labels'      : cell array of strings,
%                    size [munits, maximum_number_of_labels]
%    'topol'       : topology struct (prod(topol.msize)=munits)
%    'mask'        : vector, size [dim, 1]
%    'neigh'       : string ('gaussian' or 'cutgauss' or 'bubble' or 'ep')
%    'trainhist'   : struct array of train structs
%    'name'        : string
%    'comp_names'  : cellstr, size [dim, 1], e.g. {'c1','c2','c3'}
%    'comp_norm'   : cell array, size [dim, 1], of cell arrays
%                    of normalization structs
%    Also the following can be used (although they are fields
%    of the topology struct)
%    'msize'       : vector (prod(msize)=munits)
%    'lattice'     : string ('rect' or 'hexa')
%    'shape'       : string ('sheet' or 'cyl' or 'toroid')
%
%  'som_data' (data struct)
%    'data'        : matrix, size [dlen, dim]
%    'name'        : string
%    'labels'      : cell array of strings,
%                    size [dlen, m]
%    'comp_names'  : cellstr, size [dim, 1], e.g. {'c1','c2','c3'}
%    'comp_norm'   : cell array, size [dim, 1], of cell arrays
%                    of normalization structs
%    'label_names' : cellstr, size [m, 1]
%
% 'som_topol' (topology struct)
%    'msize'       : vector
%    'lattice'     : string ('rect' or 'hexa')
%    'shape'       : string ('sheet' or 'cyl' or 'toroid')
%
% 'som_norm' (normalization struct)
%    'method'      : string
%    'params'      : varies
%    'status'      : string ('done' or 'undone' or 'uninit')
%
% 'som_train' (train struct)
%    'algorithm'   : string ('seq' or 'batch' or 'lininit' or 'randinit')
%    'data_name'   : string
%    'mask'        : vector, size [dim, 1]
%    'neigh'       : string ('gaussian' or 'cutgauss' or 'bubble' or 'ep')
%    'radius_ini'  : scalar
%    'radius_fin'  : scalar
%    'alpha_ini'   : scalar
%    'alpha_type'  : string ('linear' or 'inv' or 'power')
%    'trainlen'    : scalar
%    'time'        : string
%
% 'som_grid' (grid struct) : checking the values has not been implemented yet!
%    'lattice'     : string ('rect' or 'hexa') or
%                    (sparce) matrix, size munits x munits
%    'shape'       : string ('sheet' or 'cyl' or 'toroid')
%    'msize'       : vector, size 1x2
%    'coord'       : matrix, size munits x 2 or munits x 3
%    'line'        : string (linespec, e.g. '-', or 'none')
%    'linecolor'   : RGB triple or string (colorspec, e.g. 'k') or
%                    munits x munits x 3 (sparce) matrix or cell
%                    array of RGB triples
%    'linewidth'   : scalar or munits x munits (sparce) matrix
%    'marker'      : string (markerspec, e.g. 'o', or 'none') or
%                    munits x 1 cell or char array of these
%    'markersize'  : scalar or munits x 1 vector
%    'markercolor' : RGB triple or string (colorspec, e.g. 'k')
%    'surf'        : [], munits x 1 or munits x 3 matrix of RGB triples
%    'label'       : [] or munits x 1 char array or
%                    munits x l cell array of strings
%    'labelcolor'  : RGB triple or string (colorspec, e.g. 'g' or 'none')
%    'labelsize'   : scalar
%
% OUTPUT ARGUMENTS
%
%  sS    (struct)  the created / updated struct
%  ok    (vector)  length = number of field-contents pairs, gives
%                  validity status for each pair (0=invalid, 1 otherwise)
%  msgs  (cellstr) length = number of field-contents pairs, gives
%                  error/warning message for each pair ('' if ok)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% create struct if necessary

if ischar(sS),
  switch sS
   case 'som_map',
    sS=struct('type', 'som_map', ...
              'codebook', [], ...
              'topol', som_set('som_topol'), ...
              'labels', cell(1), ...
              'neigh', 'gaussian', ...
              'mask', [], ...
              'trainhist', cell(1), ...
              'name', '',...
              'comp_names', {''}, ...
              'comp_norm', cell(1));
   case 'som_data',
    sS=struct('type', 'som_data', ...
              'data', [], ...
              'labels', cell(1), ...
              'name', '', ...
              'comp_names', {''}, ...
              'comp_norm', cell(1), ...
              'label_names', []);
   case 'som_topol',
    sS=struct('type', 'som_topol', ...
              'msize', 0, ...
              'lattice', 'hexa', ...
              'shape', 'sheet');
   case 'som_train',
    sS=struct('type', 'som_train', ...
              'algorithm', '', ...
              'data_name', '', ...
              'neigh', 'gaussian', ...
              'mask', [], ...
              'radius_ini', NaN, ...
              'radius_fin', NaN, ...
              'alpha_ini', NaN, ...
              'alpha_type', 'inv', ...
              'trainlen', NaN, ...
              'time', '');
   case 'som_norm',
    sS=struct('type', 'som_norm', ...
              'method', 'var', ...
              'params', [], ...
              'status', 'uninit');
   case 'som_grid',
    sS=struct('type','som_grid',...
          'lattice','hexa',...
          'shape','sheet',...
          'msize',[1 1],...
          'coord',[],...
          'line','-',...
          'linecolor',[.9 .9 .9],...
          'linewidth',0.5,...
          'marker','o',...
          'markersize',6,...
          'markercolor','k',...
          'surf',[],...
          'label',[],...
          'labelcolor','g',...
          'labelsize',12);
   otherwise
    ok=0; msgs = {['Unrecognized struct type: ' sS]}; sS = [];
    return;
  end

elseif isstruct(sS) & length(varargin)==0,

  % check all fields
  fields = fieldnames(sS);
  if ~any(strcmp('type',fields)),
    error('The struct has no ''type'' field.');
  end
  k = 0;
  for i=1:length(fields),
    contents = getfield(sS,fields{i});
    if ~strcmp(fields{i},'type'),
      varargin{k+1} = fields{i};
      varargin{k+2} = contents;
      k = k + 2;
    else
      if ~any(strcmp(contents, ...
        {'som_map','som_data','som_topol','som_train','som_norm'})),
    error(['Unknown struct type: ' contents]);
      end
    end
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% set field values

p = ceil(length(varargin)/2);
ok = ones(p,1);
msgs = cell(p,1);

for i=1:p,
  field = varargin{2*i-1};
  content = varargin{2*i};
  msg = '';
  isok = 0;

  si = size(content);
  isscalar = (prod(si)==1);
  isvector = (sum(si>1)==1);
  isrowvector = (isvector & si(1)==1);
  if isnumeric(content),
    iscomplete = all(~isnan(content(:)));
    ispositive = all(content(:)>0);
    isinteger  = all(content(:)==ceil(content(:)));
    isrgb = all(content(:)>=0 & content(:)<=1) & size(content,2)==3;
  end

  switch sS.type,
   case 'som_map',
    [munits dim] = size(sS.codebook);
    switch field,
     case 'codebook',
      if ~isnumeric(content),
    msg = '''codebook'' should be a numeric matrix';
      elseif size(content) ~= size(sS.codebook) & ~isempty(sS.codebook),
    msg = 'New ''codebook'' must be equal in size to the old one.';
      elseif ~iscomplete,
    msg = 'Map codebook must not contain NaN''s.';
      else
    sS.codebook = content; isok=1;
      end
     case 'labels',
      if isempty(content),
    sS.labels = cell(munits,1); isok = 1;
      elseif size(content,1) ~= munits,
    msg = 'Length of labels array must be equal to the number of map units.';
      elseif ~iscell(content) & ~ischar(content),
    msg = '''labels'' must be a string array or a cell array/matrix.';
      else
    isok = 1;
    if ischar(content), content = cellstr(content);
    elseif ~iscellstr(content),
      l = prod(size(content));
      for j=1:l,
        if ischar(content{j}),
          if ~isempty(content{j}),
        msg = 'Invalid ''labels'' array.';
        isok = 0;
        break;
          else
        content{j} = '';
          end
        end
      end
    end
    if isok, sS.labels = content; end
      end
     case 'topol',
      if ~isstruct(content),
    msg = '''topol'' should be a topology struct.';
      elseif ~isfield(content,'msize') | ...
        ~isfield(content,'lattice') | ...
        ~isfield(content,'shape'),
    msg = '''topol'' is not a valid topology struct.';
      elseif prod(content.msize) ~= munits,
    msg = '''topol''.msize does not match the number of map units.';
      else
    sS.topol = content; isok = 1;
      end
     case 'msize',
      if ~isnumeric(content) | ~isvector | ~ispositive | ~isinteger,
    msg = '''msize'' should be a vector with positive integer elements.';
      elseif prod(content) ~= munits,
    msg = '''msize'' does not match the map size.';
      else
    sS.topol.msize = content; isok = 1;
      end
     case 'lattice',
      if ~ischar(content),
    msg = '''lattice'' should be a string';
      elseif ~strcmp(content,'rect') & ~strcmp(content,'hexa'),
    msg = ['Unknown lattice type: ' content];
    sS.topol.lattice = content; isok = 1;
      else
    sS.topol.lattice = content; isok = 1;
      end
     case 'shape',
      if ~ischar(content),
    msg = '''shape'' should be a string';
      elseif ~strcmp(content,'sheet') & ~strcmp(content,'cyl') & ...
        ~strcmp(content,'toroid'),
    msg = ['Unknown shape type:' content];
    sS.topol.shape = content; isok = 1;
      else
    sS.topol.shape = content; isok = 1;
      end
     case 'neigh',
      if ~ischar(content),
    msg = '''neigh'' should be a string';
      elseif ~strcmp(content,'gaussian') & ~strcmp(content,'ep') & ...
        ~strcmp(content,'cutgauss') & ~strcmp(content,'bubble'),
    msg = ['Unknown neighborhood function: ' content];
    sS.neigh = content; isok = 1;
      else
    sS.neigh = content; isok = 1;
      end
     case 'mask',
      if size(content,1) == 1, content = content'; end
      if ~isnumeric(content) | size(content) ~= [dim 1],
    msg = '''mask'' should be a column vector (size dim x 1).';
      else
    sS.mask = content; isok = 1;
      end
     case 'name',
      if ~ischar(content),
    msg = '''name'' should be a string.';
      else
    sS.name = content; isok = 1;
      end
     case 'comp_names',
      if ~iscell(content) & ~ischar(content),
    msg = '''comp_names'' should be a cell string or a string array.';
      elseif length(content) ~= dim,
    msg = 'Length of ''comp_names'' should be equal to dim.';
      else
    if ischar(content), content = cellstr(content); end
    if size(content,1)==1, content = content'; end
    sS.comp_names = content;
    isok = 1;
      end
     case 'comp_norm',
      if ~iscell(content) & length(content)>0,
    msg = '''comp_norm'' should be a cell array.';
      elseif length(content) ~= dim,
    msg = 'Length of ''comp_norm'' should be equal to dim.';
      else
    isok = 1;
    for j=1:length(content),
      if ~isempty(content{j}) & (~isfield(content{j}(1),'type') | ...
                     ~strcmp(content{j}(1).type,'som_norm')),
        msg = 'Each cell in ''comp_norm'' should be either empty or type
''som_norm''.';
        isok = 0;
        break;
      end
    end
    if isok, sS.comp_norm = content; end
      end
     case 'trainhist',
      if ~isstruct(content) & ~isempty(content),
    msg = '''trainhist'' should be a struct array or empty.';
      else
    isok = 1;
    for j=1:length(content),
      if ~isfield(content(j),'type') | ~strcmp(content(j).type,'som_train'),
        msg = 'Each cell in ''trainhist'' should be of type ''som_train''.';
        isok = 0;
        break;
      end
    end
    if isok, sS.trainhist = content; end
      end
     otherwise,
      msg = ['Invalid field for map struct: ' field];
    end

   case 'som_data',
    [dlen dim] = size(sS.data);
    switch field,
     case 'data',
      [dummy dim2] = size(content);
      if prod(si)==0,
    msg = '''data'' is empty';
      elseif ~isnumeric(content),
    msg = '''data'' should be numeric matrix.';
      elseif dim ~= dim2 & ~isempty(sS.data),
    msg = 'New ''data'' must have the same dimension as old one.';
      else
    sS.data = content; isok = 1;
      end
     case 'labels',
      if isempty(content),
    sS.labels = cell(dlen,1); isok = 1;
      elseif size(content,1) ~= dlen,
    msg = 'Length of ''labels'' must be equal to the number of data vectors.';
      elseif ~iscell(content) & ~ischar(content),
    msg = '''labels'' must be a string array or a cell array/matrix.';
      else
    isok = 1;
    if ischar(content), content = cellstr(content);
    elseif ~iscellstr(content),
      l = prod(size(content));
      for j=1:l,
        if ~ischar(content{j}),
          if ~isempty(content{j}),
        msg = 'Invalid ''labels'' array.';
        isok = 0; j
        break;
          else
        content{j} = '';
          end
        end
      end
    end
    if isok, sS.labels = content; end
      end
     case 'name',
      if ~ischar(content),
    msg = '''name'' should be a string.';
      else
    sS.name = content; isok = 1;
      end
     case 'comp_names',
      if ~iscell(content) & ~ischar(content),
    msg = '''comp_names'' should be a cell string or a string array.';
      elseif length(content) ~= dim,
    msg = 'Length of ''comp_names'' should be equal to dim.';
      else
    if ischar(content), content = cellstr(content); end
    if size(content,1)==1, content = content'; end
    sS.comp_names = content;
    isok = 1;
      end
     case 'comp_norm',
      if ~iscell(content) & length(content)>0,
    msg = '''comp_norm'' should be a cell array.';
      elseif length(content) ~= dim,
    msg = 'Length of ''comp_norm'' should be equal to dim.';
      else
    isok = 1;
    for j=1:length(content),
      if ~isempty(content{j}) & (~isfield(content{j}(1),'type') | ...
                     ~strcmp(content{j}(1).type,'som_norm')),
        msg = 'Each cell in ''comp_norm'' should be either empty or type
''som_norm''.';
        isok = 0;
```

```
        break;
      end
    end
    if isok, sS.comp_norm = content; end
      end
     case 'label_names',
      if ~iscell(content) & ~ischar(content) & ~isempty(content),
    msg = ['''label_names'' should be a cell string, a string array or' ...
           ' empty.'];
      else
    if ~isempty(content),
      if ischar(content), content = cellstr(content); end
      if size(content,1)==1, content = content'; end
    end
    sS.label_names = content;
    isok = 1;
      end
     otherwise,
      msg = ['Invalid field for data struct: ' field];
    end

   case 'som_topol',
    switch field,
     case 'msize',
      if ~isnumeric(content) | ~isvector | ~ispositive | ~isinteger,
    msg = '''msize'' should be a vector with positive integer elements.';
      else
    sS.msize = content; isok=1;
      end
     case 'lattice',
      if ~ischar(content),
    msg = '''lattice'' should be a string';
      elseif ~strcmp(content,'rect') & ~strcmp(content,'hexa'),
    msg = ['Unknown lattice type: ' content];
    sS.lattice = content; isok = 1;
      else
    sS.lattice = content; isok = 1;
      end
     case 'shape',
      if ~ischar(content),
    msg = '''shape'' should be a string';
      elseif ~strcmp(content,'sheet') & ~strcmp(content,'cyl') & ...
        ~strcmp(content,'toroid'),
    msg = ['Unknown shape type: ' content];
    sS.shape = content; isok = 1;
      else
    sS.shape = content; isok = 1;
      end
     otherwise,
      msg = ['Invalid field for topology struct: ' field];
    end

   case 'som_train',
    switch field,
     case 'algorithm',
      if ~ischar(content),
    msg = '''algorithm'' should be a string.';
      else
    sS.algorithm = content; isok = 1;
      end
     case 'data_name',
      if ~ischar(content),
    msg = '''data_name'' should be a string';
      else
    sS.data_name = content; isok = 1;
      end
     case 'neigh',
      if ~ischar(content),
    msg = '''neigh'' should be a string';
      elseif ~isempty(content) & ~strcmp(content,'gaussian') & ~strcmp(content,'ep')
& ...
        ~strcmp(content,'cutgauss') & ~strcmp(content,'bubble'),
    msg = ['Unknown neighborhood function: ' content];
    sS.neigh = content; isok = 1;
      else
    sS.neigh = content; isok = 1;
      end
     case 'mask',
      if size(content,1) == 1, content = content'; end
      dim = size(content,1); %[munits dim] = size(sS.data);
      if ~isnumeric(content) | size(content) ~= [dim 1],
    msg = '''mask'' should be a column vector (size dim x 1).';
      else
    sS.mask = content; isok = 1;
      end
     case 'radius_ini',
      if ~isnumeric(content) | ~isscalar,
    msg = '''radius_ini'' should be a scalar.';
      else
    sS.radius_ini = content; isok = 1;
      end
     case 'radius_fin',
      if ~isnumeric(content) | ~isscalar,
    msg = '''radius_fin'' should be a scalar.';
      else
    sS.radius_fin = content; isok = 1;
      end
     case 'alpha_ini',
      if ~isnumeric(content) | ~isscalar,
    msg = '''alpha_ini'' should be a scalar.';
      else
    sS.alpha_ini = content; isok = 1;
      end
     case 'alpha_type',
      if ~ischar(content),
    msg = '''alpha_type'' should be a string';
      elseif ~strcmp(content,'linear') & ~strcmp(content,'inv') & ...
        ~strcmp(content,'power') & ~strcmp(content,'constant') & ~strcmp(content,''),
    msg = ['Unknown alpha type: ' content];
    sS.alpha_type = content; isok = 1;
      else
    sS.alpha_type = content; isok = 1;
      end
     case 'trainlen',
      if ~isnumeric(content) | ~isscalar,
    msg = '''trainlen'' should be a scalar.';
      else
    sS.trainlen = content; isok = 1;
      end
     case 'time',
      if ~ischar(content),
    msg = '''time'' should be a string';
      else
    sS.time = content; isok = 1;
      end
     otherwise,
      msg = ['Invalid field for train struct: ' field];
    end

   case 'som_norm',
    switch field,
     case 'method',
      if ~ischar(field),
    msg = '''method'' should be a string.';
      else
    sS.method = content; isok = 1;
      end
     case 'params',
      sS.params = content; isok = 1;
     case 'status',
      if ~ischar(content),
    msg = '''status'' should be a string';
      elseif ~strcmp(content,'done') & ~strcmp(content,'undone') & ...
        ~strcmp(content,'uninit'),
    msg = ['Unknown status type: ' content];
    sS.status = content; isok = 1;
      else
    sS.status = content; isok = 1;
      end
     otherwise,
      msg = ['Invalid field for normalization struct: ' field];
    end

   case 'som_grid',
    if any(strcmp(field,{'lattice', 'shape', 'msize', 'coord',...
             'line', 'linecolor', 'linewidth', ...
             'marker', 'markersize', 'markercolor', 'surf', ...
             'label', 'labelcolor', 'labelsize'})),
      warning('No checking done on field identifier or content.');
      sS = setfield(sS,field,content);
      isok = 1;
    else
      msg = ['Invalid field for grid struct: ' field];
    end

   otherwise,
    error('Unrecognized structure.');

  end

  msgs{i} = msg;
  ok(i) = isok;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% return

if nargout < 2,
  for i=1:p,
    if ~isempty(msgs{i}),
      if ~ok(i), fprintf(1,'[Error! ');
      else fprintf(1,'[Notice ');
      end
      fprintf(1,'in setting %s] ',varargin{2*i-1});
      fprintf(1,'%s\n',msgs{i});
    end
  end
end
```

Function8- SOM_TOPOL_STRUCT Default values for SOM topology.

```
function sTopol = som_topol_struct(varargin)

%SOM_TOPOL_STRUCT Default values for SOM topology.

%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_topol_struct
%
% PURPOSE
%
% Default values for map topology and training parameters.
%
% SYNTAX
%
%  sT = som_topol_struct('argID',value,...);
%  sT = som_topol_struct(value,...);
%
% DESCRIPTION
%
% This function is used to give sensible values for map topology (ie. map
% size). The topology struct is returned.
%
% The topology struct has three fields: '.msize', '.lattice' and
% '.shape'. Of these, default value for '.lattice' is 'hexa' and for
% '.shape' 'sheet'. Only the '.msize' field depends on the optional
% arguments: 'dlen', 'munits' and 'data'.  The value for '.msize' field is
% determined as follows.
%
% First, the number of map units is determined (unless it is given). A
% heuristic formula of 'munits = 5*sqrt(dlen)' is used to calculate
% it. After this, the map size is determined. Basically, the two biggest
% eigenvalues of the training data are calculated and the ratio between
% sidelengths of the map grid is set to the square root of this ratio. The
% actual sidelengths are then set so that their product is as close to the
% desired number of map units as possible. If the lattice of the grid is
% 'hexa', the ratio is modified a bit to take it into account. If the
% lattice is 'hexa' and shape is 'toroid', the map size along the first axis
% must be even.
%
% OPTIONAL INPUT ARGUMENTS
%
%  argID (string) Argument identifier string (see below).
%  value (varies) Value for the argument (see below).
%
%  The optional arguments can be given as 'argID',value -pairs. If an
%  argument is given value multiple times, the last one is
%  used. The valid IDs and corresponding values are listed below. The values
%  which are unambiguous (marked with '*') can be given without the
%  preceeding argID.
%
%  'dlen'          (scalar) length of the training data
%  'data'          (matrix) the training data
%                 *(struct) the training data
%  'munits'        (scalar) number of map units
%  'msize'         (vector) map size
%  'lattice'      *(string) map lattice: 'hexa' or 'rect'
%  'shape'        *(string) map shape: 'sheet', 'cyl' or 'toroid'
%  'topol'        *(struct) incomplete topology struct: its empty fields
%                           will be given values
%  'som_topol','sTopol'     = 'topol'
%
% OUTPUT ARGUMENTS
%
%  sT     (struct) The topology struct.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% check arguments
```

```
% initialize
sTopol = som_set('som_topol','lattice','hexa','shape','sheet');
D = [];
dlen = NaN;
dim = 2;
munits = NaN;

% varargin
i=1;
while i<=length(varargin),
  argok = 1;
  if ischar(varargin{i}),
    switch varargin{i},
     case 'dlen',         i=i+1; dlen = varargin{i};
     case 'munits',       i=i+1; munits = varargin{i}; sTopol.msize = 0;
     case 'msize',        i=i+1; sTopol.msize = varargin{i};
     case 'lattice',      i=i+1; sTopol.lattice = varargin{i};
     case 'shape',        i=i+1; sTopol.shape = varargin{i};
     case 'data',
      i=i+1;
      if isstruct(varargin{i}), D = varargin{i}.data;
      else D = varargin{i};
      end
      [dlen dim] = size(D);
     case {'hexa','rect'}, sTopol.lattice = varargin{i};
     case {'sheet','cyl','toroid'}, sTopol.shape = varargin{i};
     case {'som_topol','sTopol','topol'},
      i=i+1;
      if ~isempty(varargin{i}.msize) & prod(varargin{i}.msize),
    sTopol.msize = varargin{i}.msize;
      end
      if ~isempty(varargin{i}.lattice), sTopol.lattice = varargin{i}.lattice; end
      if ~isempty(varargin{i}.shape), sTopol.shape = varargin{i}.shape; end
     otherwise argok=0;
    end
  elseif isstruct(varargin{i}) & isfield(varargin{i},'type'),
    switch varargin{i}.type,
     case 'som_topol',
      if ~isempty(varargin{i}.msize) & prod(varargin{i}.msize),
    sTopol.msize = varargin{i}.msize;
      end
      if ~isempty(varargin{i}.lattice), sTopol.lattice = varargin{i}.lattice; end
      if ~isempty(varargin{i}.shape), sTopol.shape = varargin{i}.shape; end
     case 'som_data',
      D = varargin{i}.data;
      [dlen dim] = size(D);
     otherwise argok=0;
    end
  else
    argok = 0;
  end
  if ~argok,
    disp(['(som_topol_struct) Ignoring invalid argument #' num2str(i)]);
  end
  i = i+1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% action - topology struct

% lattice and shape set already, so if msize is also set, there's
% nothing else to do
if prod(sTopol.msize) & ~isempty(sTopol.msize), return; end

% otherwise, decide msize
% first (if necessary) determine the number of map units (munits)
if isnan(munits),
  if ~isnan(dlen),
    munits = ceil(5 * dlen^0.5); % this is just one way to make a guess...
  else
    munits = 100; % just a convenient value
  end
end

% then determine the map size (msize)
if dim == 1, % 1-D data

  sTopol.msize = [1 ceil(munits)];

elseif size(D,1)<2, % eigenvalues cannot be determined since there's no data

  sTopol.msize = round(sqrt(munits));
  sTopol.msize(2) = round(munits/sTopol.msize(1));

else % determine map size based on eigenvalues

  % initialize xdim/ydim ratio using principal components of the input
  % space; the ratio is the square root of ratio of two largest eigenvalues

  % autocorrelation matrix
  A = zeros(dim)+Inf;
  for i=1:dim, D(:,i) = D(:,i) - mean(D(isfinite(D(:,i)),i)); end
  for i=1:dim,
    for j=i:dim,
      c = D(:,i).*D(:,j); c = c(isfinite(c));
      A(i,j) = sum(c)/length(c); A(j,i) = A(i,j);
    end
  end
  % take mdim first eigenvectors with the greatest eigenvalues
  [V,S]   = eig(A);
  eigval  = diag(S);
  [y,ind] = sort(eigval);
  eigval  = eigval(ind);

  %me     = mean(D);
  %D      = D - me(ones(length(ind),1),:); % remove mean from data
  %eigval = sort(eig((D'*D)./size(D,1)));
  if eigval(end)==0 | eigval(end-1)*munits<eigval(end),
    ratio = 1;
  else
    ratio  = sqrt(eigval(end)/eigval(end-1)); % ratio between map sidelengths
  end

  % in hexagonal lattice, the sidelengths are not directly
  % proportional to the number of units since the units on the
  % y-axis are squeezed together by a factor of sqrt(0.75)
  if strcmp(sTopol.lattice,'hexa'),
    sTopol.msize(2)  = min(munits, round(sqrt(munits / ratio * sqrt(0.75))));
  else
    sTopol.msize(2)  = min(munits, round(sqrt(munits / ratio)));
  end
  sTopol.msize(1)  = round(munits / sTopol.msize(2));

  % if actual dimension of the data is 1, make the map 1-D
  if min(sTopol.msize) == 1, sTopol.msize = [1 max(sTopol.msize)]; end;

  % a special case: if the map is toroid with hexa lattice,
  % size along first axis must be even
  if strcmp(sTopol.lattice,'hexa') & strcmp(sTopol.shape,'toroid'),
    if mod(sTopol.msize(1),2), sTopol.msize(1) = sTopol.msize(1) + 1; end
  end

end

return;

Function9- SOM_TRAIN_STRUCT Default values for SOM training parameters.
function sTrain = som_train_struct(varargin)

%SOM_TRAIN_STRUCT Default values for SOM training parameters.

%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_train_struct
%
% PURPOSE
%
% Default values for SOM training parameters.
%
% SYNTAX
%
%  sT = som_train_struct('argID',value,...);
%  sT = som_train_struct(value,...);
%
% DESCRIPTION
%
% This function is used to give sensible values for SOM training
% parameters and returns a training struct. Often, the parameters
% depend on the properties of the map and the training data. These are
% given as optional arguments to the function. If a partially filled
% train struct is given, its empty fields (field value is [] or '' or
% NaN) are supplimented with default values.
%
% The training struct has a number of fields which depend on each other
% and the optional arguments in complex ways. The most important argument
% is 'phase' which can be either 'init', 'train', 'rough' or 'finetune'.
%
%  'init'     Map initialization.
%  'train'    Map training in a onepass operation, as opposed to the
%             rough-finetune combination.
%  'rough'    Rough organization of the map: large neighborhood, big
%             initial value for learning coefficient. Short training.
%  'finetune' Finetuning the map after rough organization phase. Small
%             neighborhood, learning coefficient is small already at
%             the beginning. Long training.
%
% The fields of training struct set by this function are listed below.
%
%  '.mask'  Basically, a column vector of ones. But if a previous
%           train or map struct is given, it is copied from there.
%  '.neigh' Default value is 'gaussian' but if a previous train or map
%           struct is given, it is copied from there.
%  '.alpha_type' Default value is 'inv' but if a previous training struct
%           is given, it is copied from there.
%  '.alpha_ini' For 'train' and 'rough' phases, this is 0.5, for
%           'finetune' it is 0.05.
%  '.radius_ini' Depends on the previous training operation and the
%           maximum sidelength of the map ms = max(msize).
%           if there isn't one, or it is 'randinit', rad_ini = max(1,ms/2)
%           if it is 'lininit', rad_ini = max(1,ms/8)
%           otherwise, rad_ini = rad_fin of the previous training
%  '.radius_fin' Default value is 1, but if the training phase is
%           'rough', rad_fin = max(1,rad_ini/4).
%  '.trainlen' For 'train' phase this is 20 x mpd epochs, for 'rough'
%           phase 4 x mpd epochs and for 'finetune' 16 x mpd
%           epochs, where mpd = munits/dlen. If mpd cannot be
%           calculated, it is set to be = 0.5. In any case,
%           trainlen is at least one epoch.
%  '.algorithm' Default training algorithm is 'batch' and default
%           initialization algorithm is 'lininit'.
%
% OPTIONAL INPUT ARGUMENTS
%
%  argID (string) Argument identifier string (see below).
%  value (varies) Value for the argument (see below).
%
%  The optional arguments can be given as 'argID',value -pairs. If an
%  argument is given value multiple times, the last one is used.  The
%  valid IDs and corresponding values are listed below. The values
%  which are unambiguous (marked with '*') can be given without the
%  preceeding argID.
%
%  'dim'          (scalar) input space dimension
%  'dlen'         (scalar) length of the training data
%  'data'         (matrix / struct) the training data
%  'munits'       (scalar) number of map units
%  'msize'        (vector) map size
%  'previous'     (struct) previous training struct can be given in
%                  conjunction with 'finetune' phase.
%  'phase'       *(string) training phase: 'init', 'train', 'rough' or 'finetune'
%  'algorithm'   *(string) algorithm to use: 'lininit', 'randinit',
%                  'batch' or 'seq'
%  'map'         *(struct) If a map struc is given, the last training struct
%                  in '.trainhist' field is used as the previous training
%                  struct. The map size and input space dimension are
%                  extracted from the map struct.
%  'sTrain'      *(struct) a train struct, the empty fields of which are
%                  filled with sensible values
%
% OUTPUT ARGUMENTS
%
%  sT     (struct) The training struct.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% check arguments

% initial default structs
sTrain = som_set('som_train');

% initialize optional parameters
dlen = NaN;
msize = 0;
munits = NaN;
sTprev = [];
dim = NaN;
phase = '';

% varargin
i=1;
while i<=length(varargin),
  argok = 1;
  if ischar(varargin{i}),
    switch varargin{i},
     case 'dim',        i=i+1; dim = varargin{i};
     case 'dlen',       i=i+1; dlen = varargin{i};
     case 'msize',      i=i+1; msize = varargin{i};
     case 'munits',     i=i+1; munits = varargin{i}; msize = 0;
     case 'phase',      i=i+1; phase = varargin{i};
     case 'algorithm',  i=i+1; sTrain.algorithm = varargin{i};
     case 'mask',       i=i+1; sTrain.mask = varargin{i};
     case {'previous','map'},
      i=i+1;
      if strcmp(varargin{i}.type,'som_map'),
    if length(varargin{i}.trainhist),
      sTprev = varargin{i}.trainhist(end);
      msize = varargin{i}.topol.msize;
    end
      elseif strcmp(varargin{i}.type,'som_train'),
    sTprev = varargin{i};
      end
     case 'data',
      i=i+1;
      if isstruct(varargin{i}), [dlen dim] = size(varargin{i}.data);
      else [dlen dim] = size(varargin{i});
      end
     case {'init','train','rough','finetune'},  phase = varargin{i};
     case {'lininit','randinit','seq','batch'}, sTrain.algorithm = varargin{i};
     otherwise argok=0;
    end
  elseif isstruct(varargin{i}) & isfield(varargin{i},'type'),
    switch varargin{i}.type,
     case 'som_train',
      sT = varargin{i};
      if ~isempty(sT.algorithm),  sTrain.algorithm = sT.algorithm; end
      if ~isempty(sT.neigh),      sTrain.neigh = sT.neigh; end
      if ~isempty(sT.mask),       sTrain.mask = sT.mask; end
      if ~isnan(sT.radius_ini),   sTrain.radius_ini = sT.radius_ini; end
      if ~isnan(sT.radius_fin),   sTrain.radius_fin = sT.radius_fin; end
      if ~isnan(sT.alpha_ini),    sTrain.alpha_ini = sT.alpha_ini; end
      if ~isempty(sT.alpha_type), sTrain.alpha_type = sT.alpha_type; end
      if ~isnan(sT.trainlen),     sTrain.trainlen = sT.trainlen; end
      if ~isempty(sT.data_name),  sTrain.data_name = sT.data_name; end
      if ~isempty(sT.time),       sTrain.time = sT.time; end
     case 'som_map',
      if strcmp(varargin{i}.type,'som_map'),
    if length(varargin{i}.trainhist),
      sTprev = varargin{i}.trainhist(end);
      msize = varargin{i}.topol.msize;
    end
    if ~isempty(varargin{i}.neigh) & isempty(sTrain.neigh),
      sTrain.neigh = varargin{i}.neigh;
    end
    if ~isempty(varargin{i}.mask) & isempty(sTrain.mask),
      sTrain.mask = varargin{i}.mask;
    end
      elseif strcmp(varargin{i}.type,'som_train'),
    sTprev = varargin{i};
      end
     case 'som_topol', msize = varargin{i}.msize;
     case 'som_data', [dlen dim] = size(varargin{i}.data);
     otherwise argok=0;
    end
  else
    argok = 0;
  end
  if ~argok,
    disp(['(som_train_struct) Ignoring invalid argument #' num2str(i)]);
  end
  i = i+1;
end

% dim
if ~isempty(sTprev) & isnan(dim), dim = length(sTprev.mask); end

% mask
if isempty(sTrain.mask) & ~isnan(dim), sTrain.mask = ones(dim,1); end

% msize, munits
if ~msize | isempty(msize),
  if isnan(munits), msize = [10 10];
  else s = round(sqrt(munits)); msize = [s round(munits/s)];
  end
end
munits = prod(msize);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% action

% previous training
prevalg = '';
if ~isempty(sTprev),
  if any(findstr(sTprev.algorithm,'init')), prevalg = 'init';
  else prevalg = sTprev.algorithm;
  end
end

% first determine phase
if isempty(phase),
  switch sTrain.algorithm,
   case {'lininit','randinit'},    phase = 'init';
   case {'batch','seq',''},
    if     isempty(sTprev),        phase = 'rough';
    elseif strcmp(prevalg,'init'), phase = 'rough';
    else                           phase = 'finetune';
    end
   otherwise,                      phase = 'train';
  end
end

% then determine algorithm
if isempty(sTrain.algorithm),
  if     strcmp(phase,'init'),             sTrain.algorithm = 'lininit';
  elseif any(strcmp(prevalg,{'init',''})), sTrain.algorithm = 'batch';
  else sTrain.algorithm = sTprev.algorithm;
  end
end

% mask
if isempty(sTrain.mask),
  if ~isempty(sTprev), sTrain.mask = sTprev.mask;
  elseif ~isnan(dim),  sTrain.mask = ones(dim,1);
  end
end

% neighborhood function
if isempty(sTrain.neigh),
  if ~isempty(sTprev) & ~isempty(sTprev.neigh), sTrain.neigh = sTprev.neigh;
  else sTrain.neigh = 'gaussian';
  end
end

if strcmp(phase,'init'),
  sTrain.alpha_ini = NaN;
  sTrain.alpha_type = '';
  sTrain.radius_ini = NaN;
  sTrain.radius_fin = NaN;
  sTrain.trainlen = NaN;
  sTrain.neigh = '';
else
  mode = [phase, '-', sTrain.algorithm];

  % learning rate
  if isnan(sTrain.alpha_ini),
    if strcmp(sTrain.algorithm,'batch'), sTrain.alpha_ini = NaN;
    else
      switch phase,
       case {'train','rough'}, sTrain.alpha_ini = 0.5;
       case 'finetune',        sTrain.alpha_ini = 0.05;
      end
    end
  end
  if isempty(sTrain.alpha_type),
    if ~isempty(sTprev) & ~isempty(sTprev.alpha_type) ...
      & ~strcmp(sTrain.algorithm,'batch'),
      sTrain.alpha_type = sTprev.alpha_type;
    elseif strcmp(sTrain.algorithm,'seq'),
      sTrain.alpha_type = 'inv';
    end
  end

  % radius
  ms = max(msize);
  if isnan(sTrain.radius_ini),
    if isempty(sTprev) | strcmp(sTprev.algorithm,'randinit'),
      sTrain.radius_ini = max(1,ceil(ms/4));
    elseif strcmp(sTprev.algorithm,'lininit') | isnan(sTprev.radius_fin),
      sTrain.radius_ini = max(1,ceil(ms/8));
    else
      sTrain.radius_ini = sTprev.radius_fin;
    end
  end
  if isnan(sTrain.radius_fin),
    if strcmp(phase,'rough'),
      sTrain.radius_fin = max(1,sTrain.radius_ini/4);
    else
      sTrain.radius_fin = 1;
    end
  end

  % trainlen
  if isnan(sTrain.trainlen),
    mpd = munits/dlen;
    if isnan(mpd), mpd = 0.5; end
    switch phase,
     case 'train',    sTrain.trainlen = ceil(50*mpd);
```

```
      case 'rough',     sTrain.trainlen = ceil(10*mpd);
      case 'finetune', sTrain.trainlen = ceil(40*mpd);
     end
     sTrain.trainlen = max(1,sTrain.trainlen);
   end

end

return;


Function10- SOM_UNIT_COORDS Locations of units on the SOM grid.
function Coords = som_unit_coords(topol,lattice,shape)

%SOM_UNIT_COORDS Locations of units on the SOM grid.

%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_unit_coords
%
% PURPOSE
%
% Returns map grid coordinates for the units of a Self-Organizing Map.
%
% SYNTAX
%
%  Co = som_unit_coords(sTopol);
%  Co = som_unit_coords(sM.topol);
%  Co = som_unit_coords(msize);
%  Co = som_unit_coords(msize,'hexa');
%  Co = som_unit_coords(msize,'rect','toroid');
%
% DESCRIPTION
%
% Calculates the map grid coordinates of the units of a SOM based on
% the given topology. The coordinates are such that they can be used to
% position map units in space. In case of 'sheet' shape they can be
% (and are) used to measure interunit distances.
%
% NOTE: for 'hexa' lattice, the x-coordinates of every other row are shifted
% by +0.5, and the y-coordinates are multiplied by sqrt(0.75). This is done
% to make distances of a unit to all its six neighbors equal. It is not
% possible to use 'hexa' lattice with higher than 2-dimensional map grids.
%
% 'cyl' and 'toroid' shapes: the coordinates are initially determined as
% in case of 'sheet' shape, but are then bended around the x- or the
% x- and then y-axes to get the desired shape.
%
% POSSIBLE BUGS
%
% I don't know if the bending operation works ok for high-dimensional
% map grids. Anyway, if anyone wants to make a 4-dimensional
% toroid map, (s)he deserves it.
%
% REQUIRED INPUT ARGUMENTS
%
%  topol          Map grid dimensions.
%        (struct) topology struct or map struct, the topology
%                 (msize, lattice, shape) of the map is taken from
%                 the appropriate fields (see e.g. SOM_SET)
%        (vector) the vector which gives the size of the map grid
%                 (msize-field of the topology struct).
%
% OPTIONAL INPUT ARGUMENTS
%
%  lattice (string) The map lattice, either 'rect' or 'hexa'. Default
%                   is 'rect'. 'hexa' can only be used with 1- or
%                   2-dimensional map grids.
%  shape   (string) The map shape, either 'sheet', 'cyl' or 'toroid'.
%                   Default is 'sheet'.
%
% OUTPUT ARGUMENTS
%
%  Co   (matrix) coordinates for each map units, size is [munits k]
%                where k is 2, or more if the map grid is higher
%                dimensional or the shape is 'cyl' or 'toroid'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Check arguments

error(nargchk(1, 3, nargin));

% default values
sTopol = som_set('som_topol','lattice','rect');

% topol
if isstruct(topol),
  switch topol.type,
  case 'som_map', sTopol = topol.topol;
  case 'som_topol', sTopol = topol;
  end
elseif iscell(topol),
  for i=1:length(topol),
    if isnumeric(topol{i}), sTopol.msize = topol{i};
    elseif ischar(topol{i}),
      switch topol{i},
      case {'rect','hexa'}, sTopol.lattice = topol{i};
      case {'sheet','cyl','toroid'}, sTopol.shape = topol{i};
      end
    end
  end
else
  sTopol.msize = topol;
end
if prod(sTopol.msize)==0, error('Map size is 0.'); end

% lattice
if nargin>1 & ~isempty(lattice) & ~isnan(lattice), sTopol.lattice = lattice; end

% shape
if nargin>2 & ~isempty(shape) & ~isnan(shape), sTopol.shape = shape; end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Action

msize = sTopol.msize;
lattice = sTopol.lattice;
shape = sTopol.shape;

% init variables

if length(msize)==1, msize = [msize 1]; end
munits = prod(msize);
mdim = length(msize);
Coords = zeros(munits,mdim);

% initial coordinates for each map unit ('rect' lattice, 'sheet' shape)
k = [1 cumprod(msize(1:end-1))];
inds = [0:(munits-1)]';
for i = mdim:-1:1,
  Coords(:,i) = floor(inds/k(i)); % these are subscripts in matrix-notation
  inds = rem(inds,k(i));
end
% change subscripts to coordinates (move from (ij)-notation to (xy)-notation)
Coords(:,[1 2]) = fliplr(Coords(:,[1 2]));

% 'hexa' lattice
if strcmp(lattice,'hexa'),
  % check
  if mdim > 2,
    error('You can only use hexa lattice with 1- or 2-dimensional maps.');
  end
  % offset x-coordinates of every other row
  inds_for_row = (cumsum(ones(msize(2),1))-1)*msize(1);
  for i=2:2:msize(1),
    Coords(i+inds_for_row,1) = Coords(i+inds_for_row,1) + 0.5;
  end
end

% shapes
switch shape,
case 'sheet',
  if strcmp(lattice,'hexa'),
    % this correction is made to make distances to all
    % neighboring units equal
    Coords(:,2) = Coords(:,2)*sqrt(0.75);
  end

case 'cyl',
  % to make cylinder the coordinates must lie in 3D space, at least
  if mdim<3, Coords = [Coords ones(munits,1)]; mdim = 3; end

  % Bend the coordinates to a circle in the plane formed by x- and
  % and z-axis. Notice that the angle to which the last coordinates
  % are bended is _not_ 360 degrees, because that would be equal to
  % the angle of the first coordinates (0 degrees).

  Coords(:,1)     = Coords(:,1)/max(Coords(:,1));
  Coords(:,1)     = 2*pi * Coords(:,1) * msize(2)/(msize(2)+1);
  Coords(:,[1 3]) = [cos(Coords(:,1)) sin(Coords(:,1))];

case 'toroid',

  % NOTE: if lattice is 'hexa', the msize(1) should be even, otherwise
  % the bending the upper and lower edges of the map do not match
  % to each other
  if strcmp(lattice,'hexa') & rem(msize(1),2)==1,
    warning('Map size along y-coordinate is not even.');
  end

  % to make toroid the coordinates must lie in 3D space, at least
  if mdim<3, Coords = [Coords ones(munits,1)]; mdim = 3; end

  % First bend the coordinates to a circle in the plane formed
  % by x- and z-axis. Then bend in the plane formed by y- and
  % z-axis. (See also the notes in 'cyl').

  Coords(:,1)     = Coords(:,1)/max(Coords(:,1));
  Coords(:,1)     = 2*pi * Coords(:,1) * msize(2)/(msize(2)+1);
  Coords(:,[1 3]) = [cos(Coords(:,1)) sin(Coords(:,1))];

  Coords(:,2)     = Coords(:,2)/max(Coords(:,2));
  Coords(:,2)     = 2*pi * Coords(:,2) * msize(1)/(msize(1)+1);
  Coords(:,3)     = Coords(:,3) - min(Coords(:,3)) + 1;
  Coords(:,[2 3]) = Coords(:,[3 3]) .* [cos(Coords(:,2)) sin(Coords(:,2))];

end

return;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% subfunctions

function C = bend(cx,cy,angle,xishexa)

  dx = max(cx) - min(cx);
  if dx ~= 0,
    % in case of hexagonal lattice it must be taken into account that
    % coordinates of every second row are +0.5 off to the right
    if xishexa, dx = dx-0.5; end
    cx = angle*(cx - min(cx))/dx;
  end
  C(:,1) = (cy - min(cy)+1) .* cos(cx);
  C(:,2) = (cy - min(cy)+1) .* sin(cx);

% end of bend

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Function11- SOM_UNIT_DISTS Distances between unit-locations on the map grid.
function Ud = som_unit_dists(topol,lattice,shape)

%SOM_UNIT_DISTS Distances between unit-locations on the map grid.

%%%%%%%%%%%%% DETAILED DESCRIPTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% som_unit_dists
%
% PURPOSE
%
% Returns interunit distances between the units of a Self-Organizing Map
% along the map grid.
%
% SYNTAX
%
%  Ud = som_unit_dists(sTopol);
%  Ud = som_unit_dists(sM.topol);
%  Ud = som_unit_dists(msize);
%  Ud = som_unit_dists(msize,'hexa');
%  Ud = som_unit_dists(msize,'rect','toroid');
%
% DESCRIPTION
%
% Calculates the distances between the units of a SOM based on the
% given topology. The distance are euclidian and they are measured
% along the map grid (in the output space).
%
% In case of 'sheet' shape, the distances can be measured directly
% from the unit coordinates given by SOM_UNIT_COORDS.
%
% In case of 'cyl' and 'toroid' shapes this is not so. In these cases
% the coordinates are calculated as in the case of 'sheet' shape and
% the shape is then taken into account by shifting the map grid into
% different positions.
%
% Consider, for example, a 4x3 map. The basic position of map units
% is shown on the left (with '1' - 'C' each denoting one map unit).
% In case of a 'cyl' shape, units on the left and right edges are
% neighbors, so for this purpose the map is copied on the left and
% right sides of the map, as on right.
%
%    basic              left     basic    right
%    -------            -------  -------  -------
%    1  5  9            1  5  9  1  5  9  1  5  9
%    2  6  a            2  6  a  2  6  a  2  6  a
%    3  7  b            3  7  b  3  7  b  3  7  b
%    4  8  c            4  8  c  4  8  c  4  8  c
%
% For the 'toroid' shape a similar trick is done, except that the
% copies are placed all around the basic position:
%
%             1  5  9  1  5  9  1  5  9
%             2  6  a  2  6  a  2  6  a
%             3  7  b  3  7  b  3  7  b
%             4  8  c  4  8  c  4  8  c
%             1  5  9  1  5  9  1  5  9
%             2  6  a  2  6  a  2  6  a
%             3  7  b  3  7  b  3  7  b
%             4  8  c  4  8  c  4  8  c
%             1  5  9  1  5  9  1  5  9
%             2  6  a  2  6  a  2  6  a
%             3  7  b  3  7  b  3  7  b
%             4  8  c  4  8  c  4  8  c
%
% From this we can see that the distance from unit '1' is 1 to units
% '9','2','4' and '5', and sqrt(2) to units 'C','A','8' and '6'. Notice
% that in the case of a 'hexa' lattice and 'toroid' shape, the size
% of the map in y-direction should be even. The reason can be clearly
% seen from the two figures below. On the left the basic positions for
% a 3x3 map. If the map is copied above itself, it can be seen that the
% lattice is broken (on the right):
%
%     basic positions                example of broken lattice
%     ---------------                -------------------------
%                                    1  4  7
%                                     2  5  8
%                                    3  6  9
%     1  4  7                        1  4  7
%      2  5  8                        2  5  8
%     3  6  9                        3  6  9
%
%
% REQUIRED INPUT ARGUMENTS
%
%  topol          Map grid dimensions.
%        (struct) topology struct or map struct, the topology
%                 (msize, lattice, shape) of the map is taken from
%                 the appropriate fields (see e.g. SOM_SET)
%        (vector) the vector which gives the size of the map grid
%                 (msize-field of the topology struct).
%
% OPTIONAL INPUT ARGUMENTS
%
%  lattice (string) The map lattice, either 'rect' or 'hexa'. Default
%                   is 'rect'. 'hexa' can only be used with 1- or
%                   2-dimensional map grids.
%  shape   (string) The map shape, either 'sheet', 'cyl' or 'toroid'.
%                   Default is 'sheet'.
%
% OUTPUT ARGUMENTS
%
%  Ud   (matrix) distances from each map unit to each other map unit,
%                size is [munits munits]
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Check arguments

error(nargchk(1, 3, nargin));

% default values
sTopol = som_set('som_topol','lattice','rect');

% topol

if isstruct(topol),
  switch topol.type,
  case 'som_map', sTopol = topol.topol;
  case 'som_topol', sTopol = topol;
  end
elseif iscell(topol),
  for i=1:length(topol),
    if isnumeric(topol{i}), sTopol.msize = topol{i};
    elseif ischar(topol{i}),
      switch topol{i},
      case {'rect','hexa'}, sTopol.lattice = topol{i};
      case {'sheet','cyl','toroid'}, sTopol.shape = topol{i};
      end
    end
  end
else
  sTopol.msize = topol;
end
if prod(sTopol.msize)==0, error('Map size is 0.'); end

% lattice
if nargin>1 & ~isempty(lattice) & ~isnan(lattice), sTopol.lattice = lattice; end

% shape
if nargin>2 & ~isempty(shape) & ~isnan(shape), sTopol.shape = shape; end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Action

msize = sTopol.msize;
lattice = sTopol.lattice;
shape = sTopol.shape;

munits = prod(msize);
Ud = zeros(munits,munits);

% free topology
if strcmp(lattice,'free'),
  N1 = sTopol.connection;
  Ud = som_neighborhood(N1,Inf);
end

% coordinates of map units when the grid is spread on a plane
Coords = som_unit_coords(msize,lattice,'sheet');

% width and height of the grid
dx = max(Coords(:,1))-min(Coords(:,1));
if msize(1)>1, dx = dx*msize(1)/(msize(1)-1); else dx = dx+1; end
dy = max(Coords(:,2))-min(Coords(:,2));
if msize(2)>1, dy = dy*msize(2)/(msize(2)-1); else dy = dy+1; end

% calculate distance from each location to each other location
switch shape,
case 'sheet',
  for i=1:(munits-1),
    inds = [(i+1):munits];
    Dco = (Coords(inds,:) - Coords(ones(munits-i,1)*i,:))';
    Ud(i,inds) = sqrt(sum(Dco.^2));
  end

case 'cyl',
  for i=1:(munits-1),
    inds = [(i+1):munits];
    Dco  = (Coords(inds,:) - Coords(ones(munits-i,1)*i,:))';
    dist = sum(Dco.^2);
    % The cylinder shape is taken into account by adding and substracting
    % the width of the map (dx) from the x-coordinate (ie. shifting the
    % map right and left).
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) + dx;  %East (x+dx)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) - dx;  %West (x-dx)
    dist = min(dist,sum(DcoS.^2));
    Ud(i,inds) = sqrt(dist);
  end

case 'toroid',
  for i=1:(munits-1),
    inds = [(i+1):munits];
    Dco  = (Coords(inds,:) - Coords(ones(munits-i,1)*i,:))';
    dist = sum(Dco.^2);
    % The toroid shape is taken into account as the cylinder shape was
    % (see above), except that the map is shifted also vertically.
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) + dx;  %East (x+dx)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) - dx;  %West (x+dx)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(2,:) = DcoS(2,:) + dy;  %South (y+dy)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(2,:) = DcoS(2,:) - dy;  %North (y-dy)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) + dx; DcoS(2,:) = DcoS(2,:) - dy; %NorthEast
(x+dx, y-dy)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) + dx; DcoS(2,:) = DcoS(2,:) + dy; %SouthEast
(x+dx, y+dy)
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) - dx; DcoS(2,:) = DcoS(2,:) + dy; %SouthWest
(x-dx, y+dy)
```

```
    dist = min(dist,sum(DcoS.^2));
    DcoS = Dco; DcoS(1,:) = DcoS(1,:) - dx; DcoS(2,:) = DcoS(2,:) - dy; %NorthWest
(x-dx, y-dy)
    dist = min(dist,sum(DcoS.^2));
    Ud(i,inds) = sqrt(dist);
  end

otherwise,
  error (['Unknown shape: ', shape]);

end

Ud = Ud + Ud';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Function12- VIS_VALUETYPE Used for type checks in SOM Toolbox visualization routines.
function flag=vis_valuetype(value, valid, str);

% VIS_VALUETYPE Used for type checks in SOM Toolbox visualization routines
%
% flag = vis_valuetype(value, valid, str)
%
% Input and output arguments:
% value (varies) variable to be checked
% valid (cell array) size 1xN, cells are strings or vectors (see below)
% str (string) 'all' or 'any' (default), determines whether
% all or just any of the types listed in argument 'valid'
% should be true for 'value'
%
% flag (scalar) 1 or 0 (true or false)
%
% This is an internal function of SOM Toolbox visualization. It makes
% various type checks. For example:
%
% % Return 1 if X is a numeric scalar otherwise 0:
% f=vis_valuetype(X,{'1x1'});
%
% % Return 1 if X is a ColorSpec, that is, a 1x3 vector presenting an RGB
% % value or any of strings 'red','blue','green','yellow','magenta','cyan',
% % 'white' or 'black' or their shortenings 'r','g','b','y','m','c','w','k':
% f=vis_valueype(X,{'1x3rgb','colorstyle'})
%
% % Return 1 if X is _both_ 10x3 size numeric matrix and has RGB values as rows
% f=vis_valuetype(X,{'nx3rgb',[10 3]},'all')
%
% Strings that may be used in argument valid:
% id is true if value is
%
% [n1 n2 ... nn] any n1 x n2 x ... x nn sized numeric matrix
% '1x1' scalar (numeric)
% '1x2' 1x2 vector (numeric)
% 'nx1' any nx1 numeric vector
% 'nx2' nx2
% 'nx3' nx3
% 'nxn' any numeric square matrix
% 'nxn[0,1]' numeric square matrix with values in interval [0,1]
% 'nxm' any numeric matrix
% '1xn' any 1xn numeric vector
% '1x3rgb' 1x3 vector v for which all(v>=0 & v<=1), e.g., a RGB code
% 'nx3rgb' nx3 numeric matrix that contains n RGB values as rows
% 'nx3dimrgb' nx3xdim numeric matrix that contains RGB values
% 'nxnx3rgb' nxnx3 numeric matrix of nxn RGB triples
% 'none' string 'none'
% 'xor' string 'xor'
% 'indexed' string 'indexed'
% 'colorstyle' strings 'red','blue','green','yellow','magenta','cyan','white'
% or 'black', or 'r','g','b','y','m','c','w','k'
% 'markerstyle' any of Matlab's marker chars '.','o','x','+','*','s','d','v',
% '^','<','>','p'or 'h'
% 'linestyle' any or Matlab's line style strings '-',':','--', or '-.'
% 'cellcolumn' a nx1 cell array
% 'topol_cell' {lattice, msize, shape}
% 'topol_cell_no_shape' {lattice, msize}
% 'string' any string (1xn array of char)
% 'chararray' any MxN char array


if nargin == 2
  str='any';
end

flag=0;
sz=size(value);
dims=ndims(value);
```

```
% isnumeric
numeric=isnumeric(value);
character=ischar(value);

% main loop: go through all types in arg. 'valid'
for i=1:length(valid),
  if isnumeric(valid{i}), % numeric size for double matrix
    if numeric & length(valid{i}) == dims,
      flag(i)=all(sz == valid{i});
    else
      flag(i)=0; % not numeric or wrong dimension
    end
  else
    msg=''; % for a error message inside try
    try
      switch valid{i}

% scalar
       case '1x1'
flag(i)=numeric & dims == 2 & sz(1)==1 & sz(2) ==1;

% 1x2 numeric vector
       case '1x2'
flag(i)=numeric & dims == 2 & sz(1)==1 & sz(2) == 2;

% 1xn numeric vector
       case '1xn'
flag(i)=numeric & dims == 2 & sz(1) == 1;

% any numeric matrix
       case 'nxm'
flag(i)=numeric & dims == 2;

% nx3 numeric matrix
       case 'nx3'
flag(i)=numeric & dims == 2 & sz(2) == 3;

% nx2 numeric matrix
       case 'nx2'
flag(i)=numeric & dims == 2 & sz(2) == 2;

% nx1 numeric vector
       case 'nx1'
flag(i)=numeric & dims == 2 & sz(2) == 1;

% nx1xm numric matrix
       case 'nx1xm'
flag(i)=numeric & dims == 3 & sz(2) == 1;

% nx3 matrix of RGB triples
       case 'nx3rgb'
flag(i)=numeric & dims == 2 & sz(2) == 3 & in0_1(value);

% RGB triple (ColorSpec vector)
       case '1x3rgb'
flag(i) = numeric & dims == 2 & sz(1)==1 & sz(2) == 3 & in0_1(value);

% any square matrix
       case 'nxn'
flag(i)=numeric & dims == 2 & sz(1) == sz(2);

% nx3xdim array of nxdim RGB triples
       case 'nx3xdimrgb'
flag(i)=numeric & dims == 3 & sz(2) == 3 & in0_1(value);

% nxnx3 array of nxn RGB triples
       case 'nxnx3rgb'
flag(i)= numeric & dims == 3 & sz(1) == sz(2) & sz(3) == 3 ...
& in0_1(value);

% nxn matrix of values between [0,1]
       case 'nxn[0,1]'

flag(i)=numeric & dims == 2 & sz(1) == sz(2) & in0_1(value);

% string 'indexed'
       case 'indexed'
flag(i) = ischar(value) & strcmp(value,'indexed');

% string 'none'
       case 'none'
flag(i) = character & strcmp(value,'none');

% string 'xor'
       case 'xor'
flag(i) = character & strcmp(value,'xor');
```

```
% any string (1xn char array)
       case 'string'
flag(i) = character & dims == 2 & sz(1)<=1;

% any char array
       case 'chararray'
flag(i) = character & dims == 2 & sz(1)>0;

% ColorSpec string
       case 'colorstyle'
flag(i)=(character & sz(1) == 1 & sz(2) == 1 & ...
any(ismember('ymcrgbwk',value))) | ...
(ischar(value) & any(strcmp(value,{'none','yellow','magenta',...
'cyan','red','green','blue','white','black'})));
% any valid Matlab's Marker
case 'markerstyle'
flag(i)=character & sz(1) == 1 & sz(2) == 1 & ...
any(ismember('.ox+*sdv^<>ph',value));
% any valid Matlab's LineStyle
case 'linestyle'
str=strrep(strrep(strrep(value,'z','1'),'--','z'),'-.','z');
flag(i)=character & any(ismember(str,'z-:')) & sz(1)==1 & (sz(2)==1 | sz(2)==2);

% any struct
       case 'struct'
flag(i)=isstruct(value);

% nx1 cell array of strings
       case 'cellcolumn_of_char'
flag(i)=iscell(value) & dims == 2 & sz(2)==1;
try, char(value); catch, flag(i)=0; end

% mxn cell array of strings
       case '2Dcellarray_of_char'
flag(i)=iscell(value) & dims == 2;
try, char(cat(2,value{:})); catch, flag(i)=0; end

% valid {lattice, msize}
       case 'topol_cell_no_shape'
flag(i)=1;
if ~iscell(value) | length(size(value)) ~= 2 | size(value,2)~=2
flag(i)=0;
else
if vis_valuetype(value{1},{'string'}),
switch value{1}
case { 'hexa','rect'}
;
otherwise
flag(i)=0;
end
end
if ~vis_valuetype(value{2},{'1xn'}),
flag(i)=0;
end
end

% valid {lattice, msize, shape}
       case 'topol_cell'
flag(i)=1;
if ~iscell(value) | length(size(value)) ~= 2 | size(value,2) ~= 3,
flag(i)=0;
else
if vis_valuetype(value{1},{'string'}),
switch value{1}
case { 'hexa','rect'}
;
otherwise
flag(i)=0;
end
end
if ~vis_valuetype(value{2},{'1xn'})
flag(i)=0;
end
if ~vis_valuetype(value{3},{'string'})
flag(i)=0;
else
switch value{3}
case { 'sheet','cyl', 'toroid'}
;
otherwise
flag(i)=0;
end
end
end
       otherwise
msg='Unknown valuetype!';
      end
```

```
    catch
      % error during type check is due to wrong type of value:
      % lets set flag(i) to 0
      flag(i)=0;
    end
    % Unknown indetifier?
    error(msg);
  end
  % set flag according to 3rd parameter (all ~ AND, any ~ OR)
  if strcmp(str,'all');
    flag=all(flag);
  else
    flag=any(flag);
  end
end


function f=in0_1(value)

f=all(value(:) >= 0 & value(:)<=1);
```

D3. Matlab code for ARFIMA method.

```
%% ARFIMA
clc
clear
close all

[data] = xlsread('data LND_ret.xlsx');

%%%%%%%%%%%%%%%%%%%%%%%
datalog = log(data);
number = 30;
n=length(data);
n=n-701;
datalog = datalog(repmat(1:n+1-number,number,1) + repmat([0:number-1]',1,n+1-
number));
datalogdiff = diff(datalog);
RV = sqrt(sum(datalogdiff.^2))';

predictValue=zeros(25,length(RV)); % Note 25
%% ARFIMA
for i=1:5
    for j=1:5
        predictValue((i-1)*5+j,:)=arfimaPredict(RV,i,j);
    end
end

%%%%%%%%%%%
q=zeros(1,25);
error=zeros(1,25);
FQPredict=zeros(1,length(RV));
FQPredict(1:6)=RV(1:6);
for i=7:length(RV)
    i
    for j=1:25
        error(i)=RV(i-1)-predictValue(j,i-1);
        q(j) =exp(-1000*error(j));
    end

    qsum = sum(q);
    q=q/qsum;

    xpart=zeros(1,25);
    for k = 1 :25
        u = rand; % uniform random number between 0 and 1
        qtempsum = 0;
        for j = 1 : 25
            qtempsum = qtempsum + q(j);
            if qtempsum >= u

                xpart(k) =predictValue(j,i);
                break;
            end
        end
    end

    FQPredict(i)=mean(xpart);

end
figure;
plot(FQPredict,'r')
hold on
plot(RV)
xlabel('Time in Dates');
```

```
ylabel('Values');
title('Forecasts VS Real values');
legend('forecast with ARFIMA Method','Real Time Series');
grid on;

figure;
plot(error,'r-','linewidth',1);
xlabel('Time in Dates');
ylabel('Values');
title('Error with ARFIMA method');
grid on;
NMSE=var(error)/var(data);

Arfima function
% implements the full maximum likelihood estimation of the ARFIMA(p,d,q) model
function [RV,Errorfit] = arfima(r,m)
% ARFIMA
% RV:
% Errorfit:

data = data(:);
%x = diff(data(:));

%%%%%%%%%%%%%%%%
datalog = log(data);
number = 30;
n=length(data);
datalog = datalog(repmat(1:n+1-number,number,1) + repmat([0:number-1]',1,n+1-
number));
datalogdiff = diff(datalog);
RV = sqrt(sum(datalogdiff.^2))';

x = RV;
n = length(x); %%%%%%%%%%%%%
d = estimate_hurst_exponent(x)-1; %%%%%%%%%%

%%%%%%%%%%%%%
k = 0:n-1;
g = zeros(1,n);
%fractional
for i = 2:n
    g(i) = prod(1-(1+d)./(1:k(i)));
end
g(1) = 1;
% g = gamma(k-d)./(gamma(k+1).*gamma(-d));
G = zeros(n);
for i = 1:n
    G(i,i:end) = g(1:n-i+1);
end
W = x'*G;

rm = max(r,m)+1;
%%%%%%%%%%%%%%%%%%
spec = garchset('R',r,'M',m,'Display','off');
%%%%%%%%%%%%%%%%%%%%%
coeffw = garchfit(spec,W);

%%%%%%%%%%%%
period = rm:n;
wperioddata = W(repmat(1:n+1-rm,rm,1) + repmat([0:rm-1]',1,n+1-rm));
[sigmaForecast,w_Forecast] = garchpred(coeffw,wperioddata,1); %

x_Forecast = [W(1:rm-1) w_Forecast]*inv(G); %
x_Forecast = x_Forecast(rm:end);

%%%%%%%%%%%%%%%%
Errorfit = sum((x(period)-x_Forecast').^2);
%%%%%%%%%%%%%
plot(period,x(period),'c','linewidth',3)
hold on
plot(period,x_Forecast,'k')
legend('ت وصµضµ','²¤شâضµ')


% datalog = log(data);
% number = 30;
% datalog = datalog(repmat(1:n+1-number,number,1) + repmat([0:number-1]',1,n+1-
number));
% datalogdiff = diff(datalog);
% RV = sqrt(sum(datalogdiff.^2))';

%----------------------------------
%calculate predictions for the actual dependent variable
function hurst = estimate_hurst_exponent(data0)

data = data0;          % make a local copy
npoints = numel(data0);
```

```
yvals = zeros(1,npoints);
xvals = zeros(1,npoints);
data2 = zeros(1,npoints);

index = 0;
binsize = 1;

while npoints>4
    y = std(data);
    index = index+1;
    xvals(index) = binsize;
    yvals(index) = binsize*y;

    npoints = fix(npoints/2);
    binsize = binsize*2;
    % average adjacent points in pairs
    ipoints = 1:npoints;
    data2(ipoints) = (data(2*ipoints)+data((2*ipoints)-1))*0.5;
    data=data2(1:npoints);
end

xvals=xvals(1:index);
yvals=yvals(1:index);

logx=log(xvals);
logy=log(yvals);

p2=polyfit(logx,logy,1);
% Hurst exponent is the slope of the linear fit of log-log plot
hurst=p2(1);

arfimapridect function
function preidictValue=arfimaPredict(data,r,m)
%%%%%%%%%

n = length(data); %
d = estimate_hurst_exponent(data)-1; %

%%%%%%%%%%%%%%%%
k = 0:n-1;
g = zeros(1,n);
for i = 2:n
    g(i) = prod(1-(1+d)./(1:k(i)));
end
g(1) = 1;
% g = gamma(k-d)./(gamma(k+1).*gamma(-d)); the gamma (generalized factorial)
G = zeros(n);
for i = 1:n
    G(i,i:end) = g(1:n-i+1);
end
W = data'*G;

rm = max(r,m)+1;
%%%%%%%%%%%%%
spec = garchset('R',r,'M',m,'Display','off');
%%%%%%%%%%%%%%
coeffw = garchfit(spec,W);

%%%%%%%%%%%%%%%%%%%%%%
wperioddata = W(repmat(1:n+1-rm,rm,1) + repmat([0:rm-1]',1,n+1-rm));
[sigmaForecast,w_Forecast] = garchpred(coeffw,wperioddata,1); %%%%

preidictValue = [W(1:rm-1) w_Forecast]*inv(G); %

arma function

function [RV,Errorfit] = arma(r,m)
% ARMA
% RV:
% Errorfit:

% [data] = xlsread('data LND_ret_2.xlsx');
x = data(:);
n = length(x); %
rm = max(r,m)+1;

%%%%%%%%%%%%
%GARCHSET is used to set parameter values in ARMAX/GARCH model
%specification structures.
spec = garchset('R',r,'M',m,'Display','off');
%%%%%%%%%%%%
%GARCHFIT given an observed univariate time series,GARCHFIT(Spec,y,...) fits
%y with the conditional mean and variance model given by the ARMAX/GARCH
specification structure Spec.
coeffx = garchfit(spec,x);
```

```
%%%%%%%%%%%%%%%%%%
period = rm:n;
xperioddata = x(repmat(1:n+1-rm,rm,1) + repmat([0:rm-1]',1,n+1-rm));
[sigmaForecast,x_Forecast] = garchpred(coeffx,xperioddata,1); %
%Garchpred given a specification of the conditional mean and variance model for
multiple paths of a univariate time series,
%%%%%%%%%%%%%%%%%%%%%%%%%
Errorfit = sum((x(period)-x_Forecast').^2);
%%%%%%%%%%%%%
plot(period,x(period),'c','linewidth',3)
hold on
plot(period,x_Forecast,'k')
legend('صوتµضµ','²¤شâضµ')

%%%%%%%%%%%%%%%%%%%%%%%
xlog = log(x);
number = 30;
xlog = xlog(repmat(1:n+1-number,number,1) + repmat([0:number-1]',1,n+1-number));
xlogdiff = diff(xlog);
RV = sqrt(sum(xlogdiff.^2))';
```

D4. Matlab code for decision-making procedure.

```
clc;
clear all;
close all;
%% %% Read Results
p1=[xlsread('rbf_res.xlsx')]';
p2=[xlsread('nncor_res.xlsx')];
p3=[xlsread('arima_res.xlsx')];
p4=[xlsread('nnabs_res.xlsx')];

p0=[p1(1:730,1)  p2(1:730,1)  p3(1:730,1) p4(1:730,1)];
p00=[xlsread('data LND_ret.xlsx')];
pr=[p00(1:730,1)];
p00=[p00(1:730,1) p00(1:730,1) p00(1:730,1) p00(1:730,1)];

%%%%%%%%%%%% error part %%%%%%%%%%%%%%%%%%%%%%%
er1=[xlsread('error_rbf.xlsx')];
er2=[xlsread('error_cor.xlsx')];
er3=[xlsread('error_ari.xlsx')];
er4=[xlsread('error_abs.xlsx')];
ERR=[er1(1:730,1)  er2(1:730,1)  er3(1:730,1) er4(1:730,1)];

Emin=min(ERR')';
%find index of Emin
% for j=1:730
%     idx(j,1)=find(ERR==Emin(j,1));
%     if idx(j,1)<=730
%         idx(j,1)=1;
%     elseif (idx(j,1)>730)&&(idx(j,1)<=2*730)
%         idx(j,1)=2;
%     elseif (idx(j,1)>2*730)&&(idx(j,1)<=3*730)
%         idx(j,1)=3;
%     else
%         idx(j,1)=4;
%     end
% end


%% %%%%%%Find Targets
t1=zeros(730,1);
t2=t1;
t3=t1;
t4=t1;
p=p0*1000;
p=fix(p);
p=fix(p/10);
for i=1:729
    if p(i,1)==p(i+1,1)
        t1(i,1)=1;
    end
    if p(i,2)==p(i+1,2)
        t2(i,1)=2;
    end
    if p(i,3)==p(i+1,3)
        t3(i,1)=3;
    end
    if p(i,4)==p(i+1,4)
        t4(i,1)=4;
    end
end
%% %%%%%%Normalization
```

```
    mean_p0=mean(p0(:)); % Count Mean value from Data Input
    std_p0=std(double(p0(:)));% Count value of standard deviation from Data Input
    pa=(p0-mean_p0)./std_p0; % Count value Result of Normalization process from Data
Input

%% %%%%%%Clustering
p1=[pa(t1==1,1)];
p2=[pa(t2==2,1)];
p3=[pa(t3==3,1)];
p4=[pa(t4==4,1)];
p=[p1(1:580,1)  p2(1:580,1)  p3(1:580,1) p4(1:580,1) ];

K=5;  % number of clusters
opts = statset('MaxIter', 10000, 'Display', 'final');
[IDXC, cent] = kmeans(pa, K,'options',opts, 'replicates',10);

%% %%%%%% Plot Data+Clusters
figure, hold on
[h]=scatter(pa(:,3),pa(:,4), 3, IDXC,'filled');
scatter(cent(:,3),cent(:,4), 100, (1:K)', 'filled');
xlabel('Feature X');
ylabel('Feature Y');
title('K=means Clustering');
grid on;

figure;
plot(Emin,'r-','linewidth',1);
xlabel('Time in Dates');
ylabel('Values');
title('minimum Error');
grid on;

% %% %%%%%% Sequared Distance
for c=1:730
  for i=1:5
    r1_rc(c,i)=[(pa(c,1)-cent(i,3))^2+(pa(c,1)-cent(i,4))^2]...
              +[(pa(c,2)-cent(i,3))^2+(pa(c,2)-cent(i,4))^2]...
              +[(pa(c,3)-cent(i,3))^2+(pa(c,3)-cent(i,4))^2]...
              +[(pa(c,4)-cent(i,3))^2+(pa(c,4)-cent(i,4))^2];
  end
  cr(c)=find(min(min(r1_rc))==min(r1_rc));
end

% %% %%%%%%where is distance cluster minimum from
er_cr=[ERR ; cr]';

N1=mean(er_cr(find(er_cr(:,5)==1),1:4));
   m_N1=find(N1==min(N1));
N2=mean(er_cr(find(er_cr(:,5)==2),1:4));
   m_N2=find(N2==min(N2));
N3=mean(er_cr(find(er_cr(:,5)==3),1:4));
   m_N3=find(N3==min(N3));
N4=mean(er_cr(find(er_cr(:,5)==4),1:4));
   m_N4=find(N4==min(N4));
N5=mean(er_cr(find(er_cr(:,5)==5),1:4));
   m_N5=find(N5==min(N5));
%after runnibng the program many times we will get the following
N1=[xlsread('n1.xlsx')];
    m_N1=find(N1==min(N1));
N2=[xlsread('n2.xlsx')];
    m_N2=find(N2==min(N2));
N3=[xlsread('n3.xlsx')];
    m_N3=find(N3==min(N3));
N4=[xlsread('n4.xlsx')];
    m_N4=find(N4==min(N4));
N5=[xlsread('n5.xlsx')];
    m_N5=find(N5==min(N5));
w_mthd=[m_N1 m_N2 m_N3 m_N4 m_N5];

% %%%%%%%%% Winner Method has the min. value inside the cluster
m1=0;m2=0;m3=0;m4=0;
for i=1:length(w_mthd)
    if w_mthd(i)==1
        m1=m1+1;
    elseif w_mthd(i)==2
        m2=m2+1;
    elseif w_mthd(i)==3
        m3=m3+1;
        else
        m4=m4+1;
    end
end
m=[m1 m2 m3 m4];
s=sort(m,'descend');
g1=find(m==s(1));
g2=find(m==s(2));
```

```
g3=find(m==s(3));
g4=find(m==s(4));

msgbox(['Method  ' num2str(g1) '  is the winner; ' ...
        'Method  ' num2str(g2) '  is the second; ' ...
        'The Methods  ' num2str(g3) '  in third place ;']);

NMSE=var(Emin);
```