Iraqi Journal for Computer Science and Mathematics

Journal Homepage: http://journal.esj.edu.iq/index.php/IJCM
e-ISSN: 2788-7421 p-ISSN: 2958-0544

Comparison of non-linear time series models (Beta-t-EGARCH and NARMAX models) with Radial Basis Function Neural Network using Real Data

Hiba H. Abdullah[1], Nihad S. Khalaf [2], Nooruldeen A. Noori[3]

[1]Department of Mathematics, College of Education for Women, Tikrit University
[2]Department of Mathematics, College of Education for Women, Tikrit University
[3]Mathematic, Anbar Education Directorate

* Corresponding Author: Hiba H. Abdullah

DOI: https:// doi.org/10.52866/ijcsm.2024.05.03.003
Received January 2024; Accepted March 2024; Available online June 2024

ABSTRACT: This paper presents a comparison of three different non-linear time series modelling approaches: NARMAX (Non-linear Autoregressive Moving Average with Exogenous Inputs), Beta-t-EGARCH (Beta t Exponential Generalized Autoregressive Conditional Heteroscedasticity), and Radial Basis Function Neural Networks (RBFNN) applied to weekly stock market index data.

We will explain three types of models and compare their compositions and structures. Then, we will show which model gives better predictions. To study series data, the comparison involved analysing the structure of the model and its errors in various time series models and summarising their findings. We divide the data into two parts: training data to structure the time series and testing. The training data tests the model's predictions. Then, we can analyse the model with the errors and the best deterrence predictions. After selecting the NARMAX and Beta-t-EGARCH models, we test them with specific criteria. The best choice is finding the model with the lowest average errors.

For this study, we analysed the weekly average closing of the Aramco 2222 index from 15 December 2019 to 16 July 2023 and made 187 observations.

Keywords: Non-linear system, identification, Beta-t-EGARCH, Markov switching, NARMAX, Radial Basis Function, Neural Network, MSE

1. Introduction

Non-linear systems are found in the real world, and linear models cannot describe their complexity. Non-linear time series modelling is challenging in many practical applications. Identifying the best models for noisy or volatile data is difficult. It is also hard to determine the numerous unknown parameters in the model. To understand how the system works, we use the input-output model. It helps describe the dynamics and link input and output information. This lets us create time series models better to describe the data.

We begin with a model called NARMAX, which was introduced by Billings in 1981 [1]. Leontaritis and Billings extensively studied and improved this model in 1985 [2]. Since then, researchers have conducted numerous studies. In 2009, Rahrooh and Scot conducted a study on identifying non-linear systems. A broad range of non-linear systems may be represented uniformly by the NARMAX model. In 2019 [3], Billings and Hua-Liang showed that this model is better than the linear model. It can estimate parameters and make accurate predictions. In 2019, Retes and Luis proposed the RaMSS method for NARMAX model structure selection [4]. Yu and Miaolei also studied NARMAX models the same year [5]. The NARMAX model is a machine learning approach that is sparse, interpretable, and transparent [6].

Harvey and Tirthankar (2008) introduced the beta-t-EGARCH model [7], a type of dynamic point model. Nelson's (1991) exponential GARCH model is expanded. The conditional variance is estimated using the Beta-t distribution. The conditional variance is logarithmically transformed [8]. Researchers presented several studies on this model. For instance, Blazsek and Villatoro asked if Beta-t-EGARCH (1, 1) is better than GARCH (1, 1) [9]. Blazsek and Mendoza compared ARMA-GARCH and QARMA-Beta-t-EGARCH [10]. Liao et al. worked on improving the MS-Beta-t-EGARCH method, a statistical technique [11]. Ayala et al. focused on quasi-autoregressive QAR and Beta-t-EGARCH [12].

Radial Basis Function Neural Networks are a popular supervised learning algorithm. This network has been used in healthcare, data processing, and pattern classification. Powell introduced it in 1985 to solve interpolation problems in data centres [13]. Gholam et al. conducted a study to explore various methods to determine the centres, widths, and synaptic weights of RBFNN [14]. Que and Belkin analyse two common regularisation procedures. [15]. Bugshan et al. also proposed a model that uses Gaussian RBF criteria for specific training [16].

2. Non-linear Modelling:

Mathematical modelling is a process that includes several stages and cycles. To start, estimate the math model in a class of models using input-output data. However, before estimating model parameters, it is necessary to know the architecture of the model. In non-linear math modelling, it is important to structure the model correctly. This helps define the parameters accurately. Non-linear modelling is divided into four basic stages: determining the class of models, investigating the structure of the model, estimating the parameters, and validating the model system by comparing the model's predictions to the behaviour of the actual system [3].

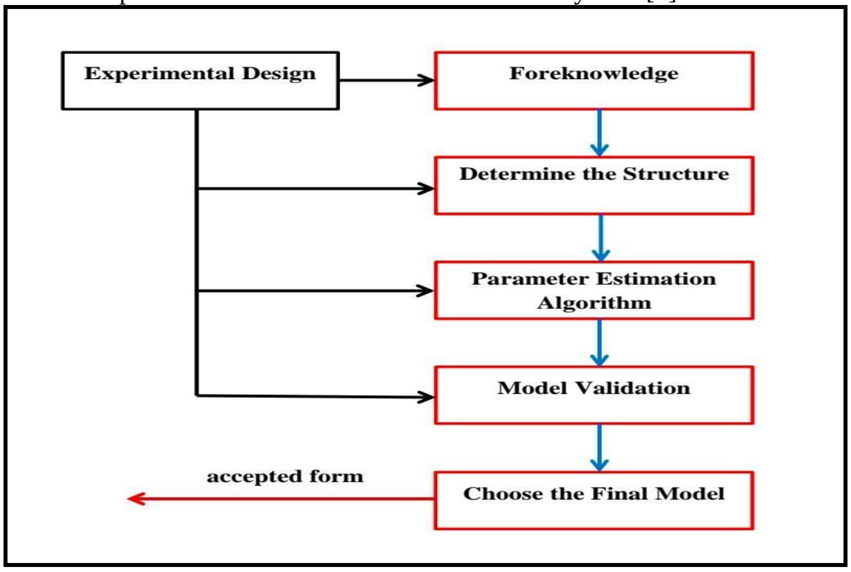


Figure.1. Non-linear time series modelling process[3]

2.1. NARMAX Model:

NARMAX is a flexible, non-linear model that can represent a wide range of dynamical systems. Parameter estimation uses the least squares approach. for Model structure selection uses error reduction ratio criteria.

The NARMAX model is a unified representation of a broad class of non-linear systems. If the system is linear, it can be represented by the linear difference equation form, which can be limited to the equation:

$$x_t = \sum_{i=1}^{n} \alpha_i x_{t-i} + \sum_{j=1}^{m} \beta_j k_{t-d-j+1} \qquad \ldots(1)$$

where n and m are order free from white noise the order of the output x_t, and k_t respectively, while α_i and β_j the parameters for the system, and d is input delay.

In a non-linear system, we can use the NARX model or develop it further. This helps us derive the NARMAX model, which can become the formula.

$$\begin{aligned} x_t = F[&x_{t-1}, \ldots, x_{t-n_x}, x_{t-i}; k_{t-d}, \ldots, k_{t-d-m} \\ &, \varepsilon_{t-1}, \ldots, \varepsilon_{t-n_\varepsilon}] + \varepsilon_t \end{aligned} \qquad \ldots(2)$$

where F is the non-linear function, x_t, k_t, and ε_t are output, input signal, and white noise, respectively, n_x, m, and n_ε the maximum lags [2].

The NARMAX model is better than other models for analysing data. It can handle noise signals that are related or have different colours. It can represent the signals in a linear or non-linear way. By expanding equation 2, we get:

$$x_t = \begin{cases} x_{t-\tau} & 1 \le \tau \le n_x \\ k_{t-\tau+n_x} & n_x + 1 \le \tau \le n_x + m \\ \epsilon_{t-\tau+n_x+m} & n_x + m + 1 \le \tau \le n_x + n_\varepsilon + m \end{cases} \qquad \ldots(3)$$

In equation (3), it contains white noise. This means that it cannot be directly measured or observed in the model selection procedure. To estimate the noise signal, the residual sequence of the model is often used instead, which represents the difference between the actual output and the expected output of the model.

$$\begin{aligned} \delta_t &= x_t - \hat{x}_t \\ \delta_t &= x_t - \hat{F}[x_{t-1}, \ldots, x_{t-n_x}, x_{t-i}; k_{t-d}, \ldots, k_{t-d-m}, \delta_{t-1}, \ldots, \delta_{t-n_\varepsilon}] \end{aligned} \qquad \ldots(4)$$

where $\hat{F}$ is the estimator of F.

Thus, the ε_t in equation (3) can be replaced by the δ_t given in equation (4). The NARMAX model is a machine learning approach designed to be sparse, interpretable, and transparent. It uses a small number of variables, is easy to understand and explain, and provides insights into the underlying data [3],[4].

2.1.1. Parameter Estimation of the NARMAX Model:

If the model structure is known and the model is linear in the parameters, a least squares method can be used to estimate the parameters of the NARX part first and then use the residual as the ε_t data to estimate the full NARMAX model. Where ε_t is updated with the new residual, and the estimation process is repeated until the stopping criterion is reached.

It is done in detail by following the following steps: We rewrite equation (2) by setting $\upsilon_1 = x_{t-1}, \upsilon = x_{t-n_x} = k_{t-d}$, and $\upsilon_s = k_{t-d-m+1}, s = n_x + m$, then we get:

$$x_t = F[\upsilon_1, \upsilon_2, \dots, \upsilon_s] \qquad \dots(5)$$

$$x_t = \sum_{i=1}^{s} \mu_i \upsilon_i + \sum_{j=1}^{s} \mu_{ij} \upsilon_i + \upsilon_j + \dots + \sum_{i=1}^{s} \dots \sum_{i=1}^{s} \mu_i \dots i\upsilon_i \dots \upsilon_i \qquad \dots(6)$$

where μ is a non-linear coefficient.

We assume that the output signal x_t generates zero mean noise ε_t, which indicates that the measured output signal is

$$y_t = x_t + \varepsilon_t \qquad \dots(7)$$

Substituting equation (7) into equation (6), we get:

$$y_t = \sum_{i=1}^{n_y} \mu_i (y_{t-i} - \varepsilon_{t-i}) + \sum_{j=1}^{m} \mu_{n_y} i(k_{t-d-i+1})$$
$$+ \sum_{i=1}^{n_y} \sum_{j=1}^{m} \mu_{ij} (y_{t-i} - \varepsilon_{t-i})(y_{t-j} - \varepsilon_{t-j})$$
$$+ \sum_{i=1}^{n_y} \mu_{i,n_y+j} (y_{t-i} - \varepsilon_{t-i})\, k_{t-d-j+1}$$
$$+ \sum_{i=1}^{n_y} \sum_{j=1}^{i} \mu_{n_y+i,n_y+j} k_{t-d-i+1} . k_{t-d-j+1} + \dots + h + \varepsilon_t \qquad \dots(8)$$

where h is a higher-order term.

The model in equation (8) is 'linear in parameter', but it introduces cross-product terms between the noise and the input and output signals of the process, making it a non-linear model. The solution of equation (8) is a very complex process that is impossible to solve directly, so it must be reformulated in the predictive errors model, which can be expressed as equation (9):

$$y_t = F[y_{t-1}, \dots, y_{t-n_y}; k_{t-d}, \dots, k_{t-d-m+1};$$
$$\zeta_{t-1}, \dots, \zeta_{t-n_\zeta}] + \zeta_t \qquad \dots(9)$$

where $\zeta_t = y_t - \hat{y}_t$, $\hat{y}_t = \hat{F}[y_{t-1}, \dots, k_{t-d}; \zeta_{t-1}, \dots, \zeta_{t-n_\zeta}]$, and $E[\zeta_t / y_{t-1}, \dots, k_{t-d}, \dots] = 0$, n_ζ is the number of added noise terms.

Equation (9) can be expanded and regrouped to produce equation (10), which separates the unknown parameters.

$$Y_t = \mathrm{H}^{yk} \left[y_{t-1}, \dots, y_{t-n_y}, k_{t-d}, \dots, k_{t-d-m+1} \right] +$$
$$\mathrm{H}^{yk\zeta} [y_{t-1}, \dots, y_{t-n_y}, k_{t-d}, \dots, k_{t-d-m+1}, \zeta_{t-1}, \dots, \zeta_{t-n_\zeta}]$$
$$+\mathrm{H}^{\zeta} [\zeta_{t-1}, \dots, \zeta_{t-n_\zeta} + \zeta_t] \qquad \dots(10)$$

where $\mathrm{H}^{yk}, \mathrm{H}^{yk\zeta}$, and H^{ζ} are functions in terms of y and k, in terms of y, k, ζ and in terms of ζ, respectively.

How the unknown parameters relate to the coefficients and polynomial terms can be illustrated in the following equation:

$$y_t = \Psi_t^T \mu_{t-1} + \zeta_t$$
$$= \left[\Psi_{t\ yk}^T \Psi_{t\ yk\zeta}^T \Psi_{t\ \zeta}^T\right]\begin{bmatrix}\mu_{t-1_{yk}}\\ \mu_{t-1_{yk\zeta}}\\ \mu_{t-1_{\zeta}}\end{bmatrix} + \zeta_t \qquad \ldots(11)$$

Then $H^{yk} = \Psi_{t\ yk}^T \mu_{t-1_{yk}}$, $H^{yk\zeta} = \Psi_{t\ yk\zeta}^T \mu_{t-1_{yk\zeta}}$, and $H^{\zeta} = \Psi_{t\ \zeta}^T \mu_{t-1_{\zeta}}$. From which we get $\varepsilon_t = \Psi_{t\ yk\zeta}^T \mu_{t-1_{yk\zeta}} + \Psi_{t\ \zeta}^T \mu_{t-1_{\zeta}} + \zeta_t$, that gives:

$$y_t = \Psi_{t\ yk}^T \mu_{t-1_{yk}} + \zeta_t \qquad \ldots(12)$$

Equation (12) shows how the model can be expressed as the sum of the expected output and the error term, which is highly correlated and can lead to biased parameter estimates. The algorithm based on least squares is a good choice for parameter estimation[3],[17].

2.1.2. Structure Detection of the NARMAX Model:

The problem of model structure selection is a critical step in the determination of the non-linear system. In this problem, the goal is to select a subset of regression factors from a set of R from $n \times m$ regressions to construct the final model. Normalise it with respect to the output variance. The error reduction ratio (ERR) generated by including the ith regression factor in the model can be calculated using a specific formula. The formula takes into account the variance of the residuals before and after the regression coefficient is included in the model. The ERR is a useful criterion for selecting the most relevant regression factors for the final model, as it quantifies the improvement in model accuracy due to the inclusion of each regression factor. ERR is widely used in non-linear system determination and has been shown to be effective in selecting the most appropriate regression factors for the final model [4]. Which can be expressed by the equation:

$$ERR_{1_i} = \frac{MSPE_1(\varphi_{i-1}) - MSPE_1(\varphi_i)}{< y, y >}, i = 1,2,\ldots,m \qquad \ldots(13)$$

where $MSPE_1$ is Mean Square Production Error with one-step forward, with i-terms regressors, amount of candidate terms is m it has been examined, and φ_i represents a family of related models nesting structures. The text of equation (13) indicates and explains the numerator and denominator of the equation. The equation counter represents the decrease in variance in the residual values due to the inclusion of the i-th regression operator. In other words, it measures to what extent the inclusion of a particular regression factor reduces the error in the model predictions, while the denominator of the equation represents the data variance, i.e., it is a measure of the data variance around the mean, so that it is used to normalise the numerator and obtain a measure of the relative importance of each regression factor. And that ERR_1 is a method used to choose the model based on the prediction error of the model. It is useful because it can be represented in a compact form [5] as

$$[ERR_1]_i = \frac{\hat{g}_i^2 < \omega_i, \omega_i >}{< y, y >}, i = 1,2,\ldots,m \qquad \ldots(14)$$

where $\hat{g}_i$ is the estimation parameter, and ω_i is i-th orthogonal regression. An expansion of ERR_1 is the Error Reduction Ratio two-step forward ERR_2 presented by Alves et al. in 2012 [18]. In addition to this extension, there are other parameters, such as the Simulation Error Reduction Ratio (SRR), introduced by Piroddi and Spinelli in 2003 [19].

We discuss here the importance of understanding that there is no clear distinction between 'true' and 'false' regressions when selecting a model structure from experimental data. While some regression factors may be more useful, no regression factor can be definitively classified as true or false. Additionally, the text notes that when working with limited and possibly imprecise data, the structures of different models may become indistinguishable. Using a probabilistic approach, it is possible to account for the uncertainty inherent in the data and make more informed decisions about which model structures to choose from. Likely to be accurate.

2.1.3. Validity Test of the NARMAX Model:

Model validation checks the residual sequence for bias that could affect parameter estimation. By equation (9), there are parameters with a vector of φ of dimension n_φ, we get:

$$x_t = F[x_{t-1},\ldots,x_{t-n_x};\zeta_{t-1}(\varphi),\ldots,\zeta_{t-n_\zeta}(\varphi),\varphi] + \zeta_t(\varphi) \qquad \ldots(15)$$

where $\zeta_t(\varphi)$ is a residual model at φ.

To get estimates of φ and calculate the correlation coefficient, we reduce the loss function through a series of steps. To analyse the even and odd functions, we sum the products of Volterra grains and Gaussian noise. Then, we sum the products of Volterra kernels and Gaussian noise, suppressing the odd terms to zero. You can evaluate the second-to-fourth correlations of the extended residues as the Volterra series. This will help you detect all the possible deleted terms in the model. In this evaluation, we assume that noise is balanced, odd terms are eliminated, even terms are kept, and the third moment is zero. From this, we get the final equation:

$$\mathrm{H}_n^0 = \sum_{t_1}^{\infty} \sum_{t_2}^{\infty} \cdots \sum_{t_1}^{\infty} h_{t_1\, t_2 \ldots t_m, i\, j \ldots m}^{i\, j \ldots m, i\, j \ldots m}\ \varepsilon_{t-t_1}^{i}\ \varepsilon_{t-t_2}^{j} \cdots \varepsilon_{t-t_m}^{m}\ \gamma_{2n}^{e} \qquad \ldots(16)$$

where $h_{t_1\, t_2 \ldots t_m, i\, j \ldots m}^{i\, j \ldots m, i\, j \ldots m}$ are Volterra kernels [5].

2.2. Volatility Model (Beta-t-EGARCH):

In 1982, Engle suggested ARCH models. However, these models were not enough to model all time series. Specifically, he assumed that positive and negative sudden effects have the same impact [20]. Therefore, a generalisation of these models was developed by Bollerslev in 1986 [21]. The aim of these models is to control the models. GARCH is a mathematical model that shows the unevenness of conditional variance. It includes extreme values and is represented by the following equations:

$$y_t = \mu_t + x_t$$

$$x_t = \sigma_t\, \epsilon_t \qquad \textit{where } \epsilon_t \sim \textit{iid } N(0,1)$$

$$\sigma_t^2 = w + \sum_{i=1}^{Q} \alpha_i\, x_{t-i}^2 + \sum_{j=1}^{P} \beta_j\, \sigma_{t-j}^2 \qquad \ldots(17)$$

where w, α_i, and β_j are parameters of constant, ARCH, and GARCH, respectively, while Q, P are the numbers of late conditional differences for the variance parameters and the number of conditional error variances, respectively.

The Exponential GARCH (EGARCH) model was developed by Nelson in 1991. This model differs from the structure of the GARCH model because of past differences in variance and is expressed by the equation:

$$\log \sigma_t^2 = w + \sum_{i=1}^{Q} \alpha_i \log \sigma_{t-j}^2 + \sum_{j=1}^{P} \vartheta_j\, g(\epsilon_{t-j}) \qquad \ldots(18)$$

where $g(\epsilon_{t-j}) = \gamma_j(\epsilon_{t-j}) + \beta_j(|\epsilon_{t-j}| - \mathrm{E}|\epsilon_{t-j}|)$ [8].

The Beta-t-EGARCH model is a type of dynamic conditional point model that is used to estimate the conditional variance of a time series. This model is an extension of the EGARCH model and uses the conditional score of the beta-t distribution to estimate conditional variance, which makes it more robust against outliers. The model is constructed from two components, the conditional mean and the conditional variance, and is powered by the conditional scale parameter λ_t. The form Beta-t-EGARCH (Q, P) can be expressed as:

$$y_t = \mu_t + v_t = \mu_t + e^{\lambda_t}.\epsilon_t \qquad \ldots(19)$$

Conditional volatility is expressed with leverage effects:

$$\lambda_t = w + \sum_{i=1}^{Q} \alpha_i k_{t-i} + \sum_{j=1}^{P} \beta_j \lambda_{t-j} \qquad \ldots(20)$$

where $k_t = [(r+1)b_t] - 1, b_t = \left[\frac{\frac{v_t^2}{r.e^{2\lambda_t}}}{1+\frac{v_t^2}{r.e^{2\lambda_t}}}\right]$, α_i, and β_j are ARCH and GARCH coefficient effects, respectively, and the standard residual value, z_t, is assumed to follow a t-Student's distribution with mean 0, variance 1, and degree of freedom $r > 2$ [22].

2.2.1. Conditional Stability and Forecast of Beta-t-EGARCH:

Beta-t-EGARCH is an extension of EGARCH that models conditional variance using the beta-t distribution. The stability condition for the covariance of the Beta-t-EGARCH (Q, P) is given by [22,11] $|\sum \beta_j| < 1$.

In order to give a predictive model, we take equation (20) of order (1,1), so we get:

$$\lambda_t = w + \alpha k_{t-1} + \beta\lambda_{t-1} \qquad \ldots(21)$$

Forecasting is one step ahead:

$$\hat{\lambda}_{t+1} = w + \alpha k_t + \beta\lambda_t$$

Forecasting is the second step ahead:

$$\hat{\lambda}_{t+2} = w + \alpha k_{t+1} + \beta\hat{\lambda}_{t+1}$$

Since $E(k_t) = \lambda_t, \forall t$. We get:

$$\hat{\lambda}_{t+2} = w + (\alpha + \beta)\hat{\lambda}_{t+1}$$

Forecasting is s-step ahead:

$$\hat{\lambda}_{t+s} = w + (\alpha + \beta)\hat{\lambda}_{t+(s-1)}$$

Thus, we get a predictive composite of k future steps in the form:

$$\hat{\lambda}_{t+s} = w + \sum_{i=1}^{s-2}(\alpha + \beta)^i + (\alpha + \beta)^{s-1}\hat{\lambda}_{t+1} \qquad \ldots(22)$$

2.2.2.QARMA-Beta-t-EGARCH:

As it is known, the Beta-t-EGARCH models were designed to predict the values of conditional variances; that is, they are not sufficient to predict the values of the series data, so they were presented in addition to a combination with other models of the mean. This study uses the QARMA-Beta-t-EGARCH model, which is particularly effective in controlling. Means that the model is able to adjust its parameters based on new information, allowing it to accurately predict future returns even in the presence of outliers. This is important because outliers can significantly impact the general distribution of returns and make it difficult to accurately predict future returns. To do this, the model uses the prior values of the conditional degree of the LL function (which is a measure of the probability of observing a given return given the prior values of the return and other relevant variables) to change the location and magnitude of the returns. So, the equation of the model is in the form:

$$y_t = \mu_t + v_t = \mu_t + e^{\lambda_t}.\epsilon_t$$

$$\mu_t = \tau + \sum_{i=1}^{p} a_i\mu_{t-i} + \sum_{j=1}^{q} c_j e_{t-j} \qquad \ldots(23)$$

$$e_t = \left[1 + \frac{v_t^2}{r.e^{2\lambda_t}}\right]^{-1} v_t \qquad \ldots(24)$$

$$\lambda_t = w + \sum_{i=1}^{Q} \alpha_i k_{t-i} + \sum_{j=1}^{P} \beta_j\lambda_{t-j} \qquad \ldots(25)$$

$$k_t = \left[\frac{(r+1)v_t^2}{re^{2\lambda_t} + v_t^2}\right] - 1 \qquad \ldots(26)$$

3. Radial Basis Function Neural Networks (RBFNN):

RBFNN is a 3-layer neural network using radial basis functions in the hidden layer. It has a non-linear mapping from the input to the hidden layer and a linear mapping from the hidden to the output layer. Training involves unsupervised learning of RBF centres/widths and supervised learning of output weights [14].

To calculate the distance between a data point and the centre of the grid, there are several types of functions:

$$\phi(r) = e^{-\frac{r^2}{2\sigma^2}}, r > o \qquad \text{Gaussian} \qquad \ldots(27)$$

$$\phi(r) = \frac{1}{(r^2 + \sigma^2)^{\alpha}}, \alpha > 0 \qquad \ldots(28)$$

$$\phi(r) = (r^2 + \sigma^2)^{\beta}, 0 < \beta < 1 \qquad \ldots(29)$$

$$\phi(r) = r \qquad \text{Linear} \qquad \ldots(30)$$

$$\phi(r) = r^2 \ln r \qquad \text{thin-plate spline} \qquad \ldots(31)$$

$$\phi(r) = \frac{1}{1 + e^{\frac{r}{\sigma^2} - \vartheta}} \qquad \text{Logistic} \qquad \ldots(32)$$

where r is represents the distance between the centre c, σ and the data point, either in equations 27, 28, 29, or 32, it is used to control the smoothness of the interpolation function, ϑ in equation (32) it is a function Table bias, and β in equation (29) Often set to 0.5, the RBF becomes Hardy's multi-quadric function. Most of the time, when training the neural network equation (27) is taken [23].

3.1. Interpolation Networks:

The interpolation problem refers to the task of estimating values between known data points. A three-layer feed-forward neural network can elegantly solve this problem. The three layers in the network have distinct roles in the learning process: the input layer consists of the nodes that receive the sensor information, the second layer is the only hidden layer and applies a non-linear transformation from the input space to the hidden space, the non-linear transformation is followed by a linear transformation from the hidden layer to the output layer. A neural network can estimate values between known data points in a high-dimensional space using a combination of non-linear and linear transformations. The mathematical formulation of the interpolation problem can be summarised below.

Let $\{b_i \in \mathbb{R}^n, i = 1,2,\dots,\mathbb{N}\}$, $\mathbb{N}$ is data points, $\{d_i \in \mathbb{R}, i = 1,2,\dots,\mathbb{N}\}$

$$f\colon \mathbb{R}^n \to \mathbb{R}$$
$$f(b_i) = d_i, \forall i, i = 1,2,\dots,\mathbb{N} \qquad \dots(33)$$

RBF has a select f technique, so it becomes:

$$f(b_i) = \sum_{i=1}^{\mathbb{N}} c_i\, h(\|x - b_i\|) + \sum_{i=1}^{L} d_i p_i(x) \ ,L \le \mathbb{N} \qquad \dots(34)$$

where h is the smooth transition function defined as RBF while $\|.\|$ is the Euclidean Norm function in $\mathbb{R}^n$, $\{p_i \in \mathbb{R}, i = 1,2,\dots,m\}$ represents the basis of the algebraic polynomials in linear space $\pi_{k-1}(\mathbb{R}^n)$ for max. order $k-1$ as k given. Let h is continuous, differentiable, and strictly monotonic at $[0,\infty)$, then we get:

$$f(b_i) = \sum_{i=1}^{\mathbb{N}} c_i\, h(\|x - b_i\|) \qquad \dots(35)$$

Now that we can implement the approximation problem with a three-layer neural network. It is possible to obtain a simplified form if we specify $\mathbb{N}$ for RBF with width σ_i:

$$f(b_i) = \sum_{i=1}^{\mathbb{N}} c_i\ g\left(\frac{\|x - b_i\|}{\sigma_i}\right) \qquad \dots(36)$$

[25]

3.2. Regularisation Networks:

It is a method used to solve unsolved problems in neural networks. The problem becomes incorrect when the data is corrupted by noise, as there are infinite solutions. To choose a particular solution, prior knowledge of the function $f(x)$ is needed, such as prior smoothing information [as a measure of the 'oscillating' behaviour of the function, as the smooth function shows oscillations are few and have less energy in the high-frequency range].

Regularisation theory can be introduced as a new approach to solving these unsolved problems, which involves fixing the solution using a non-negative auxiliary function that includes a priori information about the solution.

The solution to the regularisation problem is denoted by $f(x)$, which is required to approximate the data well and to be smooth at the same time. Let $\partial[f]$ be a smoothness function where $f(x)$ must be minimised in the following equation:

$$\mathbb{H}[f] = \sum_{i=1}^{\mathbb{N}} [d_i - f(b_i)]^2 + \eta\ \partial[f] \qquad \dots(37)$$

$\eta \in \mathbb{Z}^+$, is called the regularisation parameter. In equation (37), the first part is to find the distance to the solution f, while the second part enforces softness and measures the cost associated with its deviation from smoothness. Where the regularisation coefficient η controls it. The solution to the regulation problem is given by the following equation:

$$f(x) = \frac{1}{\eta} \sum_{i=1}^{\mathbb{N}} [d_i - f(b_i)]\ G(x, b_i) \qquad \dots(38)$$

$$f(x) = \sum_{i=1}^{\mathbb{N}} w_i \; G(x, b_i) \qquad \ldots(39)$$

where b_i are expansion centres, weights $w_i = \eta.[d_i - f(b_i)]$ represent expansion coefficients, and $f(x)$ is a linear overlay of the $\mathbb{N}$ Green functions. Equation (39) is a simplified form of equation (38), which can be interpreted as a network with only one hidden layer of units, referred to as the organising network.

Since Green's functions are symmetric with $G(x, b_i) = G(b_i, x)$ only depend on $\|x - b_i\|$, if the smoothness function $\partial[f]$ is not constant both in translational and rotational terms, then equation (39) becomes of the form:

$$f(x) = \sum_{i=1}^{\mathbb{N}} w_i \; G\|x - b_i\| \qquad \ldots(40)$$

Accordingly, we obtain the following indicators:

- Equation (40) is a special form of equation (39) that represents strict interpolation, where all training data points are used to generate the interpolation function.
- The regularisation coefficient, η, can be interpreted as an indication of the sufficiency of the available data to determine the solution $f(x)$.
- If $\eta \to 0$, the solution $f(x)$ is sufficiently determined from the available examples, while $\eta = 0$ corresponds to pure interpolation.
- If $\eta \to \infty$, then the examples are unreliable and, in practice, η falls between these two limits to enable both sample data and prior information to contribute to the solution $f(x)$.
- The regularisation problem is used to mitigate the overtraining problem in neural networks, whereby the network learns all the details of input-output pairs but not their general shape by adding noise to the input data and choosing the parameter of intelligent regularisation. [24]

3.3. Data Processing in RBFNN:

In RBFNN, data processing is different from standard supervised or unsupervised learning. In a neural network, hidden units are kernel functions that create a 'base' of input patterns. The hidden unit space maps these patterns. RBFs are the functions that perform the nucleus functions in an RBFNN. They approximate the input-output relationship. In math, the receptive field is like the nucleus [24]. It describes local interaction between neurons [26-27].

The steps involved in data flow through an RBFNN during classification, which is a typical RBFNN learning algorithm [24]

1. Initialisation, where random values are chosen for the initial weights of the RBFNN. The size of the weights should be small, and the b_i centres and k_i-form matrices of $\mathbb{N}$ are chosen given the RBF.
2. Sampling, where the pattern x is randomly drawn from the input data, which represents the input to the neural network.
3. Involves the forward computation of the hidden layer activations, where the values of the hidden layer nodes are computed using the Mahalanobis distance, which is a metric standard.
4. Front computation of output layer activations, where the values of the output nodes are calculated based on the weights and values of the hidden layer nodes.
5. Refresh, where the weights of all neurons in the output layer are adjusted based on the steepest regression rule.
6. Continuation, where the algorithm continues with (2) until no significant changes in the error function are observed.

The algorithm relies on prior knowledge of the centres of the radial basis and their shape matrices [24].

3.4. Training Techniques for RBFNN:

RBFNN has a mixed learning process that involves adjusting both the linear weights of the output neurons and the non-linear activation functions of the hidden neurons. Adjusting the weight of output neurons is a linear process, while adjusting the non-linear parameters of hidden neurons is non-linear. Since the hidden layers and output layers of the RBFNN perform different tasks, we need to separate the optimisation of these layers and use different learning techniques for each. There are degrees of freedom in choosing the kernel functions of the RBFNN, which can be fixed or modified during the training phase. There are several techniques for designing an RBFNN, depending on how the functions of the radial basis centres are defined. The best-known strategies chosen for practical applications are the following:

- Method for selecting fixed centres for RBFNN: in this method, the kernel functions are fixed and the core locations are randomly selected from the training set. Each RBF is defined as a function of the distance between the input vector and the randomly selected centre.
- This process is based on a self-organising learning strategy where the locations of the nucleus function centres are adapted based on a clustering algorithm that divides the given dataset into homogeneous subsets. A k-means clustering algorithm is used to map the centres of the RBF, and the optimal number of RBF is determined empirically. The k -means algorithm continues through initialisation, sampling, similarity matching, update, and persistence.
- All free parameters of kernel functions, including centres, are adapted based on a supervised learning strategy. A kernel function is a mathematical function used to measure the similarity between two inputs in a neural network. Centres are the points in the input space around which the kernel function is evaluated. A supervised learning strategy means that the network is trained using labelled data, where the correct output is known for each input. This method results in an RBFNN in its most general form. An RBFNN is a type of neural network that uses RBF as an activation function. The error correction learning mentioned in the text refers to a kind of learning algorithm where the network weights are modified to reduce the difference between the expected output and the actual output [24].

After completing all stages of building, training, and testing the RBFNN, the following is Figure 2, showing the structure of the RBFNN.

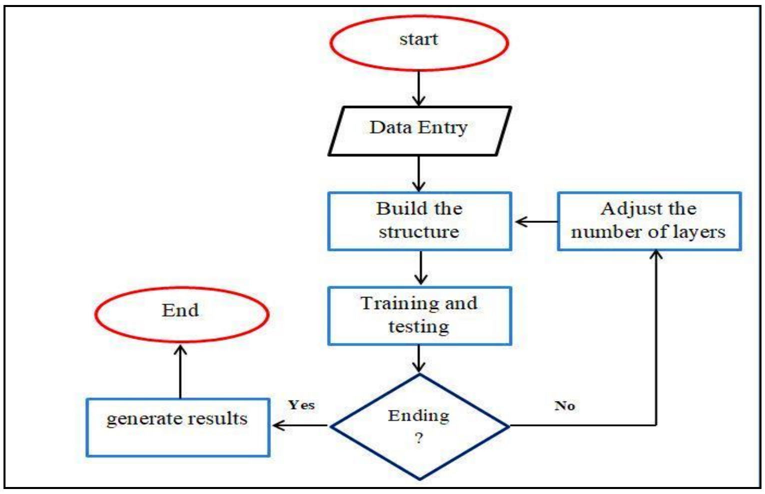


Figure.2. Structure of RBFNN [26]

4. Data analysis:

4.1.The Analysis:

First, we draw the time series, which is considered an initial step in all-time series analyses, which gives the first impression of the nature of the time series, as Figure 3 represents the weekly average of closing the trading market for the index of Aramco 2222. Obtained from www.investing.com.

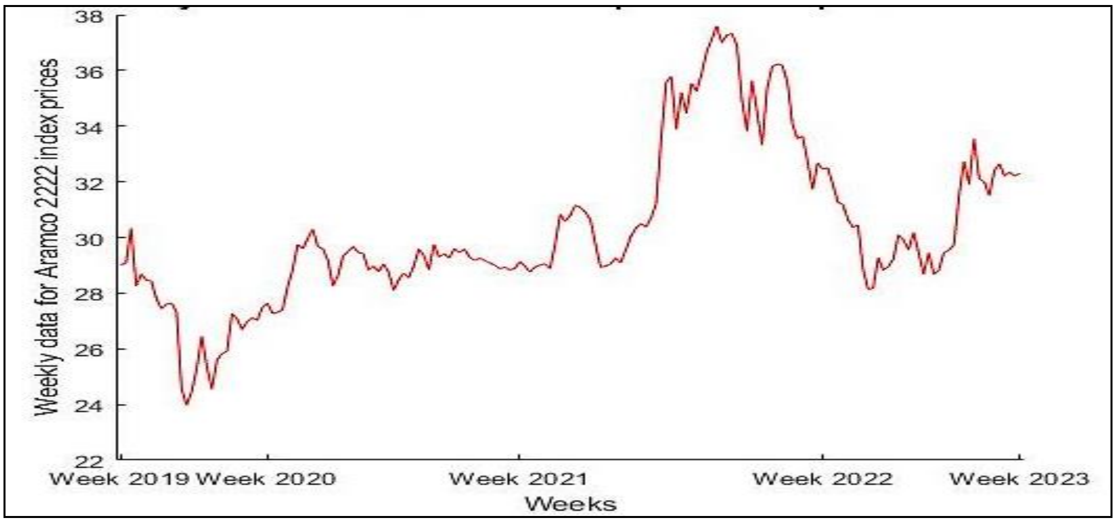


Figure.3. **plot The time series**

The time series plot is followed by the return series plot.

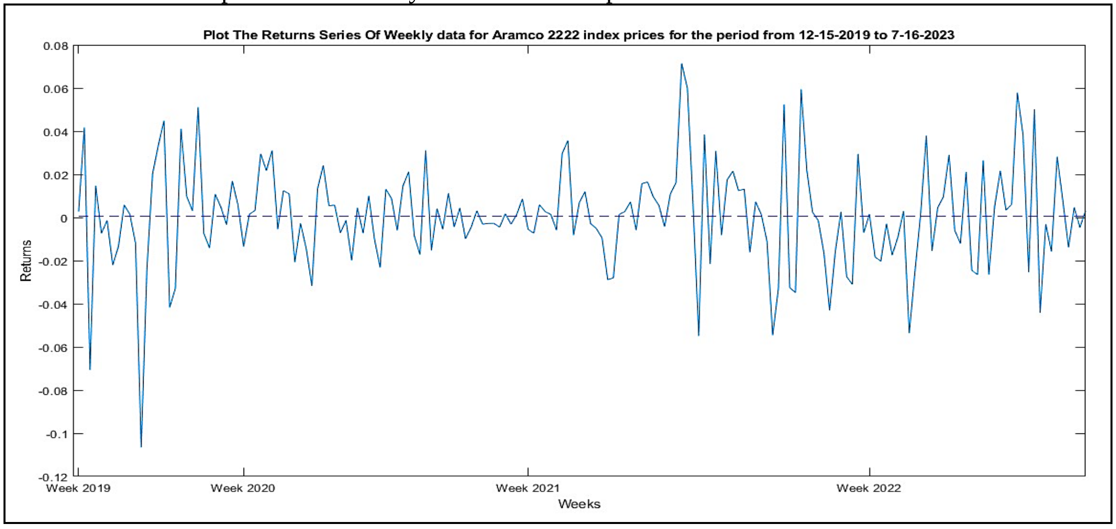


Figure.4. **The Return series plot**

4.2. NARMAX Analysis:

The modelling or analysis process in NARMAX models is very simple, as the mathematical modelling process includes a certain number of stages and cycles. From a given set of inputs and output measurement data, the importance of formulating the structure is emphasised before the model parameters are defined. modelling process

Where the initial network was configured (**Determination of the class of models**) by placing 50 hidden layers to conduct the first experimental design of the network and a training rate of 0.8, as well as setting the number of sessions during the training period to 1000 periods, a combined prediction (CP) expressed by the equation was used

$$CP = \alpha * ARIMA_E + (1-\alpha) * Y_h$$

where α is the weight of the combined prediction, $ARIMA_E$ is the ARIMA estimator, and Y_h is the non-linear prediction.

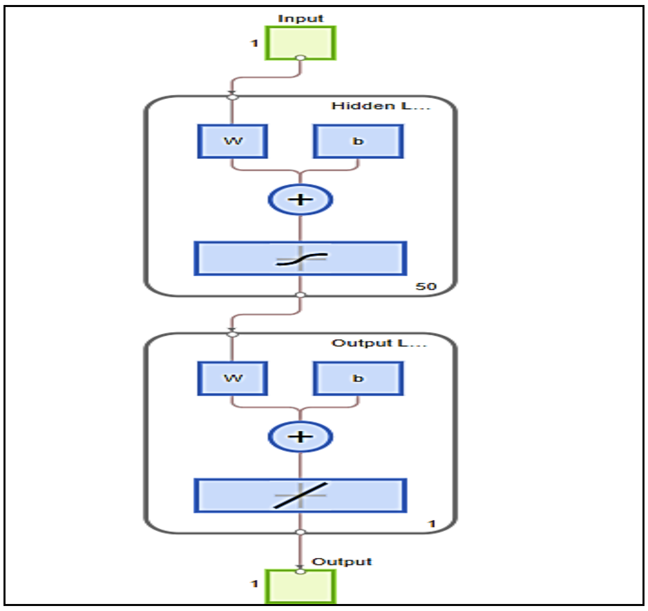


Figure.5. Adaptive neural network diagram by Matlab

The next step is the parameter estimation stage of the model, where the ranks from $n = m = 1$ to $n = 4$ and $m = 5$ were used in Equation 1 to obtain estimates of the model's parameters as well as to validate the best model. The MSE component criterion was also chosen to determine the best order.

Table.1. n and m values and MSE component

n and m values	MSE component	n and m values	MSE component
$n = 1, m = 1$	0.5891	$n = 3, m = 1$	0.6041
$n = 1, m = 2$	0.6022	$n = 3, m = 2$	0.5728
$n = 1, m = 3$	0.5941	$n = 3, m = 3$	0.5646
$n = 1, m = 4$	0.5960	$n = 3, m = 4$	0.5544
$n = 1, m = 5$	0.6091	$n = 3, m = 5$	0.5657
$n = 2, m = 1$	0.5944	$n = 4, m = 1$	0.5831
$n = 2, m = 2$	0.5530	$n = 4, m = 2$	0.5654
$n = 2, m = 3$	0.5563	$n = 4, m = 3$	0.5552
$n = 2, m = 4$	0.5595	$n = 4, m = 4$	0.5524
$n = 2, m = 5$	0.5555	$n = 4, m = 5$	0.5448

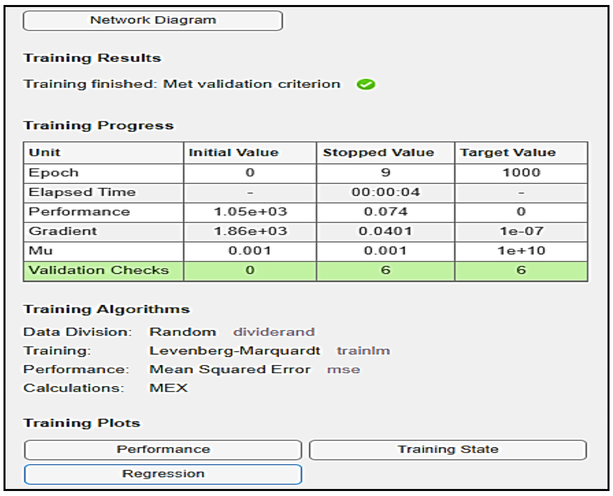
Therefore, the model $n = 4, m = 5$ is the best model because it contains the lowest value of the chosen criterion, and that the values of the estimated parameters is:

Table.2. The values of the estimated parameters

	Value	Standard Error	T-Statistic	P-Value
Constant	0.013644	0.0373	0.3658	0.71451
α_1	-0.66979	0.85374	-0.78453	0.43273
α_2	0.010036	0.50079	0.020041	0.98401
α_3	0.83327	0.38662	2.1553	0.031142
α_4	0.43424	0.56313	0.77112	0.44063
β_1	0.82222	0.85483	0.96184	0.33613
β_2	-0.014724	0.66602	-0.022108	0.98236
β_3	-1	0.46178	-2.1655	0.030346
β_4	-0.50056	0.66126	-0.75698	0.44906

β_5	0.051688	0.11116	0.46499	0.64194
var	0.45197	0.057709	7.8318	4.8095e-15

Then we say in the last step the application of the final model: below is Neural Network training.



Training Results

Training finished: Met validation criterion ✓

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	9	1000
Elapsed Time	-	00:00:04	-
Performance	1.05e+03	0.074	0
Gradient	1.86e+03	0.0401	1e-07
Mu	0.001	0.001	1e+10
Validation Checks	0	6	6

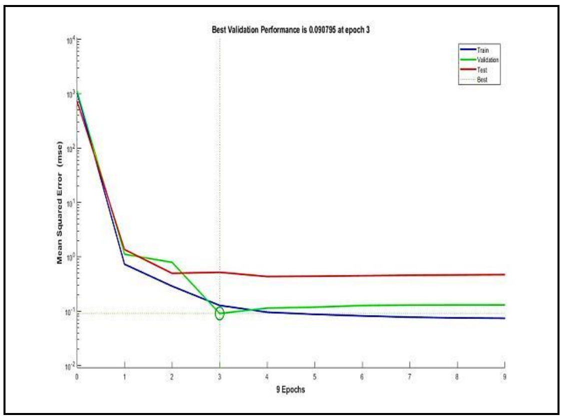
Training Algorithms

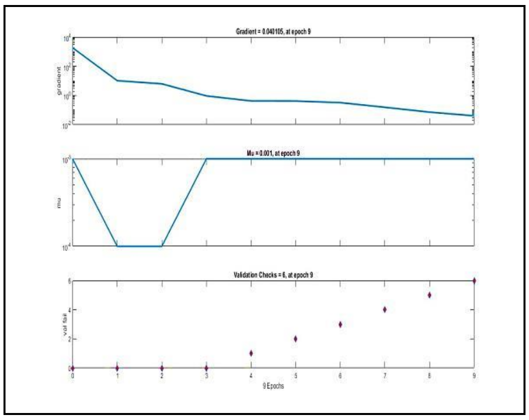
Data Division:	Random	dividerand
Training:	Levenberg-Marquardt	trainlm
Performance:	Mean Squared Error	mse
Calculations:	MEX	

Training Plots

Figure.6. Neural Network training by Matlab

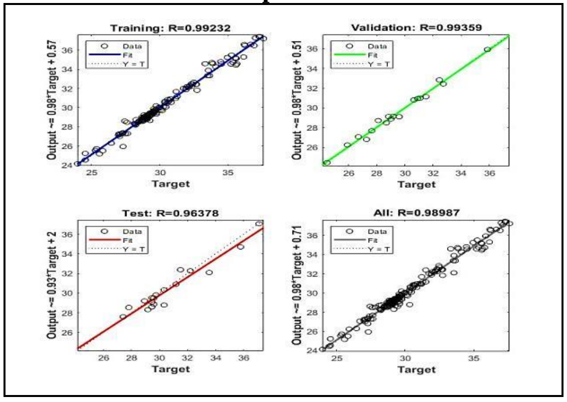
The following figure represents a set of figures for the most important results of neural network analysis.

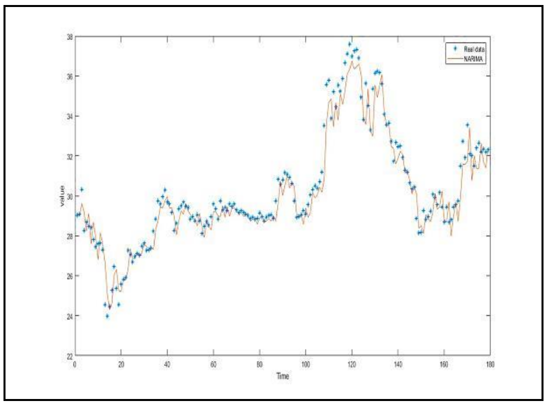




(a) Best Validation Performance is 0.90795 at the epoch

(b) Gradient, Mu, and Validation checks





(c) Output Training and Validation

(d) Real Data to NARMAX

Figure.7. Results of neural network analysis

4.3. Beta-t-EGARCH Analysis:

After drawing the time series in Figure 1 and converting the time series into a returns series, we now draw the ACF and PACF.

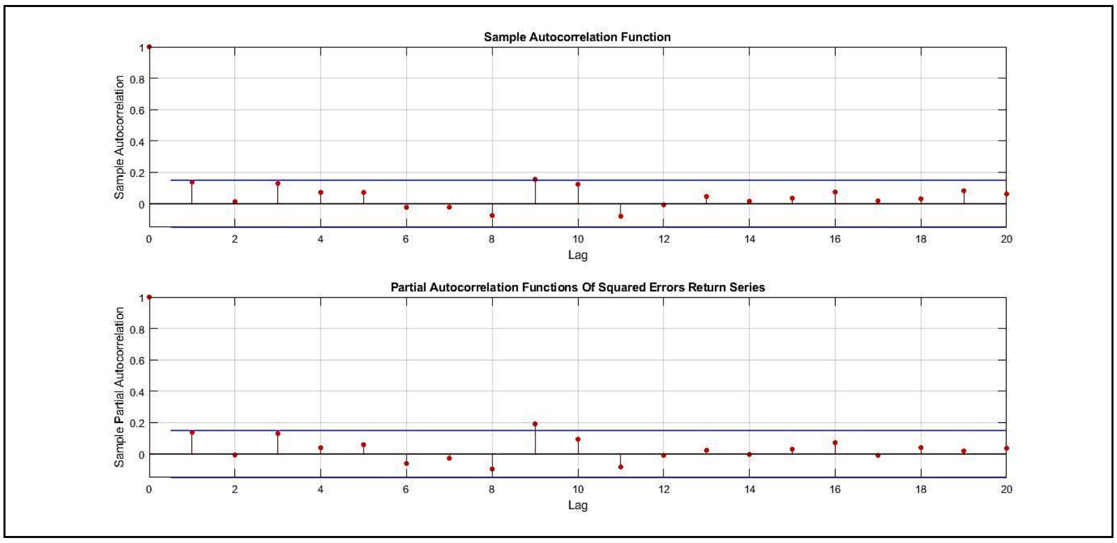


Figure.8. Plot of square error series

To discover whether or not there is an effect of heterogeneity of variance by finding the series of squared errors of the series of returns.

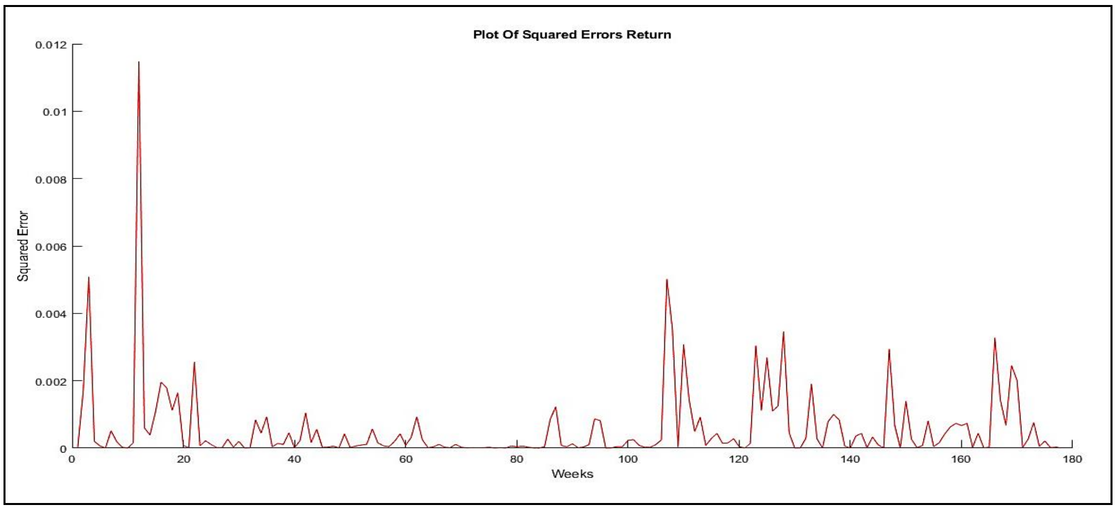


Figure.9. Plot of square error series

In order to detect the effect of heterogeneity, we will use the Ljung-Box test.

Now comes the stage of adaptation and estimation of the parameters of the model using best-order criteria, and AIC and BIC criteria were used.

Table.3. AIC and BIC criteria

Beta-t-EGRACH(Q,P)	AIC	BIC
Beta-t-EGRACH(1,1)	-848.9112	-833.0023
Beta-t-EGRACH(1,2)	-855.3407	-833.0683
Beta-t-EGRACH(1,3)	-855.3715	-826.7355
Beta-t-EGRACH(2,1)	-859.9166	-840.8259
Beta-t-EGRACH(2,2)	-857.7030	-832.2487
Beta-t-EGRACH(2,3)	-854.2385	-822.4206
Beta-t-EGRACH(3,1)	-859.1822	-836.9097
Beta-t-EGRACH(3,2)	-855.7059	-827.0698
Beta-t-EGRACH(3,3)	-878.5235	-843.5238

Therefore, the best model according to AIC and BIC standards is Beta-t-EGRACH(3,3), and the estimated parameters of this model are:

$\alpha_1 = -0.54015$	$\alpha_2 = 0.36923$	$\alpha_3 = 1$
$\beta_1 = 0.60676$	$\beta_2 = 0.67342$	$\beta_3 = -0.89064$
$Leverage_1 = 0.02226$	$Leverage_2 = -0.019084$	$Leverage_3 = -0.079213$
$constant = 4.8754$		

The stability condition for variance is achieved by the following equation:

$$\left|\sum \beta_j\right| = |0.60676 + 0.67342 + -0.89064| = 0.38954 < 1$$

We used GARCH models and QARMA models together to predict the original time series. GARCH models cannot predict the original series because they are variance models.

Table.4. AIC and BIC criteria for QARMA(p,q)-Beta-t-EGRACH(3,3)

QARMA(p,q)-Beta-t-EGRACH(3,3)	**AIC**	**BIC**
QARMA(1,0)-Beta-t-EGRACH(3,3)	-867.054	-824.7640
QARMA(1,1)-Beta-t-EGRACH(3,3)	-873.0963	-827.1219
QARMA(1,2)-Beta-t-EGRACH(3,3)	-866.0765	-821.6801
QARMA(1,3)-Beta-t-EGRACH(3,3)	868.7536	-825.481
QARMA(1,4)-Beta-t-EGRACH(3,3)	869.8645	-824.6382
QARMA(2,0)-Beta-t-EGRACH(3,3)	-872.546	-825.7903
QARMA(2,1)-Beta-t-EGRACH(3,3)	-874.5028	-826.8251
QARMA(2,2)-Beta-t-EGRACH(3,3)	-875.0684	-827.3417
QARMA(2,3)-Beta-t-EGRACH(3,3)	-871.7562	-825.8914
QARMA(2,4)-Beta-t-EGRACH(3,3)	-868.421	-823.7047

So, the model QARMA(2,2)-Beta-t-EGRACH(3,3) is the best model according to AIC and BIC criteria.

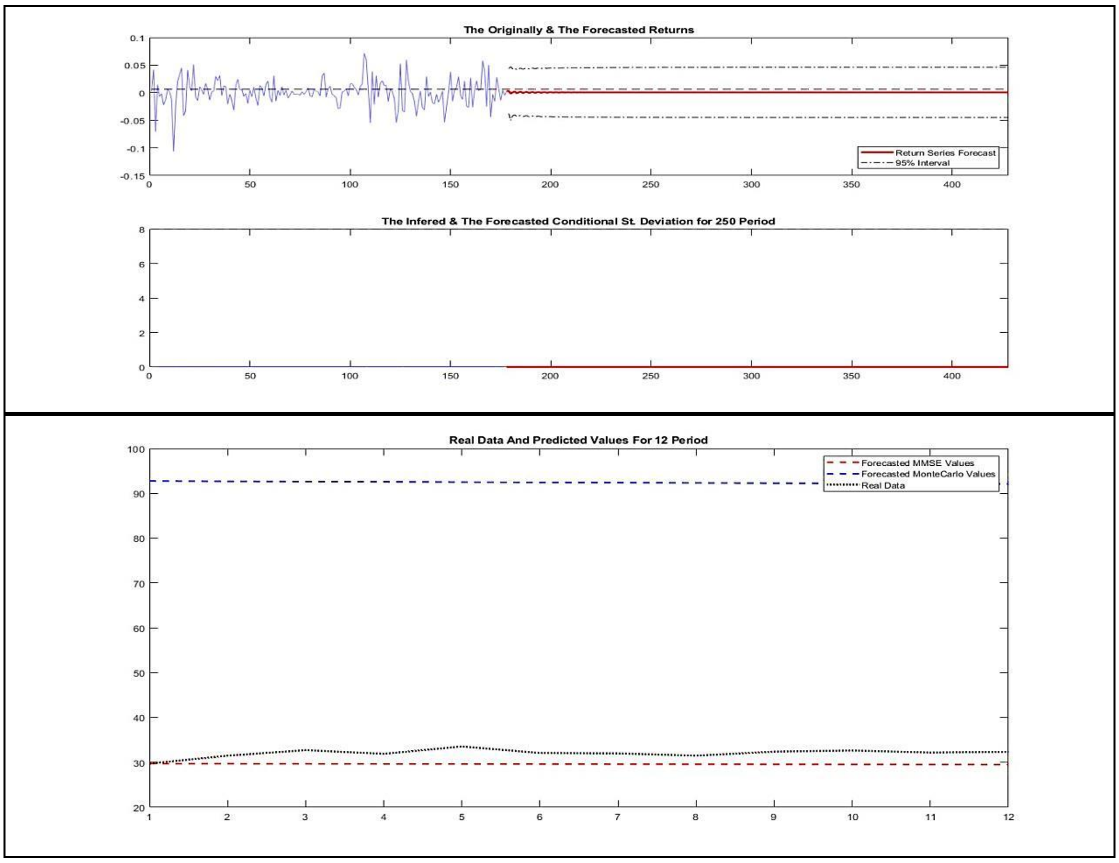


Figure.10. The inferred and forecasting Returns and Real data, prediction of 12 periods

4.4. RBFNN Analysis:

The data series is divided into two groups. The first group is called a test set with 80% of the observations of the studied series, and a training set with 20%, and 10 hidden layers with a stopping time of 1000. Where Table 5 represents the values of the training set for the ENN network, while Table 6 represents the values Network training algorithms.

To find the output of an RBF neural network in an ANN, follow these steps. To prepare your training data, get the input patterns and target outputs ready. If needed, normalise or standardise the input data for consistent scaling. Use the RAND function to generate distributed numbers from 0 to 1. The NEWRB function creates a radial basis network that approximates a defined function. It takes a training set, targets, a sum-squared error goal of 0.2, and a spread constant of 1. The spread of the radial basis neuron B is set to a very large number. We will calculate the distance between a data point and the centre of the grid using equation (27). The results are as follows:

Table.5. Training process of RBFNN

Unit	Initial value	Stopped value	Target value
Epoch	0	116	1000
Elapsed Time	-	00:00:27	-
Performance	24.9	5.11	0
Gradient	257	11.8	1e-05
Validation Checks	0	6	6

Table.6. Dimensions, connections, sub objects, functions, weight and bias values, and methods of RBFNN

Dimensions	connections	sub objects
Num Inputs: 1	Bias Connect: [1; 1]	input: Equivalent to inputs{1}
Num Layers: 2		inputs: {1x1 cell array of 1 input}
Num Outputs: 1	Input Connect: [1; 0]	layers: {2x1 cell array of 2 layers}
Num Input Delays: 0		biases: {2x1 cell array of 2 biases}
Num Layer Delays: 0	Layer Connect: [0 0; 1 0]	Input Weights: {2x1 cell array of 1 weight}
Num Feedback Delays: 0		Layer Weights: {2x2 cell array of 1 weight}
Num Weight Elements: 538	Output Connect: [0 1]	
Sample Time: 1		
functions	**weight and bias values**	**methods**
derivFcn: 'default deriv'	IW: {2x1 cell} containing 1 input weight matrix	adapt: Learn while in continuous use
initFcn: 'initlay'		configure: Configure inputs & outputs
Perform Fcn: 'mse'	LW: {2x2 cell} containing 1 layer weight matrix	gensim: Generate Simulink model
Perform Param: .regularization, .normalization		init: Initialize weights & biases
Plot Fcns: {}	b: {2x1 cell} containing 2 bias vectors	perform: Calculate performance
Plot Params: {1x0 cell array of 0 params}		sim: Evaluate network outputs given inputs
		train: Train network with examples
		view: View diagram
		Uncon figure: Uncon figure inputs & outputs

As for the following, Figure 11 represents the values of the training and validation set for the best validity. Figure 12 represents the training case. The error values in the histogram with 20 bins can be observed in Figure 13. The response of the output element-1 for the time series-1 is shown in Figure 14. Finally, Figure 15 represents the correlations between the input and output errors of the neural network.

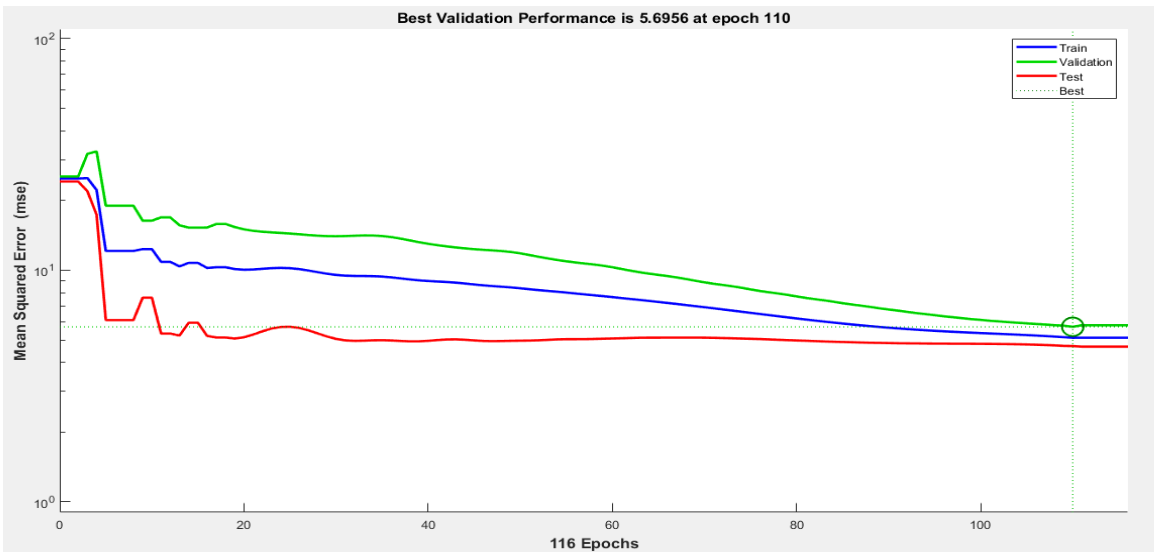


Figure.11. The values of the training and validation set for the best validity

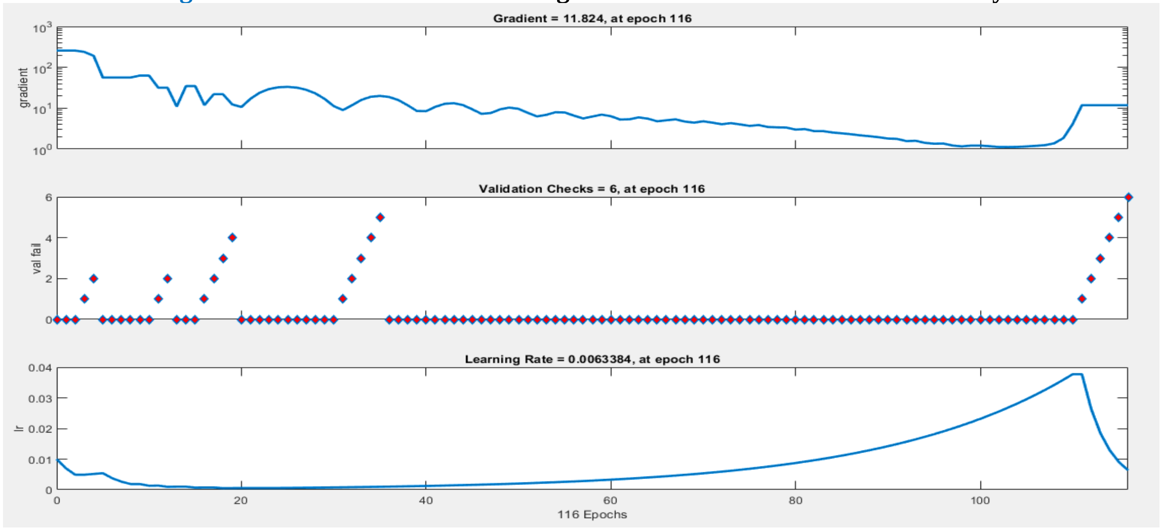


Figure.12. The training case

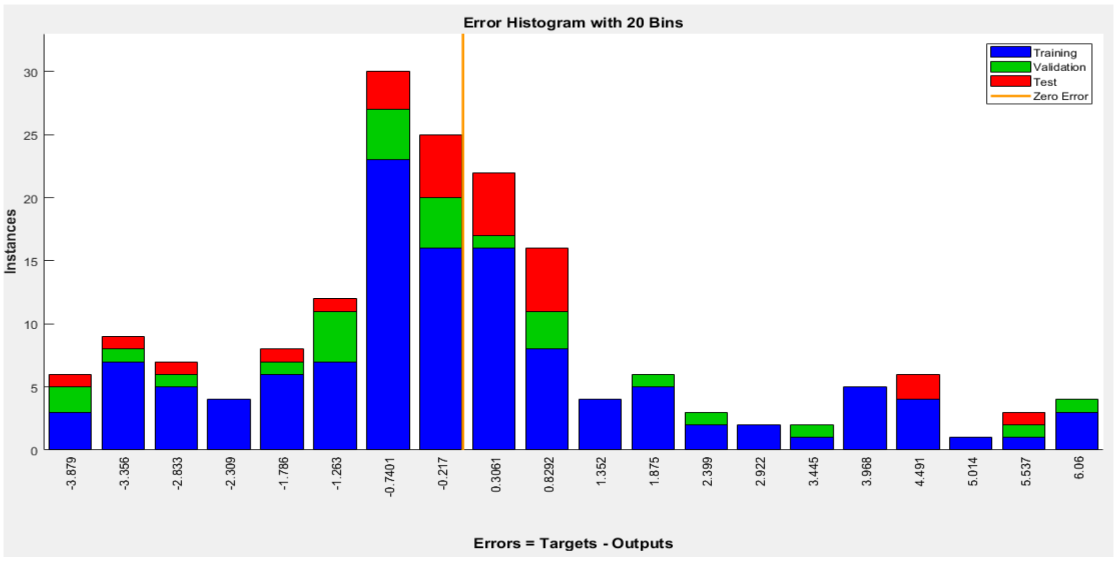


Figure.13. The error values in the histogram with 20 bins can be observed

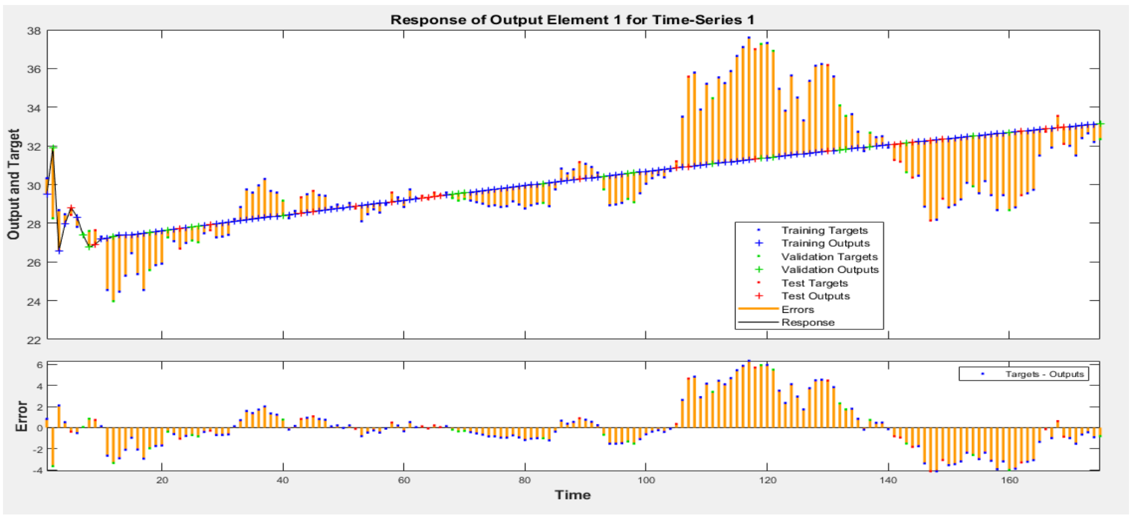


Figure.14. **The response of the output element-1 for the time series-1**

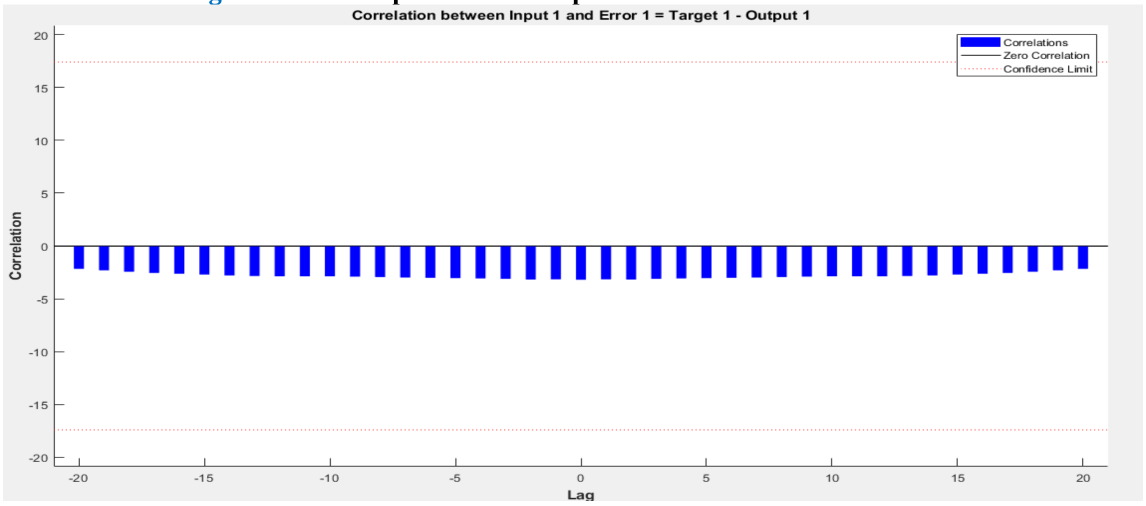


Figure.15. **The correlations between the input and output errors of the neural network**

And by using comparison criteria MSEANN, we get the value 7.4266e-04. Finally we plot the real data to FRBNN.

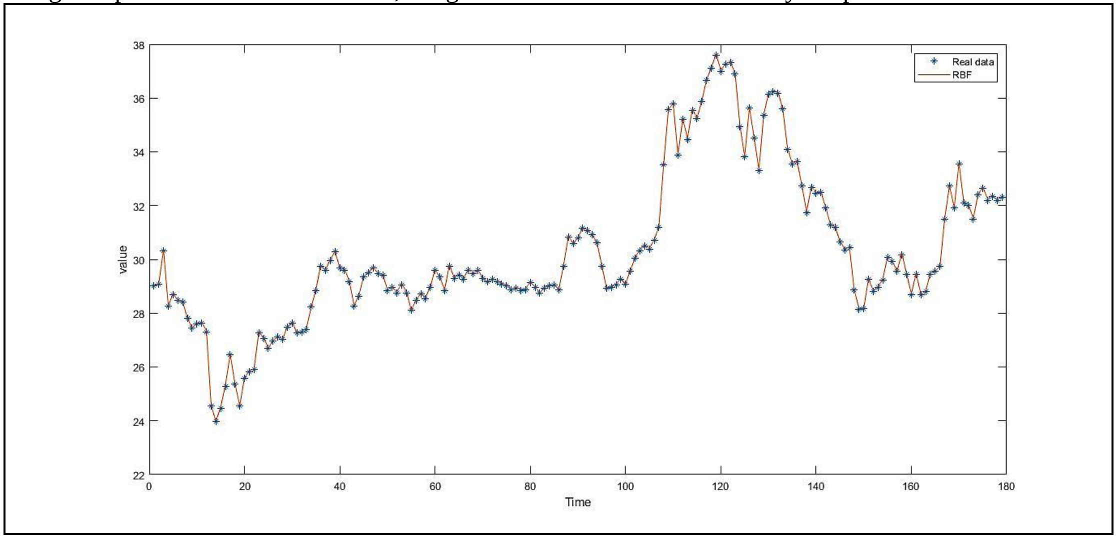


Figure.16. **Real Data to RBFNN**

5. Comparison:

The best comparison that can be made between the previous three models is to maintain an evaluation set of data represented by the last 10 observations of the time series to evaluate the performance of the used models. The following table shows the actual and predicted values for each model:

Real Data	Prediction by NARMAX	Prediction by Beta-t-EGARCH	Prediction by RBFNN
33.55	31.5853	33.0033	33.553
32.1	31.7299	32.1833	32.1117
32	33.4164	33.8233	32.0177
31.5	30.7584	31.7133	31.5411
32.4	31.9973	32.6733	32.5411
32.65	31.3494	32.9233	32.6522
32.2	31.378	32.4733	32.2084
32.35	32.6186	32.6233	32.3825
32.2	31.391	32.4733	32.2
32.3	32.3	32.5733	32.3324
MSE	0.5448	0.44064	7.4266e-04

6. Conclusion:

This paper presents a comparison of three different nonlinear time series modelling approaches NARMAX (Nonlinear Autoregressive Moving Average with Exogenous Inputs), Beta t EGARCH (Beta-t-Exponential Generalized Autoregressive Conditional Heteroscedasticity), and Radial Basis Function Neural Networks (RBFNN) applied to weekly stock market index data.

The key points are:

- NARMAX is a flexible nonlinear model that can represent a wide range of dynamical systems. Parameter estimation uses a least squares approach. Model structure selection uses error reduction ratio criteria.
- Beta-t-EGARCH is an extension of EGARCH that models conditional variance using the beta-t distribution. It is combined with ARMA models to forecast the time series. Parameters are estimated to minimize information criteria like AIC and BIC.
- RBFNN is a 3 layer neural network using radial basis functions in the hidden layer. It has a nonlinear mapping from input to hidden layer and a linear mapping from hidden to output layer. Training involves unsupervised learning of RBF centres/widths and supervised learning of output weights.
- The models were applied to weekly closing price data for a stock market index over 187 weeks. The data was divided into training and test sets.
- Model orders were selected by minimizing error criteria like MSE. RBFNN gave the lowest test MSE of 7.4266e-04, outperforming NARMAX (MSE = 0.5448) and Beta t EGARCH (MSE=0.44064).

- The study concludes RBFNN provides the most accurate forecasts on this data. But all models gave reasonably good predictions. Overall the paper provides a useful practical comparison of nonlinear time series modelling techniques.

Funding

None

ACKNOWLEDGEMENT

None

CONFLICT OF INTERESTS

None

References:

[1] S. A. Billings, “Identification of nonlinear systems using parameter estimation techniques,” Institute of Electrical Engineers Conference, 1981.

[2] I. J. Leontaritis and S. A. Billings, “Input-output parametric models for non-linear systems part II: stochastic non-linear systems,” International journal of control, vol. 41, pp. 329–344, 1985.

[3] A. Rahrooh and S. Shepard, “Identification of nonlinear systems using NARMAX model,” Nonlinear Analysis: Theory, Methods & Applications 71, vol. 12, pp. e1198–e1202, 2009.

[4] P. F. Retes and L. A. Leite, “NARMAX model identification using a randomised approach,” International Journal of Modelling, Identification and Control, vol. 31, pp. 205–216, 2019.

[5] Y. Yu, C. Zhang, and M. Zhou, “NARMAX model-based hysteresis modeling of magnetic shape memory alloy actuators,” IEEE Transactions on Nanotechnology, vol. 19, pp. 1–4, 2019.

[6] S. Billings and H.-L. Wei, “NARMAX model as a sparse, interpretable and transparent machine learning approach for big medical and healthcare data analysis,” in IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems, IEEE, 2019.

[7] Harvey, Andrew C., and Tirthankar Chakravarty. "Beta-t-(e) garch." (2008).

[8] D. B. Nelson, “Conditional heteroskedasticity in asset returns: A new approach**reprinted from Econometrica59 (1991), 347–370,” in Modelling Stock Market Volatility, Elsevier, 1996, pp. 37–64.

[9] S. Blazsek and M. Villatoro, “Is Beta-t-EGARCH (1, 1) superior to GARCH (1, 1),” Applied Economics, vol. 47, pp. 1764–1774, 2015.

[10] S. Blazsek and V. Mendoza, “QARMA-Beta-t-EGARCH versus ARMA-GARCH: An application to S&P 500,” Applied Economics, vol. 48, pp. 1119–1129, 2016.

[11] R. Liao, W. Yamaka, and S. Sriboonchitta, “Exchange rate volatility forecasting by hybrid neural network Markov switching beta-t-EGARCH,” IEEE Access, vol. 8, pp. 207563–207574, 2020.

[12] A. Ayala, S. Loretta, and A. Blazsek, “Score function scaling for QAR plus Beta-t-EGARCH: an empirical application to the S&P 500,” Applied Economics, pp. 1–14, 2023.

[13] M. Powell, “Radial basis functions for multivariable interpolation: A review,” Algorithms for the Approximation of Functions and Data, 1985.

[14] G. Montazer, D. Giveki, M. Karami, and H. Rastegar, “Radial Basis Function Neural Networks : A Review,” Computer Reviews Journal, vol. 1, pp. 52–74, 2018.

[15] Q. Que and M. Belkin, “Back to the future: Radial basis function network revisited,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 8, pp. 1856–1867, 2020.

[16] N. Bugshan, I. Khalil, N. Moustafa, M. Almashor, and A. Abuadbba, “Radial basis function network with differential privacy,” Future Gener. Comput. Syst., vol. 127, pp. 473–486, 2022.

[17] S. A. Billings, Non-linear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains. John Wiley & Sons, 2013.

[18] M. A. Alves, M. V. Corrêa, and L. A. Aguirre, “Use of self-consistency in the structure selection of NARX polynomial models,” Int. J. Model. Identif. Control, vol. 15, no. 1, p. 1, 2012.

[19] L. Piroddi and W. Spinelli, “An identification algorithm for polynomial NARX models based on simulation error minimization,” Int. J. Control, vol. 76, no. 17, pp. 1767–1781, 2003.

[20] R. T. Baillie, T. Bollerslev, and H. O. Mikkelsen, “Fractionally integrated generalized autoregressive conditional heteroskedasticity,” J. Econom., vol. 74, no. 1, pp. 3–30, 1996.

[21] R. F. Engle, “Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation,” in Arch, Oxford University PressOxford, 1995, pp. 1–23.

[22] A. Harvey and G. Sucarrat, “EGARCH models with fat tails, skewness and leverage,” Comput. Stat. Data Anal., vol. 76, pp. 320–338, 2014.

[23] Y. Wu, H. Wang, B. Zhang, and K.-L. Du, “Using radial basis function networks for function approximation and classification,” ISRN Appl. Math., vol. 2012, pp. 1–34, 2012.

[24] A. Meyer-Bäse, Pattern Recognition and Signal Analysis in Medical Imaging. Academic Press, 2004.

[25] M. Haghbin, A. Sharafati, and D. Motta, “Prediction of channel sinuosity in perennial rivers using Bayesian Mutual Information theory and support vector regression coupled with meta-heuristic algorithms,” Earth Sci. Inform., vol. 14, no. 4, pp. 2279–2292, 2021.

[26] Z. Abbood, “Speaker identification model based on deep neural networks,” Iraqi Journal For Computer Science and Mathematics, vol. 3, pp. 108–114, 2022.

[27] Y. Niu and K. Andrei, “Identification method of power Internet attack information based on machine learning,” ijcsm, pp. 1–7, 2022.