भाकृअनुप
ICAR

Twenty-One Day Online Training Manual
on

राअस्ट्रैप्रसं
NIASM

Advanced Statistical and Machine Learning Techniques for Data Analysis Using Open-Source Software for Abiotic Stress Management in Agriculture

16 July – 5 August 2025

Training Manual

Volume 03

Time Series Analysis & Machine Learning for Predictive Modeling

Edited by

Dr. Santosha Rathod
Dr. Nobin Chandra Paul
Ms. Ponnaganti Navyasree
Mr. K Ravi Kumar
Dr. Prabhat Kumar

Organised by:

School of Social Science and Policy Support
ICAR-National Institute of Abiotic Stress Management, Baramati, Maharashtra - 413115

Time Series Analysis & Machine Learning for Predictive Modeling

Editors

Santosha Rathod

Nobin Chandra Paul

Ponnaganti Navyasree

K Ravi Kumar

Prabhat Kumar

2025

School of Social Science and Policy Support
ICAR-National Institute of Abiotic Stress
Management, Baramati – 413115
Maharashtra, India

Title: Time Series Analysis & Machine Learning for Predictive Modeling

Editors: Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K Ravi Kumar, Prabhat Kumar

Published by: ICAR-National Institute of Abiotic Stress Management, Malegaon Khurd, Baramati – 413115, Maharashtra, India.

Edition: I

Volume: 3

ISBN: 978-81-985897-0-5

Citation:

Rathod, S., Paul, N. C., Ponnaganti, N., Kumar, K. R., & Kumar, P. (Eds.). (2025). *Time Series Analysis & Machine Learning for Predictive Modeling: Training manual of the twenty-one-day online training programme on "Advanced statistical and machine learning techniques for data analysis using open-source software for abiotic stress management in agriculture"* (Vol. 3). ICAR-National Institute of Abiotic Stress Management. ISBN 978-81-985897-0-5.

CONTENTS

Trend Analysis for Climatic Data

Naveena K

Centre for Water Resources Development and Management (CWRDM), Kozhikode - 673571, Kerala, India.
Email: naveenak@cwrdm.org

Introduction:

Climate change is a long-term continuous change in average weather conditions either increase or decrease pattern. Climate variability looks at changes that occur within smaller timeframes, such as a month, a season or a year. Random fluctuation in climate patterns makes important to study both long-term and short term moments to conclude precisely about climate. During changing climatic scenarios, the identification of weather movements precisely is very essential for planning and implementation of various activities, including agricultural practices, cropping system, mitigation of landslides and control of flood damages. Extremities in weather events make a greater impact on the lives of the individuals. So, it is important to determine their potential long term pattern (trends) accurately.

In order to detect trends in weather parameters, several statistical methods have been put to practice. The non-parametric tests like Mann-Kendal test, Modified Man Kendal test; innovative trend analysis are commonly using methods for trend analysis using weather information. Linear regression method is one of the parametric methods for trend analysis, but due to its predefined assumption rigidity limits the usage for long term moment analysis.

Mann-Kendall test (MK test)

Mann-Kendall test is the nonparametric test to detect the long-term moment in the time series. Test statistic for the MK test will be calculated using the sign of differences rather than the values of the random variables so the trend value will be least affected by nonlinearities compared to the parametric test like linear regression (*Sanjeevaiah et al, 2021*).

The hypothesis considered under the test is:

Null Hypothesis (H_0) = There is no presence of a monotonic trend in the time series.

Alternative Hypotheses (H_a) = There is a presence of a monotonic trend that may be decreasing or increasing in the time series.

The Mann-Kendall test statistic defined by S is calculated using the formula:

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} sign(x_j - x_i) \qquad \dots\dots\dots\dots(2)$$

Where x_j and x_i are the annual values in years *j* and *i*, *j*>*i* respectively, and n is the number of observations. The value of $sign(x_j - x_i)$ is computed as follows:

$$sign(x_j - x_i) = \begin{cases} 1 \; if \; (x_j - x_i) > 0 \\ 0 \; if \; (x_j - x_i) = 0 \\ -1 \; if \; (x_j - x_i) < 0 \end{cases} \qquad \dots\dots(3)$$

For large samples (n>10), the test is conducted using a normal approximation with the mean ($E[S]$)and the variance ($Var(S)$) as follows:

$$E[S] = 0$$

$$Var(S) = \frac{1}{18}\left[n(n-1)(2n+5) - \sum_{p=1}^{q} t_p(t_p - 1)(2t_p + 5)\right] \qquad \dots\dots\dots\dots\dots(4)$$

Here *q* is the number of tied groups, and t_p is the number of data values in the *p*th group. The values of S and Var(S) are used to compute the test statistic Z as follows:

$$Z = \begin{cases} \dfrac{S-1}{\sqrt{Var(S)}} & if \; S > 0 \\ 0 & if \; S = 0 \\ \dfrac{S+1}{\sqrt{Var(S)}} & if \; S < 0 \end{cases} \qquad \dots\dots\dots\dots\dots(5)$$

The presence of a statistically significant trend is evaluated using the *Z* value. The upward trend in the series will be indicated by a positive Z value and the downward trend by a negative Z value. H_0 is rejected if the absolute value of *Z* is greater than $Z_{1-\alpha/2}$, where $Z_{1-\alpha/2}$ is obtained from the standard normal cumulative distribution tables. The Z values were tested at 0.05 level of significance.

The monotonic relationship between lag values x_i and x_{i+1} is measured by Kendall's tau correlation coefficient (τ)

$$\tau = \frac{s}{n(n-1)/12} \qquad \dots\dots\dots\dots(6)$$

Wallis and Moore Phase-Frequency test (WM test)

The Wallis and Moore phase-frequency test is used to test the independency in series (*Wallis & Moore, 1941*). The hypothesis considered under the test is,

H_0: The series is random in nature, against the

H_a: The series is nonrandom in nature.

The test statistic in the ordered series (n>30) is

$$Z = \frac{\left| h - \left(\frac{2n-7}{3}\right) \right|}{\sqrt{\frac{16n-29}{90}}} \qquad \dots\dots\dots\dots\dots\dots\dots (7)$$

Where h is the number of phases. If n≤30 then a correction of 0.5 is included in the denominator.

Modified Man-Kendall test (MMK test)

Even though The Mann-Kendall test is the commonly used statistical tool for testing monotonic trends, it assumes that observation should be free from autocorrelation. The positive autocorrelation in the series will mislead the trend results (*Yue et al, 2002*) from the MK test by increasing the probability of significant trend. This can be corrected by Modified Mann–Kendall test (Yue and Wang 2004), which removes the linear trend component from the series and then, the effective sample size is calculated using the lag-1 serial correlation coefficient. The variance correction using an effective sample size will eliminate the effect of autocorrelation present in the time series (*Sanjeevaiah et al, 2021)*. So, in this method variance of the MK test will be replaced by modified variance and the remaining procedure will proceed the same to identify the trend. The accuracy of the MMK test is higher than the original MK test with respect to the empirical significance level.

The modified variance (V(S)*) using Effective Samples Size is given by:

$$V(S)^* = V(S).\frac{n}{n^*} \qquad \dots\dots\dots..(8)$$

Where n is the Actual sample size (ASS), n/n* is the correction Factor (CF) and n* is the Effective Sample Size.

The Effective Sample Size can be computed by:

$$n^* = \frac{n}{1 + 2\rho_k \sum_{k-1}^{n-1}(1 - \frac{k}{n})} \qquad \dots\dots\dots(9)$$

Where ρ_k is the lag-k autocorrelation coefficient which can be estimated by sample autocorrelation for kth lag (ρ_k).

The MMK test was done using "modifiedmk" package of R software.

Sen's slope estimator

The Sen's nonparametric method is used to estimate the unit changes per year in the series. Here the trend in the series can be assumed to be linear.

$$f(t) = Qt + B \qquad \dots\dots\dots..(10)$$

Where B is a constant, *t* is time and Q is the slope. The slopes of all data value pairs will be calculated to estimate the Q using the equation:

$$Q_i = \frac{x_j - x_k}{j - k} \qquad \dots\dots\dots(11)$$

Where x_j and x_k are data values at time j and k *(j>k)* respectively. If there are *n* values x_j in the time series, there will be as many as $N = n(n-1)/2$ slope estimates Q_i. The Sen's estimator of slope is the median of these *N* values of Q_i. The N values of Q_i are ranked from the smallest to the largest and the Sen's estimator is,

$Q = Q_{\left[\frac{(N+1)}{2}\right]}$, if N is odd or ………………(12)

$Q = \frac{1}{2}\left(Q_{\left[\frac{N}{2}\right]} + Q_{\left[\frac{(N+2)}{2}\right]}\right)$, if N is even. ………………...(13)

Illustration for trend analysis

Examination of spatiotemporal dynamics of rainfall pattern of Wayanad region of Kerala

Wayanad district of Kerala is one of the high altitude and high rainfall regions of northern Kerala, where the majority of livelihoods depend on agricultural activities. Recent changes in global climate making a greater impact on the distribution of rainfall, so it's important to identify potential rainfall trends accurately. For the study, we have considered two gauging stations like Mananthavady, and Vythiri for 33 years of annual rainfall data of all the months and all the seasons (South-West Monsoon; SWM (June to September), North East Monsoon; NEM (October to November), winter (December to February), and summer (March to May)).

The statistical trend analysis of monthly, seasonal and annual rainfall (mm) from 1986 to 2018 is presented in Table 1. Wald-Wolfowitz Test of randomness results indicates January (2.83), February (1.98), March (1.98), and April (2.41) in the Manthawadi region, January (3.26), February (2.40), and November (1.98) rainfall in Vettari showing Z-statistic value more than Z-critical value (1.96) for 5 % level of significance and indicates the significant dependency of the lag period. Block bootstrapping in Mann–Kendall trend test results confirms that Wayanad is under a downward rainfall trend in almost all the months, seasons, and annual rainfall. Where both the stations of the Wayanad region showing a significant downward trend for Post monsoon season rainfall. An average every year 14 mm post-monsoon rainfall is reducing in Wayanad (Mananthavady (-8.56 mm), Vythiri (-10.94 mm), and Ambalavayal (-8.50 mm)). Finally, when we compared the last 33-year northwest rainfall, even the number of rainy days (> 2.5 mm/day) in post-monsoon also taken the negative trend ((Mananthavady (Z=-1.75), and Vythiri (Z=-1.42)). About 63 per cent reduction in the number of rainy days was observed during last 10 years compared to previous past years. This has a significant effect crop productivity especially on the Coffee Production in the Wayanad region, as Coffee is the predominantly grown cash crop. Reduction in North-East monsoon increases the stress period of coffee crop (reduction in moisture level at the root zone), which results in early maturity of the crop (Awati et al, 2016). This stress along with summer showers create early flowering or irregular flowering in Coffee. Establishment of new clearings is difficult without the supplementary irrigation in case of no North-East Monsoon. Fear of Mealy bugs and sucking pests infestation will be more if there is a reduction in N-E monsoon.

Table 1: Rainfall trend analysis for waynad region, Kerala

	WM-test	MK- test	MMK test	Sen's Slope	WM-test	MK- test	MMK test	Sen's Slope
	Mananthavady				Vythiri			
Jan	2.83**	-1.17	-1.29	0.00	3.26**	-0.76	-0.88	0.00
Feb	1.98*	-1.19	-1.23	0.00	2.40*	-1.30	-1.16	0.00
Mar	1.98*	-0.11	0.10	0.00	1.56	0.23	0.27	0.074
Apr	2.41**	-1.79	-1.84	-3.34	0.14	-1.18	-1.29	-4.92
May	1.13	0.07	-1.78	-0.25	0.57	-0.54	-0.367	-1.38
Jun	0.99	-0.29	-0.18	-2.9	0.99	-1.15	-0.76	-10.21
Jul	0.70	-0.33	-0.20	-2.75	1.84	-1.00	-0.90	-11.74
Aug	0.14	-0.29	-0.34	-2.04	0.71	-1.25	-1.08	-7.25
Sep	0.70	0.17	0.20	0.68	0.71	-0.31	-0.30	-2.30
Oct	1.84	-1.78	-1.67	-6.03	0.14	-1.60	-1.69	-7.72
Nov	0.15	-1.61	-1.64	-2.49	1.98*	-2.05*	-2.13*	-3.71
Dec	0.70	-0.89	-0.97	-0.52	1.55	-1.56	-1.93	-3.25
SWM	0.28	0.01	0.05	0.00	0.71	-0.45	-0.37	-11.01
NEM	0.29	-2.06*	-1.98*	-8.56	1.84	-2.34*	-2.15*	-10.94
Winter	0.28	-1.14	-1.13	-0.96	1.56	-1.71	-2.04*	-3.81
Summer	0.28	-0.99	-1.26	-0.28	0.14	-1.25	-1.23	-0.15
Annual	1.13	0.015	-0.01	0.01	0.71	-0.73	-0.623	-18.50

Conclusion

Recent changes in global climate making a greater impact on the distribution of weather series, so it's important to identify potential trends accurately. Modified Mann–Kendall test outperform compare to Mann-Kendal test during the series under consideration exhibit significant auto correlation.

Code for trend analysis

```
##Mann-Kendall Test for Trend in R
Data<-read.csv(file.choose(),header=TRUE)
install.packages("trend")
library(trend)
#####Mann-Kendall Test
mk.test(x, alternative = "two.sided")
#Magnitude of trend
#Sen's slope
sens.slope(x)
#Seasonal trend
datas<-read.csv(file.choose(),header = TRUE)
data<-ts(datas$Rainfall,start=c(1890,1),end = c(2008,12),frequency = 12)
```

```
smk.test(data)
#Test for the Randomness
##Wallis and Moore phase-frequency test
wm.test(x)
#Modified Mankendal test
install.packages("modifiedmk")
library(modifiedmk)
#Mann-Kendall Test applied to Trend Free Pre-Whitened Time Series Data in Presence of Serial Correlation Using Yue and Pion (2002) Approach
tfpwmk(x)
#Modified Mann-Kendall Test For Serially Correlated Data Using Hamed and Rao (1998) Variance Correction Approach
mmkh(x)
```

Where x is the series under consideration for trend analysis

References:

Sanjeevaiah, S. H., Rudrappa, K. S., Lakshminarasappa, M. T., Huggi, L., Hanumanthaiah, M. M., Venkatappa, S. D., ... & Sreeman, S. M. (2021). Understanding the Temporal Variability of Rainfall for Estimating Agro-Climatic Onset of Cropping Season over South Interior Karnataka, India. Agronomy, 11(6), 1135.

Jaiswal, R. K., Lohani, A. K., & Tiwari, H. L. (2015). Statistical analysis for change detection and trend assessment in climatological parameters. Environmental Processes, 2(4), 729-749.

Buishand, T. A. (1982). Some methods for testing the homogeneity of rainfall records. Journal of hydrology, 58(1-2), 11-27.

Wallis, W. A., & Moore, G. H. (1941). A significance test for time series and other ordered observations. In A Significance Test for Time Series and Other Ordered Observations (pp. 1-67). NBER.

Yue, S., & Pilon, P. (2004). A comparison of the power of the t test, Mann-Kendall and bootstrap tests for trend detection/Une comparaison de la puissance des tests t de Student, de Mann-Kendall et du bootstrap pour la détection de tendance. Hydrological Sciences Journal, 49(1), 21-37.

Yue, S., Pilon, P., Phinney, B., and Cavadias, G. (2002). The influence of autocorrelation on the ability to detect trend in hydrological series. Hydrological Processes, 16(9): 1807–1829.

Shpakova, R. N., Kusatov, K. I., & Mustafin, S. K. (2020, April). Spatiotemporal Trends in Changes in the River Water Contents in the Sakha Republic (Yakutia). In IOP Conference Series: Earth and Environmental Science (Vol. 459, No. 5, p. 052062). IOP Publishing.

Elzopy, K. A., Chaturvedi, A. K., Chandran, K. M., Gopinath, G., & Surendran, U. (2021). Trend analysis of long-term rainfall and temperature data for Ethiopia. *South African Geographical Journal*, *103*(3), 381-394.

W. A. Wallis and G. H. Moore (1941): A significance test for time series and other ordered observations. Tech. Rep. 1. National Bureau of Economic Research. New York.

Ryberg, K. R., Hodgkins, G. A., & Dudley, R. W. (2020). Change points in annual peak streamflows: Method comparisons and historical change points in the United States. Journal of Hydrology, 583, 124307.

Time Series Analysis for Abiotic Stress Management

Santosha Rathod, Naveena K, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, Prabhat Kumar

1.ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115

2. Centre for Water Resources Development and Management (CWRDM), Kozhikode - 673571, Kerala, India.

Email: santosha.rathod@icar.org.in

1. Introduction:

Time series refers to an ordered sequence of values of a variable recorded at equally spaced time intervals. The process of analysing such data to extract meaningful insights is called time series analysis (TSA). The primary objective of time series modeling is to systematically study the historical behaviour of a variable to identify underlying patterns, trends, and seasonality—so that future values can be predicted. This makes time series forecasting a powerful tool for decision-making, as it enables researchers and policymakers to anticipate upcoming events based on past trends.

Time series analysis has been widely applied in fields like business, finance, economics, meteorology, hydrology, and engineering. In agriculture, and particularly in abiotic stress management, TSA plays a crucial role. Abiotic stresses such as drought, heat waves, cold spells, salinity, and floods often show periodic or trend-based behaviour over time. Understanding these patterns through time series forecasting allows for timely interventions, resource planning, and early warning systems to safeguard crops and improve resilience. For instance, forecasting future drought probabilities based on historical rainfall data, or predicting heat stress periods during crop flowering stages, are critical applications. In such cases, time series models such as ARIMA, SARIMA, Exponential Smoothing, and machine learning-based models (for example, LSTM) are employed to capture both short- and long-term dependencies in the data.

One of the essential properties of time series data is the dependence among successive observations, which distinguishes it from random or cross-sectional data. The accuracy and reliability of forecasts depend on both the quality and length of historical data available. According to Box and Jenkins pioneers of classical time series modelling minimum of 50 observations is generally recommended for robust model development and validation.

Thus, in the context of climate-resilient agriculture, TSA provides a scientific and data-driven foundation to tackle abiotic challenges by enabling forecasting, preparedness, and adaptation planning, ultimately contributing to more sustainable farming systems.

A time series that records the values of a single variable is referred to as a univariate time series, whereas one that involves multiple variables is known as a multivariate time series. Time series data can be categorized as either continuous or discrete. In a continuous time series, observations are captured at every moment in time, while in a discrete time series, data points are collected at specific, separate time intervals. Examples of continuous time series include temperature measurements, river flow rates, and chemical concentrations. In discrete time series, observations are typically recorded at regular intervals—such as hourly, daily, weekly, monthly, or annually. Although the observations occur at discrete points, the variable itself is usually treated as continuous and measured on a real number scale. Additionally, a continuous time series can be converted into a discrete one by aggregating data over predefined time intervals.

Time Series Analysis (TSA) generally follows two main forecasting approaches. The first involves predicting the present series based on the observed patterns in historical data, commonly referred to as the extrapolation method. The second approach, known as the explanatory method, estimates future outcomes by incorporating variables that influence the target phenomenon (Diebold and Lopez, 1996). In essence, statistical forecasting is the process of approximating the likelihood of future events based on available information. For example, in agriculture, farmers are generally interested in aspects like production, demand, consumption, and price of an item, etc., and all of these events, change with time. Statistical forecasting models are widely used for examining behavior of such time series data. So, forecasting is needed in almost all sectors viz. business production planning, multistage management decision analysis, staff scheduling, various management problems, crop yield and acreage forecasting, etc.

One simple method of describing a series is that of classical decomposition. The simplest and most basic approach to forecasting is the moving averages method, which assigns equal weights to all observations in the selected time window. However, this method does not differentiate between recent and older data points. To address this limitation, exponential smoothing methods were introduced as an improved approach that assigns exponentially decreasing weights to older observations, thereby giving more importance to recent data. These methods were initially developed as recursive techniques without relying on any specific assumptions regarding the distribution of the error terms.

Over time, it has been observed that exponential smoothing methods are, in fact, special cases of the more statistically rigorous Autoregressive Integrated Moving Average (ARIMA) models. Among the classical time series models, ARIMA remains one of the most important and widely applied due to its strong theoretical foundation and practical applicability.

The popularity of the ARIMA model is largely attributed to its linear statistical structure and the well-known Box-Jenkins methodology for model identification, estimation, and diagnostic checking (Box and Jenkins, 1970). For an extensive treatment of exponential smoothing techniques, the work of Makridakis et al. (1998) provides valuable insights. A practical guide to ARIMA modeling, including numerous case studies, is presented in Pankratz (1983). Furthermore, a comprehensive and rigorous exposition of ARIMA and related time series models is given in Box et al. (1994), which continues to serve as a foundational reference in time series analysis.

2. Components of TS:

A fundamental approach to analyzing a time series is classical decomposition, which assumes that the series can be broken down into four main components: trend, cyclical, seasonal, and irregular variations. The trend refers to the long-term direction of a series—whether it increases, decreases, or remains stable over time. For instance, population growth or housing development in a city typically exhibits an upward trend, while mortality rates or epidemic cases may show a downward trend. Seasonal variations are short-term, recurring fluctuations that occur within a year, often driven by climate, weather, cultural customs, or traditions—such as increased ice cream sales in summer or higher demand for woolen clothes in winter. Understanding these patterns is essential for businesses, retailers, and producers to plan effectively. Cyclical variations represent medium-term movements that repeat over extended periods, often spanning two or more years. These are commonly observed in economic and financial time series, such as the phases of the business cycle: prosperity, decline, depression, and recovery. In contrast, irregular or random variations are unpredictable and non-repeating disturbances caused by unforeseen events like wars, strikes, natural disasters, or political upheavals. Since these fluctuations do not follow any specific pattern, there is no standard statistical method to measure them. To account for the combined effects of these four components, time series models are typically structured using either multiplicative or additive forms

Multiplicative model: $Y(t) = T(t) * S(t) * C(t) * I(t)$

Additive model: $Y(t) = T(t) + S(t) + C(t) + I(t)$

Where, Y(t) is the original series, T(t) is the trend component, S(t) is the seasonal component, C(t) is the cyclic component and I(t) is the irregular component. The multiplicative model of a time series is based on the assumption that the four components—trend, cyclical, seasonal, and irregular—are not necessarily independent and can influence one another. In contrast, the additive model assumes that these components are independent and do not affect each other directly.

2.1 Trend analysis

The trend analysis was done in three steps. The first step is to detect the presence of increasing or decreasing trend using the nonparametric Mann-Kendall test, second step is estimation of magnitude or slope of a linear trend with the nonparametric Sen's Slope estimator, and third one is to develop regression models.

Calculation of the Mann-Kendal test

The Mann-Kendall test statistic *S* is calculated using the formula that follows:

$$S = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} sgn(x_j - x_i)$$

Where x_j and x_i are the annual values in years *j* and *i*, *j*>*i* respectively, and N is the number of data points. The value of $sgn(x_j - x_i)$ is computed as follows:

$$sgn(x_j - x_i) = \begin{cases} 1\ if\ (x_j - x_i) > 0 \\ 0\ if\ (x_j - x_i) = 0 \\ -1\ if\ (x_j - x_i) < 0 \end{cases}$$

This statistic represents the number of positive differences minus the number of negative differences for all the differences considered. For large samples (N>10), the test is conducted using a normal approximation (Z statistics) with the mean and the variance as follows:

$$E[S] = 0$$

$$Var(S) = \frac{1}{18}\left[N(N-1)(2N+5) - \sum_{p=1}^{q} t_p(t_p-1)(2t_p+5)\right]$$

Here q is the number of tied (zero difference between compared values) groups, and t_p is the number of data values in the pth group. The values of S and VAR(S) are used to compute the test statistic Z as follows

$$Z = \begin{cases} \frac{S-1}{\sqrt{Var(S)}} & if\ S > 0 \\ 0 & if\ S = 0 \\ \frac{S+1}{\sqrt{Var(S)}} & if\ S < 0 \end{cases}$$

The presence of a statistically significant trend is evaluated using the Z value. A positive value of Z indicates an upward trend and its negative value a downward trend. The statistic Z has a normal distribution. To test for either an upward or downward monotone trend (a two-tailed test) at α level of significance, H_0 is rejected if the absolute value of Z is greater than $Z_{1-\alpha/2}$, where $Z_{1-\alpha/2}$ is obtained from the standard normal cumulative distribution tables. The Z values were tested at 0.05 level of significance.

Sen’s slope estimator

To estimate the true slope of an existing trend (as change per year) the Sen's nonparametric method is used. The Sen’s method can be used in cases where the trend can be assumed to be linear.

$$f(t) = Qt + B$$

Where Q is the slope, B is a constant and t is time. To get the slope estimate Q, the slopes of all data value pairs is first calculated using the equation:

$$Q_i = \frac{x_j - x_k}{j - k}$$

Where x_j and x_k are data values at time j and k $(j>k)$ respectively. If there are n values x_j in the time series there will be as many as $N = n(n-1)/2$ slope estimates Q_i. The Sen’s estimator of slope is the median of these N values of Q_i. The N values of Q_i are ranked from the smallest to the largest and the Sen’s estimator is

$Q = Q_{\left[\frac{(N+1)}{2}\right]}$, if N is odd or $Q = \frac{1}{2}\left(Q_{\left[\frac{N}{2}\right]} + Q_{\left[\frac{(N+2)}{2}\right]}\right)$, if N is even.

To obtain an estimate of B in Equation f(t) the *n* values of differences $x_i - Qt_i$ are calculated. The median of these values gives an estimate of B.

Modified Mann–Kendall test

the Mann-Kendall test is the commonly using statistical tools for testing monotonic upward or downward trend of the variable of interest over time. The positive auto correlation in the series will miss lead the trend results (Yue et al, 2004) from Mann-Kendal test. Mann-Kendall test show significance results even though no trend in the series. The null hypothesis H0: there has been no trend in given series was tested against there has been a trend in given series. the hypothesis where were no trend, was rejected when the computed Z-transformed test Statistic value was greater in absolute value than the critical value $Z_{1-0.5\alpha}$, at 95% level of significance.

Illustration: Trend analysis

The *Manthawadi,* station of waynad (11.8014°N, 76.0044°E) district considered for the study. The daily rainfall data for the period 1987-2018 (33 years) is collected from India Meteorological Department, Pune. The methods adopted to study the characteristics of rainfall in this region are as follows: Mann–Kendall trend test (MK-test), Wald-Wolfowitz Test of randomness (WWTR), Sen's slope estimator and Block bootstrapping in Mann–Kendall trend test (BBMK).

Results

The statistical trend analysis of monthly, seasonal and annual rainfall (mm) from 1986 to 2018 is presented in Table 1. The data of the 12 months were combined into four seasons, south-west monsoon (June to September), northeast monsoon (October to November), winter (December to February), and summer (March to May). Wald-Wolfowitz Test of randomness results indicates January (2.83), February (1.98), March (1.98), and April (2.41) in the Manthawadi region showing Z-statistic value more than Z-critical value (1.96) for 5 % level of significance and indicates the significant dependency of the lag period. Block bootstrapping in Mann–Kendall trend test results confirms that Wayanad is under a downward rainfall trend in almost all the months, seasons, and annual rainfall.

Table 1: Trend analysis of Manthawadi region

	Randomness test	MK- test	BBMK test	Sen's Slope
		Mananthavadi		
Jan	2.83**	-1.17	-1.29	0.00
Feb	1.98*	-1.19	-1.23	0.00
Mar	1.98*	-0.11	0.10	0.00
Apr	2.41**	-1.79	-1.84	-3.34
May	1.13	0.07	-1.78	-0.25
Jun	0.99	-0.29	-0.18	-2.9
Jul	0.70	-0.33	-0.20	-2.75
Aug	0.14	-0.29	-0.34	-2.04
Sep	0.70	0.17	0.20	0.68
Oct	1.84	-1.78	-1.67	-6.03
Nov	0.15	-1.61	-1.64	-2.49
Dec	0.70	-0.89	-0.97	-0.52
Annual	1.13	0.015	-0.01	0.01
SW-Monsoon	0.28	0.01	0.05	0.00
Post-monsoon	0.29	-2.06*	-1.98*	-8.56
Winter	0.28	-1.14	-1.13	-0.96
Pre-monsoon	0.28	-0.99	-1.26	-0.28

Code for trend analysis

```
##Mann-Kendall Test for Trend in R
Data<-read.csv(file.choose(),header=TRUE)
install.packages("trend")
library(trend)
#####Mann-Kendall Test
mk.test(x, alternative = "two.sided")
#Magnitude of trend
#Sen's slope
sens.slope(x)
#Seasonal trend
datas<-read.csv(file.choose(),header = TRUE)
data<-ts(datas$Rainfall,start=c(1890,1),end = c(2008,12),frequency = 12)
smk.test(data)
#Test for the Randomness
```

```
##Wallis and Moore phase-frequency test
wm.test(x)
#Modified Mankendal test
install.packages("modifiedmk")
library(modifiedmk)
#Mann-Kendall Test applied to Trend Free Pre-Whitened Time Series Data in Presence of Serial Correlation Using Yue and Pion (2002) Approach
tfpwmk(x)
#Modified Mann-Kendall Test For Serially Correlated Data Using Hamed and Rao (1998) Variance Correction Approach
mmkh(x)
Change-point detection
#Pettitt's test
pettitt.test(x)
##Buishand Range Test
br.test(x)
#Buishand U Test
bu.test(x)
#Standard Normal Homogeinity Test
snh.test(x)
```

3. Moving averages and Exponential smoothing methods

3.1. Moving Average (MA)

3.1.1. Single Moving averages: Moving average is a numerical average of last N data points. In general the MA is defined as follows;

$$M_t^{[1]} = \frac{Y_t + Y_{t-1} + \cdots + Y_{t-N+1}}{N}$$

Where, Y_t is the observed time series at time t, At each successive time period the most recent observation is included and the farthest observation is excluded for computing the average. Hence the name 'moving' averages.

3.1.2: Double moving averages

The simple moving average is intended for data of constant and no trend nature. If the data have a linear or quadratic trend, the simple moving average will be misleading. In order to correct for the bias and develop an improved forecasting equation, the double moving average can be calculated. To calculate this, simply treat the single moving average time as individual data points and obtain a moving average of these averages.

3.2. Exponential Smoothing methods

3.2.1: Simple exponential smoothing (SES)

Simple exponential smoothing (Brown 1959) is best applied to time series that do not exhibit trend and do not exhibit seasonality. Its only smoothing parameter is α. The smoothing parameter α is used to control the speed which the updated forecast will adapt to local level (or mean) of the time series. This is also known as single exponential smoothing. It is used for short-range forecasting. The model assumes that the data fluctuates around a reasonably stable mean (no trend or consistent pattern of growth).

$$F_{t+1} = F_t + \alpha(Y_t - F_t)$$

F_{t+1} is the forecast of current period, based on forecast of most recent period F_t and smoothing constant α. We have to choose the smoothing constant α in such a way that the model should yield the lowest MSE value.

3.2.2: Double Exponential Smoothing (Holt)

It is best applied to time series that have a linear trend but does not exhibit seasonal behavior. The smoothing constant α is used to control speed of adaptation to local level and a second smoothing constant β is introduced to control degree of local trend carried through to multi-step-ahead forecast periods. Holt's model is more general than Brown's model because its smoothing parameters (level and trend) are not constrained by each other's values as in case with Brown's One-parameter Linear Trend Method. It is also known as Holt’s Exponential smoothing with Additive Trend.

3.2.3: Triple Exponential Smoothing (Winters)

The method under consideration is particularly recommended when the time series data exhibits seasonality. It is built upon three core smoothing equations—one each for the level, the trend, and the seasonal component. While it closely resembles Holt’s method, it introduces an additional equation to explicitly handle the seasonal variation. In fact, there exist two versions of Winter’s method, depending on the form of seasonality being modeled: the additive version for constant seasonal fluctuations and the multiplicative version for seasonality that varies with the level of the series.

4. Stationarity process of TS:

A fundamental aspect of time series analysis is to assess whether the data is stationary. A time series is considered stationary when its statistical properties—such as the mean, variance, and auto-covariance at various lags—remain constant over time, regardless of the specific time point under consideration (Fig. 1). Stationarity ensures that the model's parameters remain stable and reliable for forecasting.

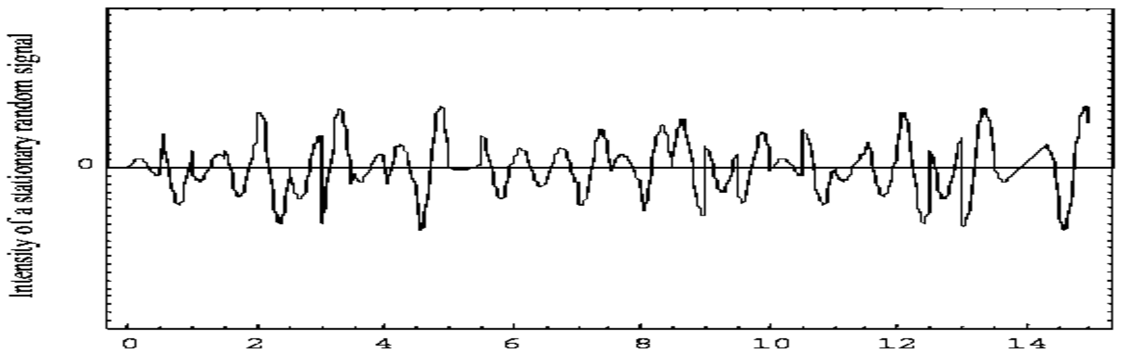


Figure 1: Sample path of a stationary process

Moreover, the time series {rt} is said to be strictly stationary if the joint distribution of rt1 ,..., rtk is identical to that of rt1-s ,..., rtk-s for all choice of t1, t2,., tk and all choice of time lag s. In other words, strict stationarity requires that the joint distribution of rt1 ,..., rtk is constant under time shift. A weaker version of stationarity is often assumed. A time series {rt} is weakly stationary if both the mean of rt and the covariance between rt and rt−s are time-invariant, where s is an arbitrary integer. More specifically, {rt} is weakly stationary if:

1) E(rt) = μ, which is a constant , for all t .
2) Cov(rt , rt−s) = γs, which only depends on all time t and lag s .

A time series is said to exhibits non-stationarity if the underlying generating process does not have a constant mean and/or a constant variance. As an example, the series given below displays considerable variation, especially since 2001, and a stationary model does not seem to be reasonable (Fig. 2).

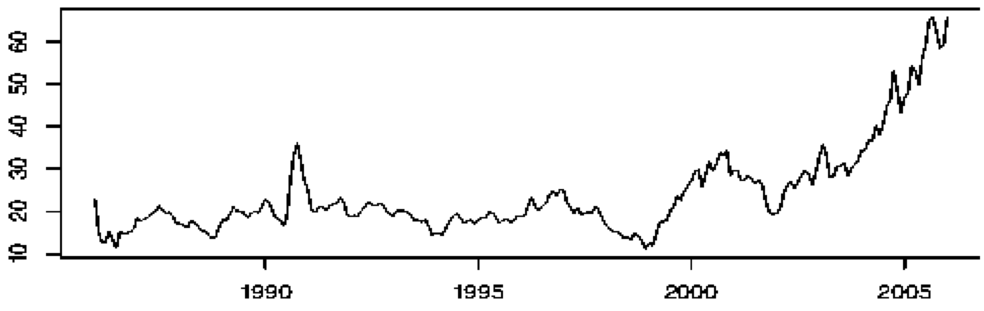


Figure 2: Sample path of a non-stationary process

A statistical test for stationarity is the most widely used Dickey Fuller test. To carry out the test, estimate by OLS the regression model. If the AR parameter is nearly zero the original series needs differencing and if AR parameter is less than zero then series is said to be already stationary.

5. **Autocorrelation Function (ACF)**

One of the most essential tools for analyzing dependence in a time series is the sample autocorrelation function. The correlation coefficient between two random variables, X and Y, indicates the strength of their linear relationship and always lies between -1 and 1. When a time series is assumed to be stationary, the autocorrelation function pk for various lags k=1,2,…can be estimated by calculating the sample correlation between observations that are k time units apart. Specifically, the correlation between Y_{t-k} and Y_{t-k} is referred to as the lag-k autocorrelation or serial correlation coefficient, and under the assumption of weak stationarity, it is defined as follows:

$$\rho_k = \frac{\sum_{t-k+1}^{T}(Y_t - \overline{Y})(Y_{t-k} - \overline{Y})}{\sum_{t-1}^{T}(Y_t - \overline{Y})^2} = \frac{\gamma_k}{\gamma_0}\ ;\ for\ k = 1,2,......where, \gamma_k = \operatorname{cov}(Y_t, Y_{t-k})$$

Since ρ_k is a correlation, it has the simple properties:

a) -1 ≤ ρ_k ≤ 1

b) ρ_k = ρ_{-k}

c) ρ_0 = 1

5.1. Partial Autocorrelation Function (PACF)

Partial autocorrelations are used to measure the degree of association between Y_t and Y_{t-k} when the y-effects at other time lags 1,2,3,…,k-1 are removed.

6. Autoregressive (AR) Model

An observed time series Y_t can be elucidate by linear function of its previous observation, Y_{t-1} and some unexplainable random error ε_t. Let us consider equally spaced time series $Y_{t,}$ Y_{t-1}, Y_{t-2} …, over an equal period of time say *t, t-1, t-2, …,* then Y_t can be defined as;

$$Y_t = \emptyset_1 Y_{t-1} + \emptyset_2 Y_{t-2} + \cdots + \emptyset_p Y_{t-p} + \varepsilon_t$$

If we represent the series in Backshift operator format, then it becomes

$$\emptyset(B) = 1 - \emptyset_1(B) - \emptyset_2 B^2 - \cdots - \emptyset_p B^p$$

Where, *B* is the backshift $BY_t = Y_{t-1}$ then the AR model can be written as $\emptyset(B)Y_t = \varepsilon_t$.

6.1. Moving Average (MA) Model

Another important model of great practical utility in the frame work of time series is finite moving average model. The MA (*q*) model is defined as;

$$Y_t = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$$

In terms of backshift operator, the MA model of order q is given as follows;

$$\theta(B) = 1 - \theta_1(B) - \theta_2 B^2 - \cdots - \theta_q B^q$$

Where *B* is the backshift operator and the moving average model can be expresses as;

$$Y_t = \theta(B)\varepsilon_t$$

6.2. Autoregressive Moving Average (ARMA) model

In order to obtain the higher efficiency and greater flexibility in modeling we combine both autoregressive and moving average processes together. These models are called as "mixed models" and are represented as ARMA *(p,q)* models

$$Y_t = \emptyset_1 Y_{t-1} + \emptyset_2 Y_{t-2} + \cdots + \emptyset_p Y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$$

Generally, in Backshift operator it is expressed as follows;

$$\emptyset(B)Y_t = \theta(B)\varepsilon_t$$

6.3. Autoregressive Integrated Moving Average (ARIMA) model

ARIMA is one of the most established methods for analyzing non-stationary time series. Unlike regression models, ARIMA explains a time series using its own past (lagged) values and random error terms. These models are often referred to as mixed models because they combine both autoregressive (AR) and moving average (MA) components. Although mixed models can make the forecasting process more complex, they generally offer more accurate predictions. In contrast, pure models consist solely of either AR or MA components, but not both. The integrated (I) part of ARIMA refers to the differencing process used to convert a non-stationary series into a stationary one, enabling forecasting. An ARIMA model is typically denoted as ARIMA(p, d, q) and is expressed in the following form:

$$\emptyset(B)(1-B)^d Y_t = \theta(B)\varepsilon_t$$

$$Y_t = \emptyset_1 Y_{t-1} + \emptyset_2 Y_{t-2} + \cdots + \emptyset_p Y_{t-p} + \varepsilon_t - \theta_1\varepsilon_{t-1} - \theta_2\varepsilon_{t-2} - \cdots - \theta_q\varepsilon_{t-q}$$

Y_t is the time series, $\emptyset_i$ and θ_j are model parameters, ε_t is random error, p is number of autoregressive terms, q is number of moving terms and B is the backshift operator such that, $BY_t = Y_{t-1}$ (Box and Jenkins 1994, Brockwell and Davis 1996).

The main stages in setting up a Box-Jenkins forecasting models are described below:

6.3.1. Identification:

The initial and most crucial step in time series modeling is to check whether the series is stationary, as most estimation techniques are valid only for stationary data. If the series is found to be non-stationary, it must first be transformed into a stationary form. Once stationarity is achieved, the next step is to identify initial estimates for the orders of seasonal and non-seasonal parameters—namely, p, q for non-seasonal and P, Q for seasonal components. These initial values can be suggested by examining the significance of autocorrelation and partial autocorrelation coefficients. For instance, if the second-order autocorrelation is significant, an AR(2), MA(2), or ARMA(2) model might be considered as a starting point. However, this is not a strict rule, since sample autocorrelations can be unreliable approximations of their population counterparts. Despite this, they serve as useful starting estimates, with the final model determined through an iterative process. The estimated ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) provide a straightforward way to explore statistical relationships within the data, helping to identify underlying patterns and dependencies. This relationship is captured mathematically through an equation. The fundamental idea behind this technique is that every stochastic process operating over time has its own characteristic Autocorrelation Function (ACF) and Partial Autocorrelation Function

(PACF). Since any observed time series is considered a specific realization of an underlying stochastic process, its theoretical ACF and PACF should closely resemble the estimated ACF and PACF derived from the actual data.

Table 2: Pattern of ACF and PACF for AR, MA and ARMA processes

Process	ACF	PACF
AR	Decays Towards zero	Cut Off to zero (lag length of last spike is the order of the process)
MA	Cut Off to zero (lag length of last spike is the order of the process)	Decays towards zero
ARMA	Tails off towards zero	Tails off towards zero

6.3.2. Estimation of parameters

During the estimation stage of time series modeling, the parameters of the identified model are computed. Typically, the method of least squares is employed, which minimizes the sum of squared residuals to determine the best-fitting coefficients. At this stage, it is also essential to verify the stationarity and invertibility of the model based on the estimated parameters. Additionally, model adequacy is assessed to ensure that the selected model fits the data well. The significance of each estimated coefficient is evaluated through statistical testing, as each has a sampling distribution and an associated standard error. Most ARIMA estimation procedures include automatic hypothesis testing to determine whether a coefficient is significantly different from zero. However, when coefficients are highly correlated, the resulting parameter estimates may be unreliable. To assess the quality of the model fit, performance metrics such as Root Mean Square Error (RMSE), Mean Absolute Percentage Error (MAPE), and others are calculated.

6.3.3. Diagnostic checking

Different models can be obtained for various combinations of AR and MA individually and collectively. The best model is obtained with following diagnostics.

(a) Low Akaike Information Criteria (AIC)/ Bayesian Information Criteria (BIC)/Schwarz-Bayesian Information Criteria (SBC)

AIC is given by (-2 log L + 2 m) where m=p+ q+ P+ Q and L is the likelihood function. Since -2 log L is approximately equal to $\{n\,(1+\log 2\pi) + n \log \sigma^2\}$ where σ^2 is the model MSE, Thus AIC can be written as AIC={n (1+log 2π) + n log σ^2 + 2 m} and because first term in this equation is a constant, it is usually omitted while comparing between models. As an alternative to AIC, sometimes SBC is also used which is given by SBC = log σ^2 + (m log n)/n.

(b) Plot of residual ACF

After fitting a suitable ARIMA model, the goodness of fit can be assessed by examining the autocorrelation function (ACF) of the model's residuals. If the majority of the residual autocorrelation coefficients fall within the range of ±1.96/√N—where N is the total number of observations used in the model—it suggests that the residuals behave like white noise. This indicates that the model has effectively captured the structure of the data and is considered a good fit.

(c) Non-significance of auto correlations of residuals via Portmonteau tests (Q-tests based on Chi-square statistics)-Box-Pierce or Ljung-Box texts

Once a tentative model has been applied to the data, it is essential to carry out diagnostic checks to evaluate the model's adequacy and identify any areas for potential improvement. A common approach to this is by analyzing the residuals of the model. One effective method for assessing the overall fit is through the use of the Box-Pierce statistic (Q), which is based on the autocorrelations of the residuals. This statistic approximately follows a Chi-square distribution and provides a quantitative measure to determine how well the model captures the underlying structure of the data. It is calculated using the following formula:

$$Q=n \sum r^2_{(j)}$$

In this context, the summation runs from 1 to k, where k is the maximum lag considered—typically chosen around 20—and n is the number of observations in the time series. The term r(j) represents the estimated autocorrelation at lag j. The Box-Pierce Q statistic follows a Chi-square distribution with (k − m − 1) degrees of freedom, where m is the number of parameters estimated in the model. A refined version of this test is the Ljung–Box statistic, calculated as:

$$q= n\ (n+2) \sum r^2_{(j)}/\ (n-j)$$

This Q statistic is then compared with critical values from the Chi-square distribution. If the model is correctly specified, the residuals should exhibit no autocorrelation, resulting in a small Q value and a large associated p-value. Conversely, a significant Q value suggests that the residuals are not white noise, indicating that the model may not adequately fit the data.

7. Forecasting

The final model is used to generate predictions about the future values and then calculate the errors for the values obtained by developed model.

8. Illustration: To build the ARIMA Model

Yearly data on total oilseed production (in million tonnes) in India from 1950–51 to 2015–16 were obtained from the agricultural statistics published by the Reserve Bank of India (RBI), Government of India (RBI Statistics, 2016). Data from 1950–51 to 2010–11 were used for model development, while the data from 2011–12 to 2015–16 were utilized for validating the forecasting performance of the models. The summary statistics and time series plot corresponding to the dataset are presented in Table 2 and Figure 3, respectively.

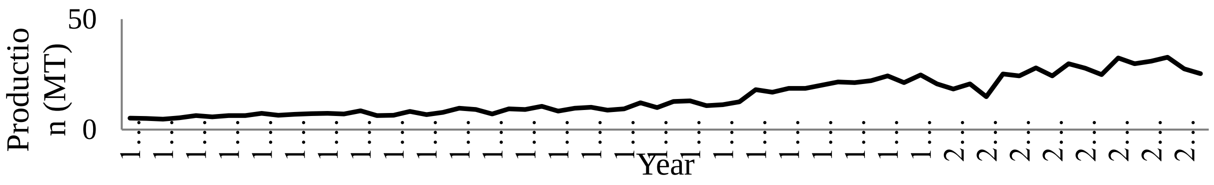
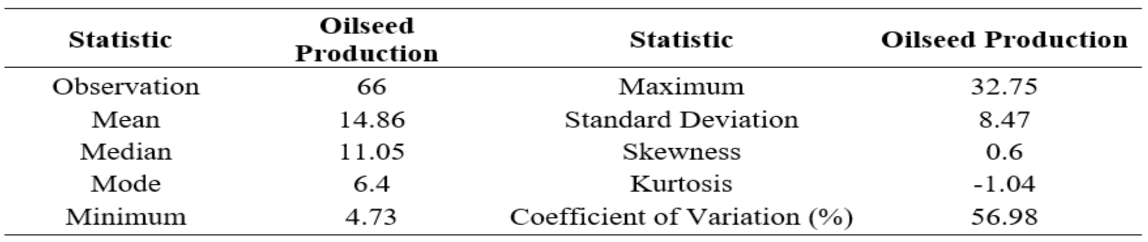


Fig.3: Time series plot of Oilseed production of India

Table 3: Summary statistics of Oilseed Production time series



Statistic	Oilseed Production	Statistic	Oilseed Production
Observation	66	Maximum	32.75
Mean	14.86	Standard Deviation	8.47
Median	11.05	Skewness	0.6
Mode	6.4	Kurtosis	-1.04
Minimum	4.73	Coefficient of Variation (%)	56.98

The ARIMA model has been built for oilseed production of India. The original time series was found to be non-stationary, so first differencing was done to make the stationary series time series (Figure 4).

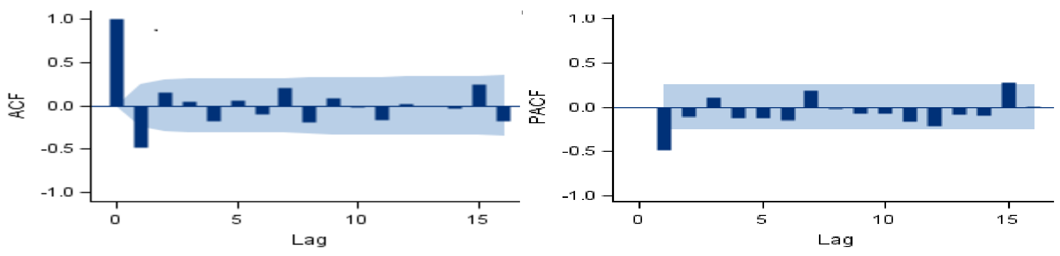


Fig. 4. ACF and PACF time series Oilseed production of India

The selected model, ARIMA(1,1,0), was determined to be appropriate based on the analysis of Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots (Figure 3). A residual autocorrelation check for the fitted ARIMA model applied to the mango production time series revealed that the residuals were non-autocorrelated, as indicated by a Chi-square test p-value of 0.45, confirming the adequacy of the model.

Table 4: Parameter estimation of ARIMA (1, 1, 0) by Maximum Likelihood Estimation method for Oilseed Production time series

Parameter	Estimate	Standard Error	t Value	Approx. Pr > |t|	Lag
MU	0.43	0.19	1.64	0.1012	0
AR1,1	-0.56	0.18	-4.39	<0.0001	1

Table 5: Model performance of Oilseed Production time series for training data set

Criteria	ARIMA
MSE	5.33
RMSE	2.32
MAPE	11.64

Table 6: Forecasted value of Oilseed Production time series using ARIMA Model

Year	Actual	Forecast
		ARIMA
2011	29.80	28.84
2012	30.94	31.54
2013	32.75	30.72
2014	27.51	31.88
2015	25.30	31.90
Criteria	MSE	13.31
	RMSE	3.64
	MAPE	10.58

R code for ARIMA model

```
library(forecast)
library(tseries)
ye=read.table(file="clipboard",header=TRUE) ## import the data
```

```
ye.ts <- ts(ye, start=1890, end=2008, frequency=1) ## define into time series frame work
plot.ts(ye.ts, col='blue', pch=2, lwd=3) ##plot the series
acf(ye.ts,lag.max = 40,col='red', pch=2, lwd=3) ## plot acf and pacf
pacf(ye.ts,lag.max = 40,col='red', pch=2, lwd=3)
stationarity=adf.test(ye.ts) ## check for stationarity
stationarity
ye.training=ye.ts[1:100]  ## devide the data into training and testing set
ye.testing=ye.ts[101:119]
arima1=arima(ye.training, order=c(1,0,1),include.mean = TRUE) ## ARIMA fitting
arima1
arima1=arima(ye.training, order=c(1,0,1),include.mean = TRUE) ##
## install the package forecast
ye.fit=auto.arima(ye.training) ## auto fitting
ye.fit
res=arima1$residuals
res_test=Box.test(res, lag = 1, type = c("Box-Pierce", "Ljung-Box"), fitdf = 0) ##diangnostic
checking
res_test
accuracy(arima1)
fcast=forecast(arima1, h=19)
fcast
fitted.test1=data.frame(fcast) ##forecast the out of sample
fitted.test=fitted.test1[,1]
DMwR::regr.eval(ye.testing, fitted.test)
```

9. Suggested Readings

Box, G.E.P., Jenkins, G.M. and Reinsel, G.C. (1994). Time series analysis: Forecasting and control, Pearson Education, Delhi.

Makridakis, S., Wheelwright, S.C. and Hyndman, R.J. (1998). Forecasting: Methods and Applications, John Wiley, New York.

Pankratz, A. (1983). Forecasting with univariate Box – Jenkins models: concepts and cases, New york: John Wiley & Sins.

Chris Chatfield, C. (2003). The Analysis of Time Series: An Introduction, Sixth Edition. Chapman & Hall/CRC Texts in Statistical Science.

ARCH Family of Models

Achal Lama[1], K N Singh[1] and Bishal Gurung[2]*

[1]ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012
[2]North-Eastern Hill University, Shillong-793022
Email: achallama.iasri@icar.org.in

1. Introduction

Time series analysis involves studying observations that are recorded sequentially over time, where the temporal order is essential, making time series data inherently dependent. These observations can be gathered at various frequencies such as hourly, daily, weekly, monthly, or annually, depending on the nature of the application. A time series is typically represented using notations like {Xt} or {Yt}, where t = 1, 2, ..., T represents the time index for a series of length T. In statistical terms, a time series is treated as a realization from an underlying stochastic process, and efforts are directed toward understanding the probabilistic law governing such processes. This understanding facilitates insight into the dynamics of the series, enables forecasting of future values, and helps guide interventions aimed at influencing future behavior.

Given the finite nature of observed data, multiple stochastic processes can theoretically explain the same dataset. However, only a subset of these processes are both statistically plausible and meaningfully interpretable. Therefore, to make inference tractable, one typically imposes structural assumptions by selecting a suitable family of probability models. This step is called modelling, while the selection of the most appropriate model within that family and estimation of its parameters is referred to as statistical inference. When the model structure is fully specified except for a finite set of parameters, it is termed a parametric model. In contrast, nonparametric models allow for greater flexibility, where the model form may not be completely specified or may involve parameters from an infinite-dimensional space.

Effective time series analysis hinges on selecting appropriate statistical models that balance interpretability, simplicity, and feasibility. A good model should adequately reflect the underlying physical law governing the data without being overly complex. The chosen model family should be broad enough to include the true data-generating process but not so broad that parameter estimation becomes unreliable. During the modelling process, it is important first to identify key features and patterns in the data and then select a model that captures those characteristics. Once parameters are estimated, the adequacy of the model fit must be verified, and refinements may be made if necessary. It is also important to note that model suitability depends heavily on the goal of the analysis—for example, a model that provides a good fit and interpretation may not necessarily be ideal for forecasting purposes.

This write-up aims to provide a foundational understanding of time series analysis. It offers an overview of both linear and nonlinear time series models, with a focus on those within the ARMA framework. It also covers commonly used parametric nonlinear models such as the

Autoregressive Conditional Heteroscedastic (ARCH) model and its generalized form, GARCH. For a more comprehensive treatment of these models, readers may refer to Fan and Yao (2003). Finally, this document includes R code examples using real datasets to demonstrate the application and interpretation of both linear and nonlinear time series models for improved understanding and practical use.

2. Linear Time Series Models

The most widely used class of linear time series models is the Autoregressive Moving Average (ARMA) family, which encompasses both the purely Autoregressive (AR) and Moving Average (MA) models as special cases. These models are extensively applied to represent linear dynamic behavior in time series data, as they effectively capture the linear relationships among lagged observations. ARMA models provide a robust framework for analyzing and forecasting time-dependent phenomena by incorporating past values and past errors. A particularly important extension of ARMA models is the Autoregressive Integrated Moving Average (ARIMA) model. ARIMA models generalize ARMA processes to handle non-stationary data by incorporating a differencing component, thereby making them especially useful in practical applications where time series often exhibit trends or other forms of non-stationarity. The ARIMA class includes stationary ARMA processes as a subset and is widely adopted for its versatility and forecasting power. We have tried to briefly introduce these linear models in the subsequent sub-sections.

2.1 Autoregressive (AR) Model

A stochastic model that can be extremely useful in the representation of certain practically occurring series is the autoregressive model. In this model, current value of the process is expressed as a finite, linear aggregate of previous values of the process and a shock ε_t. Let us denote the values of a process at equally spaced time epochs $t, t-1, t-2, \ldots$ by $y_t, y_{t-1}, y_{t-2}, \ldots$ then y_t can be described as

$$y_t = \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p} + \varepsilon_t$$

If we define an autoregressive operator of order p by

$$\varphi(B) = 1 - \varphi_1 B - \varphi_2 B^2 - \cdots - \varphi_p B^p$$

where B is the backshift operator such that $By_t = y_{t-1}$, autoregressive model can be written as

$\varphi(B)\, y_t = \varepsilon_t$.

2.2 Moving Average (MA) Model

Another kind of model of great practical importance in the representation of observed time-series is finite moving average process. MA (q) model is defined as

$$y_t = \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots - \theta_q \varepsilon_{t-q}$$

If we define a moving average operator of order q by

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q$$

where B is the backshift operator such that $By_t = y_{t-1}$, moving average model can be written as $y_t = \theta(B)\varepsilon_t$.

2.3 Autoregressive Moving Average (ARMA) Model

To achieve greater flexibility in fitting of actual time-series data, it is sometimes advantageous to include both autoregressive and moving average processes. This leads to mixed autoregressive-moving average model

$$y_t = \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdots + \varphi_p y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \cdots \theta_q \varepsilon_{t-q}$$

or

$$\varphi(B)\, y_t = \theta(B)\varepsilon_t$$

and is written as ARMA(p, q). In practice, it is quite often adequate representation of actually occurring stationary time-series can be obtained with autoregressive, moving average, or mixed models, in which p and q are not greater than 2.

2.4 Autoregressive Integrated Moving Average (ARIMA) Model

A generalization of ARMA models which incorporates a wide class of non-stationary time-series is obtained by introducing the differencing into the model. The simplest example of a non-stationary process which reduces to a stationary one after differencing is Random Walk. A process $\{ y_t \}$ is said to follow an Integrated ARMA model, denoted by ARIMA (p, d, q), if $\nabla^d y_t = (1 - B)^d \varepsilon_t$ is ARMA (p, q). The model is written as

$$\varphi(B)(1-B)^d y_t = \theta(B)\varepsilon_t$$

ε_t are assumed to be independently and identically distributed with a mean zero and a constant variance of σ^2 .

3. Non-linear models: ARCH and GARCH models

After the dominance of the ARIMA model for over two decades, the need of such model was felt which could predict with varying variance of the error term. The solution was provided by Engle (1982) when he developed ARCH model to estimate the mean and variance of the United Kingdom inflation. This model has few interesting characteristics; it models the conditional variance as the square of the function of the previous error term and assumes the unconditional variance to be constant. Along with the ARCH models can model heavy tail data which are common in financial market. Besides these, Bera and Higgins (1993) pointed out that ARCH models are easy and simple to handle, can take care of clustered errors, non-linearity and importantly takes care of changes in the econometrician's ability to forecast.

The ARCH (q) model for the series $\{\varepsilon_t\}$ is defined by specifying the conditional distribution of given the information available up to time $t-1$. Let denote this information. ARCH (q) model for the series is given by

$$\varepsilon_t | \psi_{t-1} \sim N(0, h_t)$$

$$h_t = a_0 + \sum_{i=1}^{q} a_i \varepsilon_{t-i}^2$$

where, $a_0 > 0$, $a_i \geq 0$, for all i and $\sum_{i=1}^{q} a_i < 1$ are required to be satisfied to ensure non-negativity and finite unconditional variance of stationary $\{\varepsilon_t\}$ series. Bollerslev (1986) and Taylor (1986) proposed the Generalized ARCH (GARCH) model independently of each other, in which conditional variance is also a linear function of its own lags and has the following form

$$\varepsilon_t = \xi_t h_t^{1/2} \qquad (1)$$

where ξ_t ~ N (0,1). A sufficient condition for the conditional variance to be positive is

$$a_0 > 0,\ a_i \geq 0,\ i = 1,2,\ldots,q.\ \ b_j \geq 0,\ \ j = 1,2,\ldots,p$$

The GARCH (p, q) process is weakly stationary if and only if

$$.\ \sum_{i=1}^{q} a_i + \sum_{j=1}^{p} b_j < 1$$

The conditional variance defined by (1) has the property that the unconditional autocorrelation function of ε_t^2 ; if it exists, can decay slowly. For the ARCH family, the decay rate is too rapid compared to what is typically observed in financial time-series, unless the maximum lag q is long. As (1) is a more parsimonious model of the conditional variance than a high-order ARCH model, most users prefer it to the simpler ARCH alternative. The most popular GARCH model in applications is the GARCH (*1,1*) model.

Step 1: Determine whether the time series is stationary.

Before applying any time series model, it is essential to ensure that the series under analysis is stationary. A stationary time series is characterized by constant statistical properties over time, such as mean, variance, and autocorrelation structure. Stationarity implies that the underlying process generating the data does not change with time, which is a fundamental assumption for many time series models, including ARIMA.

Preliminary detection of stationarity can be done visually by plotting the raw data, as well as examining the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots. However, to statistically verify the presence of stationarity, formal tests such as the Dickey-Fuller test, Augmented Dickey-Fuller (ADF) test, Phillips-Perron test, and the KPSS test (developed by Kwiatkowski, Phillips, Schmidt, and Shin) are widely employed. These tests help determine whether differencing or other transformations are needed to make the series stationary before modeling.

Step 2: Identify the model.

Once the time series has been rendered stationary, the next step involves identifying an appropriate mean model, typically using the Autoregressive Integrated Moving Average (ARIMA) framework. The ARIMA model, denoted as ARIMA(p, d, q), is specified by determining three key parameters:

- p, the order of the autoregressive (AR) component,
- d, the order of differencing required to achieve stationarity, and
- q, the order of the moving average (MA) component.

The values of p and q are selected based on the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots of the stationary series. Specifically, the PACF plot is used to identify the order of the AR term (p), while the ACF plot helps in identifying the order of the MA term (q). The parameter d reflects the number of differencing operations applied to the original series to induce stationarity.

Step 3: Estimate the model parameters and diagnostic checking.

Once a few tentative models have been identified, the estimation of model parameters is carried out using standard statistical procedures. Typically, the Maximum Likelihood Estimation (MLE) method is employed, which estimates parameters by maximizing the likelihood function or, equivalently, by minimizing an overall measure of forecast errors. This stage primarily aims to assess whether the assumptions made regarding the error structure of the model are satisfied. To validate this, diagnostic checking is performed, most commonly using the Portmanteau test (such as the Box–Pierce or Ljung–Box test). This test examines whether the residuals from the fitted model behave like white noise—that is, they are uncorrelated and have a constant variance over time. The null hypothesis for the test states that the residuals constitute white noise, and rejecting it would indicate that the model may be inadequate and require re-specification.

The Ljung-Box statistic is given by:

$$Q = n(n+2)\sum_{k=1}^{h}(n-k)^{-1}r_k{}^2$$

where, *h* is the maximum lag, *n* is the number of observations, *k* is the number of parameters in the model. If the data are white noise, the Ljung-Box Q statistics has a chi-square distribution with *(h-k)* degrees of freedom.

Step 4: Select the most suitable ARIMA model

The most suitable ARIMA model is selected using the smallest Akaike Information Criterion (AIC) or Schwarz-Bayesian Criterion (SBC). AIC is given by

$$AIC = (-2\log L + 2m)$$

where, $m= p+q$ and L is the likelihood function. SBC is also used as an alternative to AIC which is given by

$$SBC = \log\sigma^2 + (m\log n)/n$$

If the model is not adequate, a new tentative model should be identified, which is again followed by the parameter estimation and model verification. Diagnostic information may help suggest alternative model(s). The steps of model building process are typically repeated several times until a satisfactory mean model is finally selected. The final model can then be used for prediction purposes.

Step 5: Determination of residuals and heteroscedasticity test.

After finding the mean model now the residuals are to be determined. And we create a new variable called ‘rsquare’ by squaring the residuals. Then the ACF and PACF values of the ‘rsquare’ are determined and the lags in which these values are found to be significant are identified. The test for heteroscedasticity is done at identified significant lags. The test employed is the ARCH-LM test.

Step 6: Residuals and diagnostic checking.

The residuals obtained from the mean model used for fitting the different GARCH models were squared and stored in a new variable called ‘esquare’. As already mentioned previously, the diagnostic tests are employed to check whether the residuals are white noise or not.

Step 7: Estimation of parameters.

The parameters of the obtained model are estimated using method of maximum likelihood (MLE). And then forecasting is done using the selecting model.

5. Illustration

In this illustration Cotlook A index data is used and was collected from the commodity price bulletin, published by the United Nations Convention of Trade and Development (UNCTAD). The series contains 360 data pints, 346 data points are used for modelling and remaining 14 points for forecasting. At first the ARIMA model was applied to the data set and on unsatisfactory performance of the model, the GARCH model was used.

5.1 Fitting of the Cotlook A index

Various combinations of the ARIMA models were tried, among all, the AR (1) model had minimum AIC and BIC values. The AIC value for fitted GARCH model has been found to be minimum when the mean equation depends on two recent pasts only. Investigating the autocorrelation function (Acf) of squared residuals of AR (2) model, it is found that the Acf and Pacf are maximum at lag 3, which is 0.226 and 0.221 respectively. But if we go for AR (2)-ARCH (3) model, a large number of parameters are needed to be estimated. So, to get a parsimonious model, the AR (2)-GARCH (1, 1) model is selected.

The mean and conditional variance for fitted AR (2)-GARCH (1, 1) model is computed as follows:

$$y_t = 141.9264 \;\; -1.3905 \;\; y_{t-1} + 0.4538\, y_{t-2} + \varepsilon_t$$

$$(3.94) \quad (0.05) \qquad\qquad (0.05)$$

where

$$\varepsilon_t = h_t^{1/2}\xi_t\,,$$

and h_t satisfies the variance equation

$$h_t = 8.470 + 0.208\;\varepsilon_{t-1}^2 + 0.215\, h_{t-1}$$

(1.97) (0.09) (0.079)

The values within brackets denote corresponding standard errors of the estimates. The AIC value, for fitted GARCH model is 2288.88.

Table 1. Forecast of the Cotlook A index series

MONTH	ACTUAL VALUE	FORECAST ARIMA(1,1,0)	FORECAST AR(2)-GARCH(1,1)
Feb-11	469.98	408.34(8.30)	389.59(26.46)
Mar-11	506.34	416.47(15.56)	371.55(25.74)
Apr-11	477.56	421.40(22.35)	348.54(25.05)
May-11	364.91	424.53(28.55)	324.69(24.39)
Jun-11	317.75	426.66(34.17)	301.98(23.75)
Jul-11	268.96	428.23(39.29)	281.25(23.13)
Aug-11	251.55	429.49(43.97)	262.76(22.54)
Sep-11	257.63	430.57(48.29)	246.50(21.97)
Oct-11	243.85	431.55(52.30)	232.32(21.42)
Nov-11	230.78	432.48(56.05)	220.01(20.90)
Dec-11	210.43	433.37(59.58)	209.35(20.39)
Jan-12	222.91	434.25(54.45)	200.15(19.91)
Feb-12	222.12	435.12(57.13)	192.21(19.44)
Mar-12	219.36	435.99(59.68)	185.37(19.01)

Table 2. Forecast evaluation of the Cotlook A index series

MODEL	RMSE	RMAPE (%)
ARIMA(1,1,0)	44.03	60.72
AR(2)-GARCH(1,1)	15.38	9.36

6. R code for analysing a time series data using ARCH family of models

```
library(“tseries”)
library(“forecast”)
library(“fgarch”)
setwd("C:/Users/Achal/Desktop") # Setting of the work directory
data<-read.table("bishal.txt") # Importing data
datats<-ts(data,frequency=12,start=c(1982,4)) # Converting data set into time series
plot.ts(datats) # Plot of the data set
```

```
adf.test(datats) # Test for stationarity
diffdatats<-diff(datats,differences=1) # Differencing the series
datatsacf<-acf(datats,lag.max=12) # Obtaining the ACF plot
datapacf<-pacf(datats,lag.max=12) # Obtaining the PACF plot
auto.arima(diffdatats) # Finding the order of ARIMA model
datatsarima<-arima(diffdatats,order=c(1,0,1),include.mean=TRUE) # Fitting of ARIMA model
forearimadatats<-forecast.Arima(datatsarima,h=12) # Forecasting using ARIMA model
plot.forecast(forearimadatats) # Plot of the forecast
residualarima<-resid(datatsarima) # Obtaining residuals
archTest(residualarima,lag=12) # Test for heteroscedascity
# Fitting of AR-GARCH model
garchdatats<-garchFit(formula = ~ arma(2)+garch(1, 1), data = datats, cond.dist = c("norm"), include.mean = TRUE, include.delta = NULL, include.skew = NULL, include.shape = NULL, leverage = NULL, trace = TRUE,algorithm = c("nlminb"))
# Forecasting using AR-GARCH model
forecastgarch<-predict(garchdatats, n.ahead = 12, trace = FALSE, mse = c("uncond"), plot=FALSE, nx=NULL, crit_val=NULL, conf=NULL)
plot.ts(forecastgarch) # Plot of the forecast
```

Bibliography:

Bera, A. K., and Higgins, M. L. (1993), ARCH Models: Properties, Estimation and Testing, *Journal of Economic Survey*, **7**, 307-366.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, **31**, 307-327.

Box, G. E. P., Jenkins, G. M. and Reinsel, G. C. (2007). Time-Series Analysis:

Engle, R.F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of U.K. inflation. *Econometrica*, **50**, 987-1008.

Fan, J. and Yao, Q. (2003). *Nonlinear time series:nonparametric and parametric methods*. Springer, U.S.A.

Forecasting and Control. 3rd edition. *Pearson education*, India.

Lama, A., Jha, G.K., Gurung, B., Paul,R.K. and Sinha, K. (2016). VAR-MGARCH Models for Volatility Modelling of Pulses Prices: An Application. *Journal of the Indian Society of Agricultural Statistics*, **70**, 145-151.

Sims, C. (1980). Macroeconomics and reality. *Econometrica*, **48**, 1-48.

Taylor, S. J. (1986). Modeling financial time series. Wiley, New York.

Introduction to Bayesian Time Series Analysis

Achal Lama[1*], K N Singh[1] and Bishal Gurung[2]

[1]ICAR-Indian Agricultural Statistics Research Institute, New Delhi-110012
[2]North-Eastern Hill University, Shillong-793022
Email: achallama.iasri@icar.org.in

1. Introduction

Bayesian estimation and inference offer several advantages in statistical modeling, particularly when incorporating prior knowledge is essential. At the heart of the Bayesian paradigm lies the specification of prior distributions, which reflect the analyst's belief or available information before observing the current data. The fundamental assumption in Bayesian analysis is that the data alone may not capture the entire underlying behavior of the process. Thus, prior information is formally combined with data through Bayes' theorem, producing a posterior distribution of the model parameters. Let us consider the parametric space θ denote the vector of model parameters, denote the vector of model parameters, $\pi(\theta)$ and **Y** is the data vector. According to Bayes' rule, the posterior density

$$\pi(\theta|y) \propto L(Y|\theta)\pi(\theta)$$

where, $L(Y|\theta)$ is the likelihood function. The straightforward way to estimate θ is to compute the posterior mean of θ as follows:

$$\hat{\theta} = \int \theta\pi(\theta|y)d\theta$$

One of the strengths of the Bayesian framework is that it provides full probability distributions for parameters, as opposed to point or interval estimates in classical (frequentist) approaches. This is particularly beneficial in fields like finance, where rapid information flow justifies the use of prior knowledge.

Bayesian modeling allows for various types of priors. Non-informative priors are used when little is known a priori, while conjugate priors simplify calculations since the posterior belongs to the same family as the prior. For example, when the likelihood is from the exponential family, deriving conjugate priors becomes more tractable (Lee, 2004). Additionally, conjugate priors facilitate updating in light of new data by modifying hyperparameters instead of the entire distribution.

Given the computational complexity involved, especially in high-dimensional settings, Markov Chain Monte Carlo (MCMC) methods are widely used for Bayesian estimation. MCMC methods possess two vital characteristics:

The first advantage is its capability to handle high-dimensional problems efficiently, and the second is its ability to draw random samples directly from the posterior distribution. To illustrate the process, consider a scenario where information is desired about a distribution, known only up to a constant CCC, under the assumption that the state space EEE is either finite or countable. In such a case, the distribution can be expressed with a probability mass function proportional to the available information. The primary objective of employing the Markov Chain Monte Carlo (MCMC) method is to obtain samples from this posterior distribution.

$$\pi^*\left(\theta|y\right)=\frac{f\left(y|\theta\right)p(\theta)}{\sum_{\theta\in E} f\left(y|\theta\right)p(\theta)}$$

To obtain the posterior distribution, the following steps are undertaken. First, an ergodic Markov Chain is constructed, which converges to a stationary posterior distribution. Next, the Markov Chain is used to simulate values over a large number of iterations, denoted by *l*+*k*. The *l*-1 samples are discarded to ensure convergence to the stationary distribution, and the remaining samples are used for analysis. From these *l*+*k* samples, the expectation and other summary statistics are computed, as these represent stationary values. The expectation of the posterior distribution is particularly significant because it is used to estimate the parameters of the model under study.

$$E_{\pi^*(\theta)}=\sum_{\theta\in E}\theta\pi^*\left(\theta|y\right)$$

$$=\frac{\sum_{\theta\in E}\theta f\left(y|\theta\right)p(\theta)}{\sum_{\theta\in E} f\left(y|\theta\right)p(\theta)}$$

But, if the posterior distribution is high dimensional or else complicated, it is difficult to obtain closed form solutions for *C*. The answer to this is the MCMC method. The two very widely used MCMC algorithms are Metropolis-Hastings (MH) algorithm and Gibbs sampling. Gibbs sampling is considered to be a special sampler of MH algorithm.

2. Sampling Algorithms

2.1. Metropolis-Hastings algorithm

The MH algorithm is a popular algorithm which is used to obtain a sequence of random samples from a proposed distribution $q(\theta,\zeta)$ where direct sampling is difficult. The algorithm is first proposed by Metropolis *et al.*, (1953) and extended by Hastings (1970). The idea of MH is simple, in this method a proposal point ξ is generated from the proposal distribution $q(\theta,.)$ with an acceptance probability as

$\alpha(\theta,\xi) = \min\{1, r(\theta,\xi)\}$,where

$$r(\theta,\xi) = \frac{\pi(\xi)q(\theta,\xi)}{\pi(\theta)q(\theta,\xi)}$$

This process can be thought of as generating a random number X from a uniform distribution U [0,1] and accepting the state ξ if $X < \alpha(\theta,\xi)$, otherwise the point θ is rejected and the algorithm remains in the same state. The quantity $r(\theta,\xi)$ is known as the MH ratio and hence the algorithm as MH algorithm. This algorithm can be summarized in following steps

1. A proposal distribution is selected with transition matrix $Q=(q(I,j))_{I,j \in E}$. Select an integer s between 1 and *n*.
2. Assign *n*=0 and θ_0=*s*.
3. A random variable θ is generated such that $P(\theta = j) = q(\theta_n, j)$ and X is generated independently.
4. If $X < \alpha(s,j)$, then θ=*j*, otherwise θ = *s*.
5. Next *n* is set as *n*=*n*+*1* and $\theta_n = \theta$
6. Go to step 3.

Random walk algorithm is considered a special case of the Metropolis-Hastings (MH) algorithm, where the proposal distribution exhibits symmetry. However, its performance significantly deteriorates in high-dimensional models, as increasing dimensionality tends to reduce the acceptance rate of proposed samples.

2.2. Gibbs sampling

Gibbs sampling, on the other hand, is another popular MCMC algorithm named after Josiah Willard Gibbs, though it was introduced by Geman and Geman in 1984. It is known for its simplicity, ease of implementation, and effectiveness in addressing high-dimensional problems. While conjugate priors are often useful in Bayesian analysis, constructing a joint conjugate prior for multiple parameters can be challenging. In such cases, conditional conjugate priors are relatively easier to define. Gibbs sampling leverages these conditional priors to transform a complex multidimensional sampling problem into a series of simpler one-dimensional problems. The key advantage is that the conditional conjugate prior maintains the same form as the posterior distribution. Assuming a data set $y = (y_1, y_2, \ldots, y_n)$ where each y_i is associated with v parameters, for each j=1,2,..., y_i = 1, 2, ..., v, a one-dimensional conjugate prior is specified and the corresponding conditional posterior is derived using Bayes' theorem. The Gibbs sampling then proceeds iteratively through a sequence of these conditional updates.

1. The initial parameter vector $\left(\theta^0{}_1, \ldots, \theta^0{}_v\right)$ is defined.
2. Parameter vector is updated by sampling as follows:

$$\theta^1{}_1 \sim p\left(\theta_1 \middle| \theta^0{}_2, \ldots, \theta^0{}_v, y\right)$$

$$\theta^1{}_2 \sim p\left(\theta_2 \middle| \theta^1{}_1, \theta^0{}_3, \ldots, \theta^0{}_v, y\right)$$

.

.

.

$$\theta^1{}_v \sim p\left(\theta_v \middle| \theta^1{}_1, \theta^1{}_2, \ldots, \theta^1{}_{v-1}, y\right)$$

3. Using this updated values as starting parameter values the sampling is repeated M times. M is a constant which is selected to be sufficiently large and referred to as burn-in period.
4. After simulating $\{\theta^{(M+1)}, \theta^{(M+2)}, \ldots, \theta^{(M+n)}\}$ from the Gibbs sampling Bayesian inferences are drawn.

The main drawback of this method is that it is infeasible to apply when complete conditional distribution is not known.

3. Bayesian time series models

The Bayesian framework has been widely extended to various time series models. In this context, the focus is primarily on MGARCH and VAR models, both of which are multivariate and are extensively applied in macroeconomic analysis. These models are capable of capturing dynamic relationships among multiple time series variables. Let Y_t represent a $(k\times1)$ vector of time series variables. The standard form of the p-lag vector autoregressive model, denoted as VAR(p), is given as:

$$Y_t = A + B_1 Y_{t-1} + B_2 Y_{t-2} + B_3 Y_{t-3} + \ldots + B_p Y_{t-p} + \varepsilon_t$$

where, A is $k \times 1$ vector of intercepts , B_i (i =1, 2, …, p) is $k \times k$ matrices of parameters and

$$\varepsilon_t \sim iidN(0, \Sigma)$$

For a multivariate time series $y_t = (y_{1t}, \ldots, y_{kn})'$ the MGARCH model is given by:

$$y_t = H_t^{1/2} \varepsilon_t$$

Where $H_t^{1/2}$ is a k x k positive-definite matrix representing the conditional variance-covariance of εt . Here, k denotes the number of series and t =1,2,…,n indicates the number of observations. The formulation of the MGARCH model mainly depends on how the conditional variance is specified. Engle and Kroner (1995) proposed the BEKK model, which is a direct extension of the univariate GARCH model to a multivariate setting. In this framework, the conditional variance evolves based on the current and past information available in the system. A general GARCH(p, q) model, as introduced by Bollerslev (1986), can be expressed as:

$$h_t = \alpha_0 + \alpha_1 \varepsilon^2_{t-1} + \cdots + \alpha_p \varepsilon^2_{t-p} + \beta_1 h_{t-1} + \cdots + \beta_q h_{t-q}\,, \quad \alpha_i > 0, \beta_i > 0\,, \quad \alpha_i + \beta_i < 1$$

where, h_t is the conditional variances which depends on the previous error terms as well as previous conditional variances of the process.

Equation (2) can be transferred into multivariate GARCH model with a generalization of the resulting variance matrix H_t

$$H_t = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

Each element of H_t depends on the p delayed values of the squared ε_t , the cross product of ε_t and on the q delayed values of elements from H_t . In general, multivariate GARCH (1, 1) model can be written as:

$$H_t = C_0' C_0 + \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} \begin{pmatrix} \varepsilon^2{}_1 & \varepsilon_1\varepsilon_2 & \varepsilon_1\varepsilon_3 \\ \varepsilon_2\varepsilon_1 & \varepsilon^2{}_2 & \varepsilon_2\varepsilon_3 \\ \varepsilon_3\varepsilon_1 & \varepsilon_3\varepsilon_2 & \varepsilon^2{}_3 \end{pmatrix} \begin{pmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{pmatrix} + \cdots$$

$$\begin{pmatrix} b_{11} & 0 & 0 \\ 0 & b_{22} & 0 \\ 0 & 0 & b_{33} \end{pmatrix} \begin{pmatrix} h_{11} & h_1h_2 & h_1h_3 \\ h_2h_1 & h_{22} & h_2h_3 \\ h_3h_1 & h_3h_2 & h_{33} \end{pmatrix} \begin{pmatrix} b_{11} & 0 & 0 \\ 0 & b_{22} & 0 \\ 0 & 0 & b_{33} \end{pmatrix}$$

In compact form, the above equation can also be written as:

$$H_t = C_0^{'} C_0 + A'\varepsilon_{t-1}\varepsilon'_{t-1}A + B' H_{t-1} B$$

For 2 variable case the model can be represented as:

$$H_t = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}' \begin{bmatrix} \varepsilon^2_{1,t-1} & \varepsilon_{1,t-1}\varepsilon_{2,t-1} \\ \varepsilon_{2,t-1}\varepsilon_{1,t-1} & \varepsilon^2_{2,t-1} \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}' H_{t-1} \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix}$$

$$h_{11,t} = c_{11} + a_{11}^2\varepsilon^2_{1,t-1} + 2a_{11}a_{21}\varepsilon_{1,t-1}\varepsilon_{2,t-1} + a_{21}^2\varepsilon^2_{2,t-1} + g_{11}^2 h_{11,t-1} + 2g_{11}g_{21}h_{12,t-1} + g_{21}^2 h_{22,t-1}$$

$$h_{12,t} = c_{12} + a_{11}a_{21}\varepsilon^2_{1,t-1} + (a_{21}a_{12} + a_{11}a_{22})\varepsilon_{1,t-1}\varepsilon_{2,t-1} + a_{21}a_{22}\varepsilon^2_{2,t-1} + g_{11}g_{12}h_{11,t-1}$$
$$+ (g_{21}g_{12} + g_{11}g_{22})h_{12,t-1} + g_{21}g_{22}h_{22,t-1}$$

$$h_{22,t} = c_{22} + a_{12}^2\varepsilon^2_{1,t-1} + 2a_{12}a_{22}\varepsilon_{1,t-1}\varepsilon_{2,t-1} + a_{22}^2\varepsilon^2_{2,t-1} + g_{12}^2 h_{11,t-1}$$
$$+2g_{12}g_{22}h_{12,t-1} + g_{22}^2 h_{22,t-1}$$

As already discussed, Bayesian analysis requires the assignment of prior distributions to the parameters of the model. Accordingly, priors for MGARCH and VAR models are defined. For the MGARCH model, normal priors are employed with different parametric ranges based on the parameters being estimated. The constant terms of each model are assigned a N(0,10) prior, while other parameters are given N(0,100) priors. The use of normal priors is primarily due to their conjugate properties, which simplify the computation of the posterior distributions. These priors are assigned following Fioruci et al. (2014), who demonstrated their effectiveness in the context of MGARCH models.

$\alpha \sim N\left(\underline{\alpha}_{Min}, \underline{V}_{Min}\right)$

If $\underline{V}_i$ denotes the block of $\underline{V}_{Min}$ associated with the K coefficients in equation i and $\underline{V}_{i,jj}$ as its diagonal elements, then a common implementation of the Minnesota prior would set:

$\underline{V}_{i,jj} = \dfrac{\underline{a}_1}{p^2}$ for coefficients on own lags

$\dfrac{\underline{a}_2\sigma_{ii}}{p^2\sigma_{jj}}$ for coefficients on lags of variable $j \neq i$

$\underline{a}_3\sigma_{ii}$ for coefficients on exogenous variables

This prior simplifies the complicated choice of fully specifying all the elements of $\underline{V}_{Min}$ in choosing three scalars $\underline{a_1}, \underline{a_2}, \underline{a_3}$.

The next prior used is a natural conjugate prior Normal-Wishart. The form of the prior is as follows:

$$\alpha|\Sigma \sim N\left(\underline{\alpha}, \Sigma \otimes \underline{V}\right)$$
$$\Sigma^{-1} \sim W\left(\underline{S}^{-1}, \underline{v}\right)$$

where, $\underline{\alpha}$, $\underline{V}$, $\underline{v}$ and $\underline{S}$ are to be selected by the experimenter depending upon the data set in use. Then the posterior of this prior is as follows:

$$\alpha|\Sigma, y \sim N\left(\overline{\alpha}, \Sigma \otimes \overline{V}\right)$$
$$\Sigma^{-1}|y \sim W\left(\overline{S}^{-1}, \overline{v}\right)$$

where,

$$\overline{\alpha} = vec\left(\overline{B}\right)$$
$$\overline{B} = \overline{V}\left[\underline{V}^{-1}\underline{B} + X^{'}X B\right]$$
$$\overline{S} = S + \underline{S} + B^{'}X^{'}X B + \underline{B}^{'}\underline{V}^{-1}\underline{B} - \overline{B}^{'}\left(\underline{V}^{-1} + X^{'}X\right)\overline{B}$$
$$\overline{v} = T + \overline{v}$$

The third prior taken up is the independent Normal-Wishart, which has the following form:

$$p(\beta, \Sigma^{-1}) = p(\beta)p(\Sigma^{-1})$$

where

$$\beta \sim N(\underline{\beta}, \underline{V_\beta})$$

and

$$\Sigma^{-1} \sim W(\underline{S}^{-1}, \underline{v})$$

This prior allows for the prior covariance matrix $\underline{V_\beta}$, to take any values chosen by the researcher, rather than the restrictive $\sum \otimes \underline{V}$ form of the natural conjugate prior. In this prior, the joint posterior $p(\beta, \Sigma^{-1}|y)$ does result in an easily computable form that would allow easy Bayesian analysis this is due to the fact that posterior means and variances do not have analytical forms.

4. Data description and illustration

To demonstrate the application of the Bayesian framework, we implement it within the context of a Multivariate GARCH (MGARCH) model, specifically using the BEKK (Baba-Engle-Kraft-Kroner) specification. For this illustration, a dataset comprising two monthly time series—the International Price Index and the Domestic Price Index of edible oils—has been utilized. The International edible oil price index was obtained from the World Bank's Commodity Prices Indices (Pink Sheet), accessible via its official website. The Domestic edible oil price index was sourced from the Office of the Economic Adviser, Ministry of Commerce and Industry, Government of India. The dataset spans from January 1990 to January 2016, consisting of 313 monthly observations.

In accordance with the Bayesian approach discussed earlier, prior distributions were assigned to the model parameters, and the posterior distributions were derived using Markov Chain Monte Carlo (MCMC) simulation techniques. The time series plots of both the International and Domestic price indices are presented in Figure 1, providing a visual overview of the series' behavior over time. The Bayesian parameter estimates for the BEKK-MGARCH model are summarized in Table 1, offering insight into the volatility dynamics captured by the model. Furthermore, the estimated conditional volatilities of the two series, derived from the posterior distributions, are graphically represented in Figure 2, illustrating the time-varying volatility patterns inherent in the data.

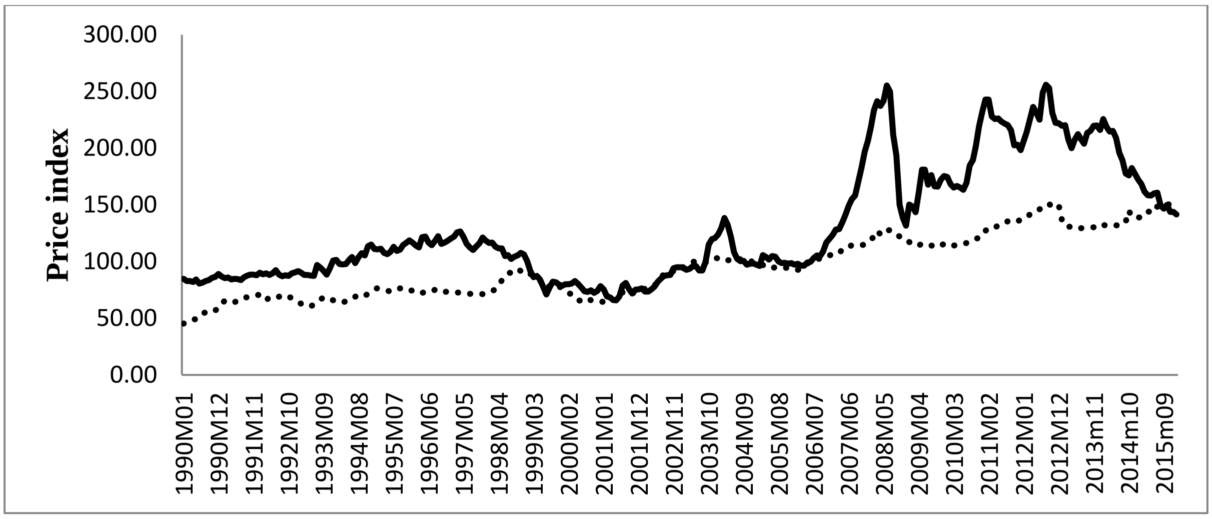


Figure 1. Time plot of International (bold) and Domestic (dotted) edible oils price indices

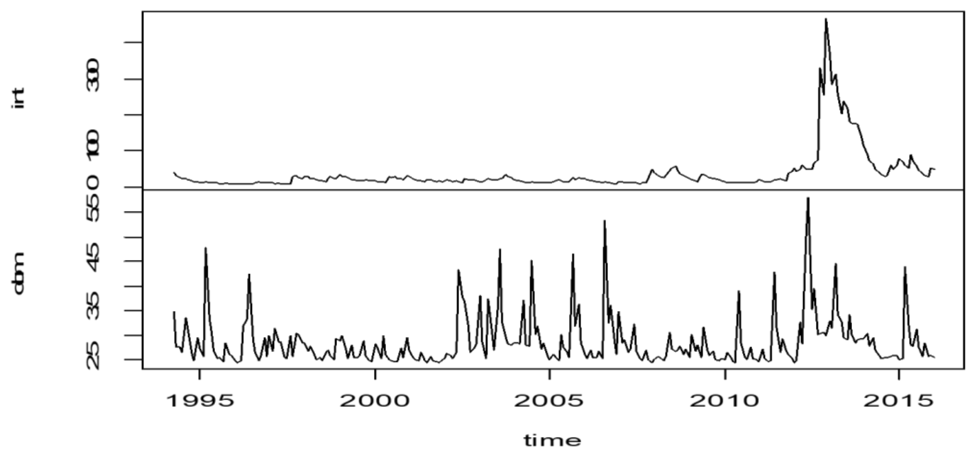


Figure 2. Conditional variance of Domestic (dom) and International (int) edible oils price indices after fitting MGARCH-BEKK model

Table 1. Estimates of Bayesian MGARCH-BEKK model for International and Domestic edible oils price indices

Coefficients	**Estimate**	**Std. Error**	**t value**	**P value**
C11	-0.158	0.026	-6.076	<0.001
C21	0.193	0.025	7.720	<0.001
C22	0.079	0.026	3.038	<0.001
A11	0.231	0.024	9.625	<0.001
A21	0.488	0.025	19.520	<0.001
A12	-0.166	0.026	-6.384	<0.001
A22	-0.004	0.025	-0.160	0.873
B11	-0.025	0.027	-0.925	0.355
B21	0.507	0.026	19.500	<0.001
B12	0.374	0.025	14.960	<0.001
B22	0.264	0.026	10.153	<0.001

R code for analysing a time series data using ARIMA and AR-GARCH model

```
library(“tseries”)
library(“forecast”)
library(“fgarch”)
setwd(&quot;C:/Users/Desktop&quot;) # Setting of the work directory
data&lt;-read.table(&quot;data.txt&quot;) # Importing data
datats&lt;-ts(data,frequency=12,start=c(1982,4)) # Converting data set into time series
```

```
plot.ts(datats) # Plot of the data set
adf.test(datats) # Test for stationarity
diffdatats&lt;-diff(datats,differences=1) # Differencing the series
datatsacf&lt;-acf(datats,lag.max=12) # Obtaining the ACF plot
datapacf&lt;-pacf(datats,lag.max=12) # Obtaining the PACF plot
auto.arima(diffdatats) # Finding the order of ARIMA model
datatsarima&lt;-arima(diffdatats,order=c(1,0,1),include.mean=TRUE) # Fitting of ARIMA
model
forearimadatats&lt;-forecast.Arima(datatsarima,h=12) # Forecasting using ARIMA model
plot.forecast(forearimadatats) # Plot of the forecast
residualarima&lt;-resid(datatsarima) # Obtaining residuals
archTest(residualarima,lag=12) # Test for heteroscedascity
# Fitting of AR-GARCH model
garchdatats&lt;-garchFit(formula = ~ arma(2)+garch(1, 1), data = datats, cond.dist =
c(&quot;norm&quot;),
include.mean = TRUE, include.delta = NULL, include.skew = NULL, include.shape =
NULL, leverage = NULL, trace = TRUE,algorithm = c(&quot;nlminb&quot;))
# Forecasting using AR-GARCH model
forecastgarch&lt;-predict(garchdatats, n.ahead = 12, trace = FALSE, mse =
c(&quot;uncond&quot;),
plot=FALSE, nx=NULL, crit_val=NULL, conf=NULL)
plot.ts(forecastgarch) # Plot of the forecast
```

Bibliography

Asai, M., McAleer, M. and Yu, J. (2006). Multivariate stochastic volatility: a review. *Econometric Reviews*, **25(2–3),** 145–175.

Asai, M. (2015). Bayesian analysis of general asymmetric multivariate garch models and news impact curves. *Journl of Japan Statistical Society*, **45**, 129-144.

Bauwens, L., Laurent, S. and Rombouts, J. V. K. (2006). Multivariate GARCH models: a survey. *Journal of Applied Econometrics*, **21**, 79-109.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroscedasticity. *Journal of Econometrics*, **31**, 307-327.

Carriero, A., Kapetanios, G. and Marcellino, M. (2009). Forecasting exchange rates with a large Bayesian VAR. *International Journal of Forecasting*, **25**, 400–417.

Engle, R.F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of U.K. inflation. *Econometrica*, **50**, 987-1008.

Fioruci, J. A., Ehlers, R. S. and Filho, M. G. A. (2014). Bayesian multivariate GARCH models with dynamic correlations and asymmetric error distributions. *Journal of Applied Statistics*, **41**, 320-331.

Lama, A. (2017). Investigations on Bayesian multivariate time-series models. *Unpublished PhD thesis*, PG School, ICAR-IARI.

Sims, C. (1980). Macroeconomics and reality. *Econometrica*, **48**, 1-48.

Count Time Series Models

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, Prabhat Kumar

ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115

Email: santosha.rathod@icar.org.in

Introduction

Count data refers to data in which observations assume only non-negative integer values, typically arising from counting occurrences rather than ranking or measuring. These data exhibit unique characteristics, such as discreteness, skewness, over-dispersion (where variance exceeds the mean), and often excess zeros. Count data is prevalent in various domains and is increasingly encountered in time series formats. Examples include: modeling pest and disease outbreaks in agriculture, assessing the health effects of environmental pollution, analyzing daily rainfall, air quality indices, monthly polio cases, daily hospital admissions for asthma, and traffic accident frequencies. Similarly, purchase frequencies in a store or consumer demand over time can also be modeled as time series of counts.

Time series analysis of count data is an evolving field motivated by its wide range of applications. Such data involve temporal dependence (autocorrelation) and discrete-valued distributions, and therefore standard time series models—which often assume normality and continuity—may not be appropriate. Neglecting either the discrete nature or the serial dependence in the data can lead to serious model misspecification.

A successful model for count time series must effectively account for both dependence between observations and over-dispersion. In many cases, events are relatively rare, rendering the use of the normal distribution inadequate. Consequently, models such as the Autoregressive Conditional Poisson (ACP) model have been developed. In its basic form, the ACP model assumes that counts follow a Poisson distribution, where the conditional mean (given past observations) evolves according to an autoregressive structure. While such a model is conditionally equi-dispersed, it often becomes unconditionally over-dispersed due to temporal dependence.

To further handle situations where mean and variance are not equal, more flexible approaches have been proposed. Notably, the Integer-valued Generalized Autoregressive Conditional Heteroscedasticity (INGARCH) models extend the Poisson and negative binomial frameworks by incorporating conditional heteroscedasticity into count data modeling. The INGARCH model is considered a member of the generalized linear model (GLM) family, adapted specifically for integer-valued time series.

In agricultural research and related fields, count-based time series—such as number of pest attacks, disease incidences, or harvest losses—are common. Employing appropriate statistical models like Poisson regression, Negative Binomial models, ACP, and INGARCH is crucial for accurate inference, forecasting, and policy recommendations.

Poisson Regression Model

Poisson distribution is a class of generalized linear model which follows Poisson distribution. Let us consider a random variable y follows a Poisson distribution with parameter μ, if it takes integer values $y = 0, 1, 2, \dots$ with probability distribution

$$P(Y = y) = \frac{e^{-\mu}\mu^{y}}{y!} \quad \mu > 0, \qquad (1)$$

where the mean and variance expression of the distribution is $E(y) = VAR(y) = \mu$. So it is seen that the mean and variance expression are equal and if any factor influences the mean value, it will influence the variance also. Thus, the homoscedasticity would not be appropriate for this kind of distribution (Chen and Lee 2016). The parameters of Poisson regression model $y_i = \exp(X_i{}'\beta) + \varepsilon_i$ can be estimated using maximum likelihood method.

Assumptions in Poisson Regression

- The probability of at least one occurrence of the event in a given time interval is proportional to the length of the interval.
- The probability of two or more occurrences of the event in a very small time interval is negligible.
- The numbers of occurrences of the event in disjoint time intervals are mutually independent.

Then the probability distribution of the number of occurrences of the event in a fixed time interval is Poisson with mean $\mu = \lambda t$, where λ is the rate of occurrence of the event per unit of time and t is the length of the time interval. A process satisfying the three assumptions listed above is called as a Poisson process. The most important point in estimating parameters of Poisson regression is its relationship between the mean and the variance. A useful property of the Poisson distribution is that the sum of independent Poisson random variables is also Poisson. Specifically, if Y1 and Y2 are independent with Yi ~ P(μi) for i = 1, 2 then Y1 + Y2 ~ P(μ1 + μ2).

Negative Binomial Regression Model

A distribution of counts will usually have a variance that’s not equal to its mean. When we see this happen with data that we assume (or hope) is Poisson distributed, we say we have under or over-dispersion, depending on if the variance is smaller or larger than the mean. Performing Poisson regression on count data that exhibits this behavior results in a model that doesn’t fit well. Negative binomial regression is a type of generalized linear model in which the dependent variable Y is a count of the number of times an event occurs. It can be used for over-dispersed count data, that is when the conditional variance exceeds the conditional mean. It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression and it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for the Negative binomial regression are likely to be narrower as compared to those from a

Poisson regression model. Unlike the Poisson distribution, the variance and the mean are not equivalent. This suggests it might serve as a useful approximation for modeling counts with variability different from its mean.

Let Y is random variable which follows the negative binomial distribution with parameters (r, θ), where θ ∈ (0, 1) and r an integer, then its probability mass function is given by

$$P[Y = y] = \binom{y+r-1}{y} \theta^y (1-\theta)^r, \quad y = 0,1,2, \ldots$$
(2)

$y \sim NegBin(r, \theta)$ Therefore $E[Y] = \frac{r\theta}{(1-\theta)}$ and $Var[Y] = \frac{r\theta}{(1-\theta)^2}$.

Model Assumption

As we mentioned earlier, negative binomial models assume the conditional means are not equal to the conditional variances. This inequality is captured by estimating a dispersion parameter (not shown in the output) that is held constant in a Poisson model. Thus, the Poisson model is actually nested in the negative binomial model. We can then use a likelihood ratio test to compare these two and test this model assumption. To do this, we will run our model as a Poisson.

Negative binomial regression can be used for over-dispersed count data that is when the conditional variance exceeds the conditional mean. It can be considered as a generalization of Poisson regression since it has the same mean structure as Poisson regression and it has an extra parameter to model the over-dispersion. If the conditional distribution of the outcome variable is over-dispersed, the confidence intervals for the Negative binomial regression are likely to be narrower as compared to those from a Poisson regression model.

Generalized Linear Model

Let us denote the count time series by $\{Y_t : t \in N\}$ and time varying r-dimensional covariate vector say $\{X_t : t \in N\}$ i.e. $X_t = (X_{t,1,} \ldots, X_{t,r,})^T$. The conditional mean becomes $E(Y_t | F_{t-1}) = \lambda_t$ and F_t is historical data.

The generalized model form is expressed as follows

$$g(\lambda_t) = \beta_0 + \sum_{k=1}^{p} \beta_k \tilde{g}\ (Y_{t-i_k}) + \sum_{l=1}^{q} \alpha_l g\ (\lambda_{t-jl}) + \eta^T \quad (3)$$

Where g is link function and $\tilde{g}$ is transformation function. $g(\lambda_t)$ is linear predictor. To allow for regression on arbitrary past observations of the response, $P = \{i_{1,} i_{2,} \ldots, i_{p,}\}$ and 0< $0 < i_1 < i_2 < \cdots < i_p < \infty$ for leads to lagged observations $Y_{t-i_1}, \ldots, Y_{t-i_p}$. Set $Q = \{j_{1,} j_{2,} \ldots, j_{q,}\}$ and 0<$0 < j_1 < j_2 < \cdots < j_p < \infty$. The set Q lagged in parameter mean i.e. $\lambda_{t-i_1}, \ldots, \lambda_{t-i_p}$.

Specification of the model order, i.e., of the sets P and Q, are guided by considering the empirical autocorrelation functions of the observed data. This approach is described for ARMA models in many time series analysis literatures.

Cases of GLM

Case 1: Consider the situation where g and $\tilde{g}$ are equal to identity i.e. $g(x) = \tilde{g}(x) = x$, further P={1,…,p}, Q={1,…,q} and $\eta = 0$ then the GLM model (3) becomes

$$\lambda_t = \beta_0 + \sum_{k=1}^{p} \beta_k \; Y_{t-i_k} + \sum_{l=1}^{q} \alpha_l \lambda_{t-jl} \qquad (4)$$

Assuming further that $Y_t|Y_{t-1}$ is Poisson distributed, then we obtain an integer-valued GARCH model of order p and q, abbreviated as INGARCH(p,q). These models are also known as autoregressive conditional Poisson (ACP) models (Heinen 2003, Ferland *et al.* 2006 and Fokianos, *et al.* 2009).

Case 2: The Negative Binomial distribution allows for a conditional variance to be larger than the mean $\lambda_t$which is often referred to as over-dispersion (Christou and Fokianos 2014). It is assumed that $Y_t|F_{t-1} \sim NegBionom(\lambda_t, \emptyset)$. The Poisson distribution is a limiting case of the Negative Binomial when $\emptyset \to \infty$.

R codes to implement count TS models

```
rm(list = ls())
d1=read.table(file = "CHN.txt", header = T)
head(d1)
attach(d1)
reg1=cbind(MAXT, MINT, RF, MRH,  ERH)
YSB1=as.integer(YSB)
Box.test(YSB1)
##### training data  ############
reg11=cbind(MAXT[1:421], MINT[1:421], RF[1:421], MRH[1:421],  ERH[1:421])
reg12=cbind(MAXT[422:428], MINT[422:428], RF[422:428], MRH[422:428],
ERH[422:428])
reg21=cbind(MAXT[1:451], MINT[1:451], RF[1:451], MRH[1:451],  ERH[1:451])
reg22=cbind(MAXT[452:461], MINT[452:461], RF[452:461], MRH[452:461],
ERH[452:461])
ysb.train1=YSB1[1:421]
ysb.test1=YSB1[422:428]
ysb.train2=YSB1[1:451]
ysb.test2=YSB1[452:461]
############## Poisson INGARCH #############
M1=tsglm(ysb.train2, model=list(past_obs=5, past_mean=5),
             xreg=reg21, distr="poisson")
Box.test(M1$residuals)
summary(M1)
coeftest(M1)
M1$fitted.values
```

```
predict(M1, n.ahead=8, newxreg=reg22)
############### NB INGARCH ##############
M2=tsglm(ysb.train2, model=list(past_obs=5, past_mean=5),
        xreg=reg21, distr="nbinom")
Box.test(M2$residuals)
summary(M2)
coeftest(M2)
M2$fitted.values
predict(M2, n.ahead=8,newxreg=reg22)
```

Suggested readings

- Chen, C.W.S. and Lee, S. (2016). Generalized Poisson autoregressive models for time series of counts. Computational Statistics & Data Analysis, 99: 51-67.
- Christou V, Fokianos K (2014). Quasi-Likelihood Inference for Negative Binomial Time Series Models. Journal of Time Series Analysis, 35(1), 55–78.
- Ferland R, Latour A, Oraichi D (2006). Integer-Valued GARCH Process. Journal of Time Series Analysis, 27(6), 923–942.
- Fokianos K, Rahbek A, Tjostheim D (2009). Poisson Autoregression. Journal of the American Statistical Association, 104(488), 1430–1439.
- Heinen A (2003). Modelling Time Series Count Data: An Autoregressive Conditional Poisson Model. CORE Discussion Paper, 62.
- Liboschik,T., Fokianos, K. and Fried, R. (2016). tscount: An R Package for Analysis of Count Time Series Following Generalized Linear Models. Vignette of R package tscount version 1.3.0.

Spatiotemporal Time Series Modelling and Forecasting for Abiotic Stress Management in Agriculture

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, Prabhat Kumar

ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115

Email: santosha.rathod@icar.org.in

Introduction

Spatiotemporal time series are observations recorded across both space and time, incorporating systematic dependencies among spatial locations and temporal patterns. These models effectively handle single variables recorded over time from multiple locations. Classic examples of spatio-temporal data include daily or hourly carbon emission levels from monitoring stations, daily river discharge across various basins, frequent weather parameter recordings (temperature, rainfall, humidity, etc.) from different agro-climatic zones, and traffic flow data from multiple checkpoints. Traditionally rooted in geo-statistics, spatio-temporal modeling has now found applications in sociology, economics, environmental sciences, ecology, and notably in agricultural research.

In the context of abiotic stress management in agriculture, spatio-temporal models are highly relevant as they allow for detection, monitoring, and forecasting of stress patterns like drought, heat waves, frost, and soil salinity across different regions and seasons. These stresses are not static—they evolve both geographically and temporally, making it essential to adopt models that capture variations across space and time simultaneously. For instance, analyzing how drought severity varies across districts over cropping seasons or how rising temperature trends affect heat-stress in wheat zones helps in site-specific and timely adaptation strategies.

Research suggests that combining spatial and temporal data enhances the modeling accuracy and decision-making effectiveness, especially under conditions of uncertainty posed by climate variability. Thus, spatio-temporal modeling forms the backbone of early warning systems, stress forecasting tools, and real-time advisories that are crucial for mitigating the adverse impacts of abiotic stress on crop productivity and food security.

Despite significant advances in univariate time series modeling, progress in spatio-temporal time series analysis has been relatively limited. This is primarily due to computational complexities and the inaccessibility of simultaneous spatial and temporal information. While univariate time series models focus solely on temporal autocorrelation—typically modeled through the Box-Jenkins autoregressive moving average (ARMA) framework (Box and

Jenkins, 1970)—spatio-temporal models are designed to capture dependencies across both space and time.

Spatio-temporal models incorporate observations from multiple spatial locations across several time periods, thereby enabling a more comprehensive understanding of complex dynamic systems. In this context, spatio-temporal autoregressive moving average (STARMA) models extend the conventional ARMA models by including spatial lags in both the autoregressive and moving average components. These models are particularly useful for datasets where observations exhibit autocorrelation not only over time but also across geographic or spatial domains.

STARMA Model

The STARMA model, introduced by Pfeifer and Deutsch (1980b), is tailored to handle such scenarios. It considers a single variable $Z_i(t)$, observed at N fixed spatial locations (i = 1, 2,…, N) on T time periods (t = 1, 2, . . ., T). The N spatial locations can be a geographical location, country, state, *etc*. The spatial dependencies between N times series is incorporated through $N*N$ spatial weight matrices. Analogous to univariate time series, ***Z(t)*** is expressed as a linear combination of past observations and errors. The STARMA model (Pfeifer and Deutsch, 1980a), denoted by $STARMA(p_{\lambda_1, \lambda_2, \ldots, \lambda p}, q_{m_1, m_2, \ldots, mq})$ can be represented in the matrix equation as follows;

$$Z(t) = \sum_{K=1}^{p} \sum_{l=0}^{\lambda_k} \phi_{kl} W^l Z(t-k) - \sum_{K=1}^{q} \sum_{l=0}^{m_k} \theta_{kl} W^l \varepsilon(t-k) + \varepsilon(t) \quad \ldots (1.1)$$

Where,

$\boldsymbol{z(t)} = [\boldsymbol{z_1}(t), \ldots\ldots, \boldsymbol{z_N}(t)]'$is a $N \times 1$ vector of observations at time $t = 1, \ldots, T$,

p is the autoregressive order (AR) with respect to time,

q is the moving average order (MA) with respect to time,

λ_k is the spatial order of the kth AR term,

m_k is the spatial order of the kth MA term,

ϕ_{kl} is the AR parameter at temporal lag k and spatial lag l (scalar),

θ_{kl} is the MA parameter at temporal lag k and spatial lag l (scalar) and

W^l is the $N*N$ spatial weight matrix with spatial order l with diagonal elements zero and non-diagonal elements is the relation between sites.

The spatial weight matrix $W^{(0)} = \mathrm{I}_N$ *i.e.* Identity matrix and each row of W^l must add up to one. The random error vector $\varepsilon(t) = [\varepsilon_1(t), \varepsilon_2(t), \ldots, \varepsilon_N(t)]'$ is normally distributed at time t with $E[\varepsilon(t)] = 0$, $E[\varepsilon(t)\varepsilon'(t+s)] = \begin{cases} G = \sigma^2 I_N \; is \; s = 0 \\ 0, \;\; otherwise \end{cases}$ and $E[\varepsilon(t)\varepsilon'(t+s)] = 0, for\; s > 0$.

There are two subclasses of the STARMA model, in equation (3) when q=0, only autoregressive terms remain and consequently the model progresses toward becoming space-time autoregressive or STAR model which is represented as follows;

$$\boldsymbol{Z(t)} = \sum_{K=1}^{p} \sum_{l=0}^{\lambda_k} \phi_{kl} \, W^l \, Z(t-k) + \varepsilon(t) \qquad \ldots(1.2)$$

When p becomes 0, only moving average terms remains and hence the model becomes space-time moving average or STMA model which is represented as follows;

$$\boldsymbol{Z(t)} = \varepsilon(t) - \sum_{K=1}^{q} \sum_{l=0}^{m_k} \theta_{kl} \, W^l \, \varepsilon(t-k) \qquad \ldots(1.3)$$

Spatial weight matrix

Building the spatial weight matrix is a crucial step in STARMA modeling. The process involves determining the hierarchical ordering of neighbors for each location and selecting an appropriate sequence of weighting matrices. This selection is subjective and depends on the model builder's discretion. The complexity of the weight matrix directly influences the difficulty in estimating the parameters of the STARMA model. In most applications, it is often assumed that spatial patterns are equal and regularly spaced to simplify the modeling process. However, this is typically just a simplifying assumption, as in reality, sites are usually irregularly spaced. One common and simple method for assigning weights is the binary scheme, where a weight of one is assigned if two areas share a common border and zero otherwise, as suggested by Griffith (1996, 2009).

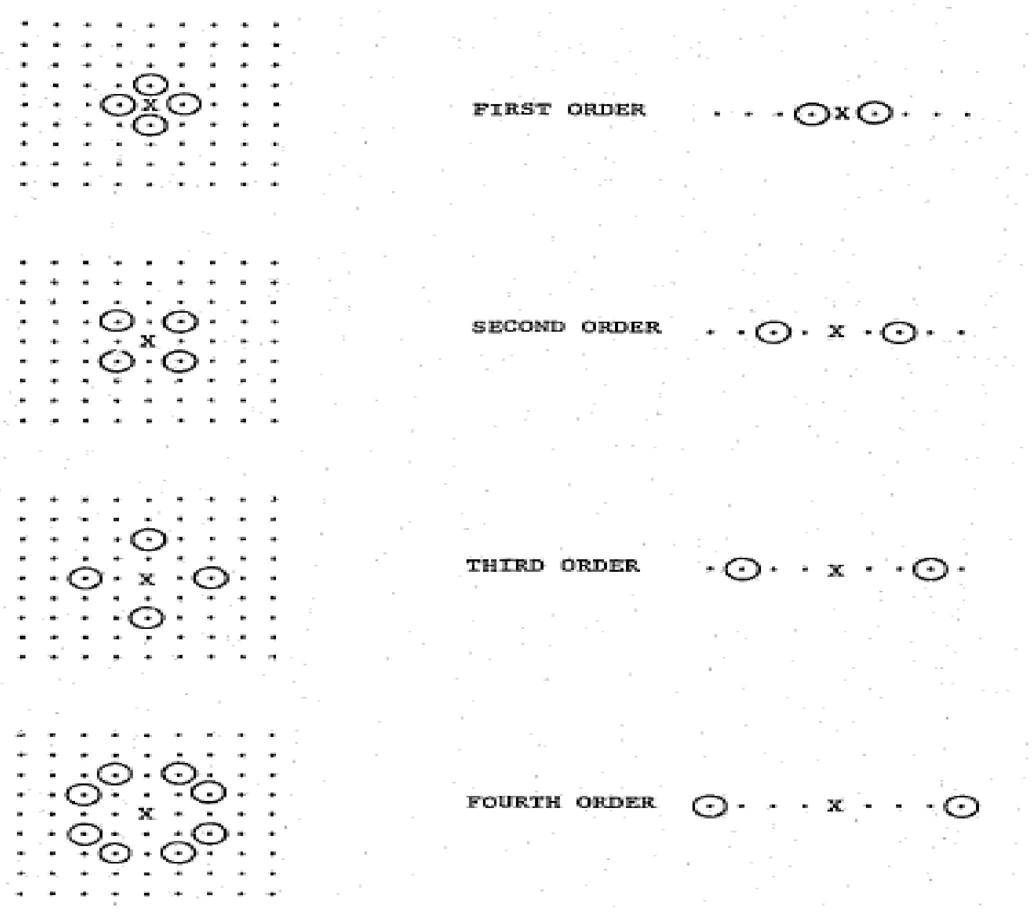


Fig.1: Schematic representation of spatial weight grid

In spatio-temporal modeling, the construction and normalization of the spatial weight matrix is a critical step, as it governs the spatial dependence structure among the observational units. A common practice in constructing these matrices is row normalization, wherein each row is scaled such that its elements sum to one. This approach standardizes the influence received by each spatial unit from its neighbors. However, some studies have employed column normalization, where the focus shifts to the influence exerted by a unit iii on others, rather than the influence received from a neighboring unit *j*.

The choice of normalization scheme is non-trivial, as it can significantly affect the inferences drawn from the model. Different weight structures may result in varying degrees of spatial influence, potentially introducing bias and leading to divergent interpretations of the underlying spatial dynamics. Moreover, in spatio-temporal data analysis, the assumption that the influence from neighboring units remains constant over time may not always be realistic.

Consequently, the specification of spatial weights should be done with careful consideration of temporal variability and hierarchical spatial relationships.

Spatial weight matrices should also incorporate the ordering of neighbors. For example, first-order neighbors are those directly adjacent to a target location, while second-order neighbors are more distant but still exert influence, followed by third-order and higher-order neighbors. This hierarchical structure allows for modeling spatial autocorrelation at multiple levels of proximity. A schematic representation of such a spatial weight grid, as proposed by Pfeifer and Deutsch (1980b), is illustrated in Figure 1.

STARMA Modeling Procedure

Similar to the well-established Box–Jenkins methodology for univariate ARIMA models, the STARMA model (Spatio-Temporal Autoregressive Moving Average) follows a three-stage modeling procedure comprising:

1. Identification
2. Estimation, and
3. Diagnostic Checking

This structured approach, as proposed by Pfeifer and Deutsch (1980b), enables systematic development and evaluation of spatio-temporal models.

A STARMA model is considered stationary if its covariance structure remains invariant over time and all the roots of the model lie within the unit root circle. In this context, the invertibility of the spatio-temporal autoregressive (STAR) model is a necessary condition for the stationarity of the STARMA model. Ensuring these properties is essential for the model to yield valid statistical inferences and reliable forecasts.

Model Identification

The space-time autocorrelation function (STACF) and space-time partial autocorrelation function (STPACF) are employed to determine the orders of the STAR and STMA models. Similar to the univariate ARIMA model, the identification of STAR and STMA orders is based on the presence of significant spikes in the STACF and STPACF plots. The space-time autocorrelation function (STACF) between the *l*th and *k*th order neighbors with a time lag *s* (where $s = 1, \dots, k$ and $h = 0, 1, \dots, \lambda$) is defined as follows.

$$\rho_{lk}(s) = \frac{\sum_{i=1}^{N}\sum_{t=1}^{T-S} W^{(l)} Z_i(t) W^{(k)} Z_i(t+s)}{[\sum_{i=1}^{N}\sum_{t=1}^{T-S}(W^{(l)}Z_i(t))^2 \ . \ (W^{(k)}Z_i(t+s))^2]^{\frac{1}{2}}} \quad \ldots(1.4)$$

The space time partial autocorrelation function (STPACF) is expressed in following equation;

$$\rho_{h0}(s) = \sum_{j=1}^{k}\sum_{l=0}^{\lambda} \phi_{jl}\rho_{hl}(s-j) \quad \ldots(1.5)$$

Characteristics of the theoretical space-time autocorrelation and partial autocorrelation functions for STAR, STMA and STARMA models (1.1) are depicted in following table.

Table 1: STACF and STPACF of STAR, STMA and STARMA models

Process	**STACF**	**STPACF**
STAR	tails off with both space and time	cuts off after p lags in time and λ_p lags in space
STMA	cuts off after q lags in time and m_q lags in space	tails off with both space and time
STARMA	tails off	tails off

Model Parameter Estimation

The maximum likelihood estimates of

$\mathbf{\Phi} = [\phi_{10,}\phi_{11}, \ldots, \phi_{1\lambda_1}, \ldots, \phi_{p0}, \phi_{p1}, \ldots, \phi_{p\lambda_p}]'$ and

$\Theta = [\theta_{10,}\theta_{11}, \ldots, \theta_{1\lambda_1}, \ldots, \theta_{q0}, \theta_{q1}, \ldots, \theta_{p\lambda_q}]'$ rely on the assumption of errors i.e. which are normally distributed with mean zero and variance-covariance matrix equal to $\sigma^2 I_N$. The likelihood function for the same is defined as follows;

$$f(\varepsilon|\Phi, \Theta, \sigma^2) = (2\pi)^{\frac{-TN}{2}} |\sigma^2 I_{NT}|^{\frac{-1}{2}} \exp\{-\frac{1}{2\sigma^2}\epsilon' I \epsilon\}$$

$$= (2\pi)^{\frac{-TN}{2}} (\sigma^2)^{\frac{-TN}{2}} \exp\{-\frac{S(\Phi,\Theta)}{2\sigma^2}\} \quad \ldots(1.6)$$

Where,

$S(\Phi, \Theta) = \epsilon' I \epsilon = \sum_{i=1}^{N}\sum_{t=0}^{T} \epsilon_i^2(t)$ is the sum of squares of the errors and $\epsilon' = [\epsilon_1(1), \ldots, \epsilon_1(T), \ldots, \epsilon_N(1), \ldots, \epsilon_N(T)]$. Finding the values of the parameters that maximize the likelihood function is equivalent to finding the values of Φ and Θ that minimize the sum of squares $S(\Phi, \Theta)$. Therefore, the problem is reduced to finding the least squares estimates of Φ and Θ .

The errors $\varepsilon(t)$ need to be recursively calculated using the equation:

$$\varepsilon(t)=z(t)+\sum_{k=1}^{p}\sum_{l=0}^{\lambda_k}\phi_{kl}\,W^{(l)}z(t-k)-\sum_{k=1}^{q}\sum_{l=0}^{m_k}\theta_{kl}\,W^{(l)}\varepsilon(t-k) \qquad \ldots(1.7)$$

for t = 1, ... , T and for given values of the parameters (Φ,Θ).

Because the values of the observations z and of the errors care unknown for times previous to time 1, these initial values need to be calculated. Thus, for any given choice ofthe parameters (Φ,Θ) and starting values $(z*, c*)$ the set of values $c(cI >, e\ I\ z*, c*, W)$ could be calculated successively given a particular data set z. The log likelihood associated with the parameter values (Φ, Θ, σ^2) conditional on the choice of $(z*, c*)$ would be

$$l_*(\Phi,\Theta,\sigma^2)=-\frac{TN}{2}\ln(2\pi)-\frac{TN}{2}\sigma^2-\frac{S_*(\Phi,\Theta)}{2\sigma^2} \qquad \ldots(1.8)$$

So for fixed σ^2 , the conditional maximum likelihood estimates of Φ, Θ are the conditional least squares estimates obtained by finding the values of Φ, Θ that minimize the conditional sum of squares function

$$S_*(\Phi,\Theta)=\sum_{i=1}^{N}\sum_{t=0}^{T}\epsilon_i^2(t) \qquad \ldots(1.9)$$

Diagnostic-Checking

At this stage, the goal is to assess whether the model adequately represents the data. If the fitted model is appropriate, the residuals should resemble Gaussian white noise, meaning they should be normally distributed with a mean of zero and a variance-covariance matrix equal to $\sigma^2 I_n$. One approach to test for correlation is to compute the sample space-time autocorrelations of the residuals and examine whether any significant structure remains. If the model is correctly specified, then

$$var\left(\hat{\rho}_{l0}(s)\right)\approx\frac{1}{N(T-s)} \qquad \ldots(1.10)$$

where $\hat{\rho}_{l0}(s)$ represents the space-time autocorrelation function of the residuals from the fitted model. Since these residual space-time autocorrelations are approximately normal, they can be standardized and assessed for statistical significance. If residuals display dependence, the structure is identified, and the model is revised accordingly.

Case Study: Forecasting Monthly Mean Maximum Temperature in North Karnataka Districts (Rathod et al., 2018). This study focuses on modeling and forecasting the monthly mean maximum temperature across nine districts in the northern region of Karnataka, India (see Fig. 1). The proposed STARMA methodology was applied to this spatio-temporal dataset. For model development, data from January 2000 to August 2015 were used, while the period from September 2015 to August 2016 was reserved for validating the forecasting performance of the fitted model.

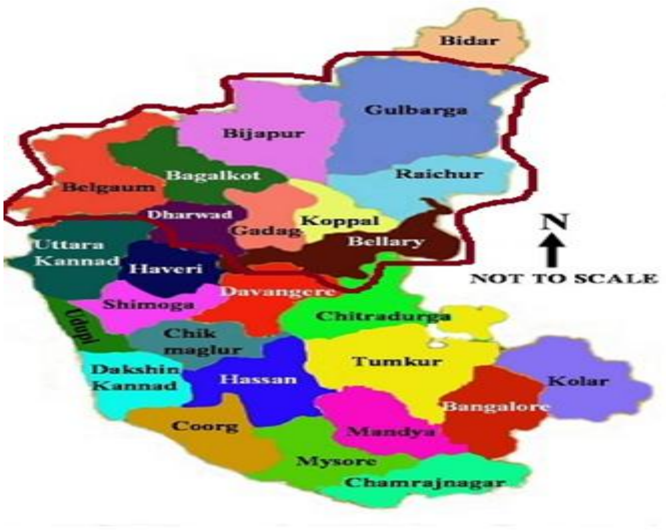


Fig. 1. Geographical map of karnataka

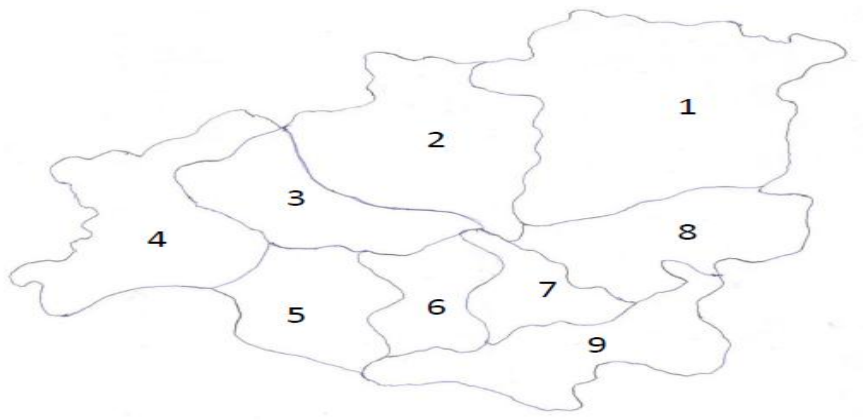
Construction of spatial weight matrix:

As described in the methodology section, the spatial weight matrix was constructed by assigning equal weights to all neighboring locations. The spatial configuration of the nine selected locations is illustrated in Figure 2.10, where each location is labeled numerically from 1 to 9. Based on their geographical proximity, a connectivity-based spatial weight matrix was formulated.

For instance, with reference to location 1, the first-order neighbors are locations 2 and 8, while locations 3, 6, and 7 are identified as second-order neighbors. The complete list of first- and second-order neighbors for all nine locations is provided in Table 1. In the uniform spatial weighting scheme, equal weights are assigned to each neighboring unit. For row normalization, which ensures that each row of the matrix sums to one, the weight assigned to each neighbor is computed as 1/*n*, where *n* is the number of neighbors for a particular location. For example, location 1 (Gulbarga) has two first-order neighbors; thus, each neighbor is assigned a weight of ½=0.5. Following this logic, weights for all nine locations are computed and presented in the first-order spatial weight matrix in Table 3.

In addition, this study also incorporates a second-order spatial weight matrix into the STARMA model. For location 1, the second-order neighbors are locations 3, 6, and 7. Since there are three neighbors, a weight 1/3≈0.33 is assigned to each. This procedure is applied similarly for all other locations, and the resulting second-order spatial weight matrix is presented in Table 4. To estimate the Spatio-Temporal Autocorrelation Function (STACF) and Spatio-Temporal Partial Autocorrelation Function (STPACF), the model requires incorporation of the zero-order (Table 2), first-order (Table 3), and second-order (Table 4) spatial weight matrices. Notably, in the case of the zero-order spatial weight matrix, where no external spatial influence is assumed, all diagonal elements are set to one, reflecting self-dependence of each spatial unit, and off-diagonal elements are zero, indicating no spatial interaction with other locations

Fig. 2.: Map of districts/locations considered



1. Gulbarga
2. Bijapur
3. Bagalkot
4. Belgaum
5. Dharwad
6. Gadag
7. Koppal
8. Raichur
9. Bellary

Table 1: Neighbors of each site for each spatial order

Location	Order	
	1	2
1	2,8	3,6,7
2	1,3	4,8,7
3	2,4,5,6	7,8
4	3,5	2
5	3,4,6	9
6	3,5,7,9	2,8
7	6,8,9	1,2,3
8	1,7,9	2,3,6
9	6,7,8	5

Table 2.: Spatial weight matrix of order zero

Location	Gulbarga	Bijapur	Raichur	Bagalkot	Belgaum	Dharwad	Gadag	Koppal	Bellary
Gulbarga	1	0	0	0	0	0	0	0	0
Bijapur	0	1	0	0	0	0	0	0	0
Raichur	0	0	1	0	0	0	0	0	0
Bagalkot	0	0	0	1	0	0	0	0	0
Belgaum	0	0	0	0	1	0	0	0	0
Dharwad	0	0	0	0	0	1	0	0	0
Gadag	0	0	0	0	0	0	1	0	0
Koppal	0	0	0	0	0	0	0	1	0
Bellary	0	0	0	0	0	0	0	0	1

Table 3: First order spatial weight matrix for Maximum temperature data

Location	Gulbarga	Bijapur	Raichur	Bagalkot	Belgaum	Dharwad	Gadag	Koppal	Bellary
Gulbarga	0	0.5	0	0	0	0	0	0.5	0
Bijapur	0.5	0	0.5	0	0	0	0	0	0
Raichur	0	0.25	0	0.25	0.25	0.25	0	0	0
Bagalkot	0	0	0.5	0	0.5	0	0	0	0
Belgaum	0	0	0.33	0.33	0	0.33	0	0	0
Dharwad	0	0	0.25	0	0.25	0	0.25	0	0.25
Gadag	0	0	0	0	0	0.33	0	0.33	0.33
Koppal	0.33	0	0	0	0	0	0.33	0	0.33
Bellary	0	0	0	0	0	0.33	0.33	0.33	0

Table 4: Second order spatial weight matrix

Location	Gulbarga	Bijapur	Raichur	Bagalkot	Belgaum	Dharwad	Gadag	Koppal	Bellary
Gulbarga	0	0	0.33	0	0	0.33	0.33	0	0
Bijapur	0	0	0	0.33	0	0	0.33	0.33	0
Raichur	0	0	0	0	0	0	0.5	0.5	0
Bagalkot	0	1	0	0	0	0	0	0	0
Belgaum	0	0	0	0	0	0	0	0	1
Dharwad	0	0.5	0	0	0	0	0	0.5	0
Gadag	0.33	0.33	0.33	0	0	0	0	0	0
Koppal	0	0.33	0.33	0	0	0.33	0	0	0
Bellary	0	0	0	0	1	0	0	0	0

STARMA model fitting

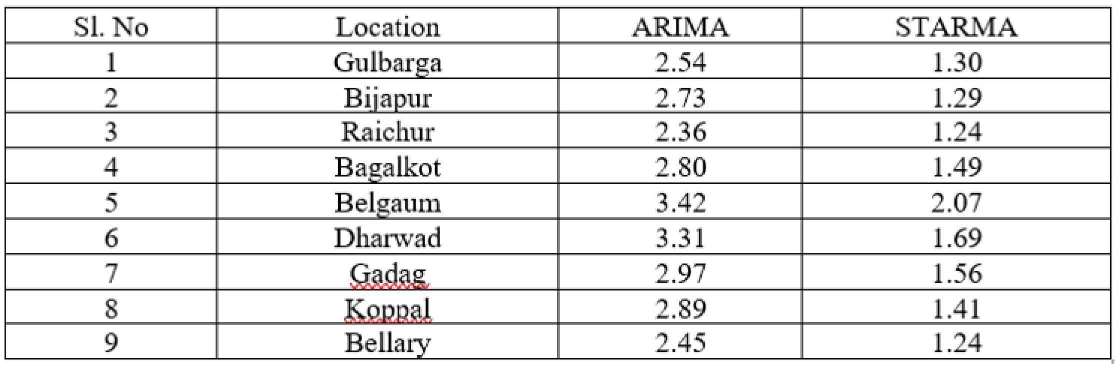
In this study, the STARMA model was estimated following the three-stage procedure outlined by Pfeiffer and Deutsch (1980a). As detailed in the methodology section, the STARMA estimation process is an extension of the Box-Jenkins ARIMA framework adapted to a spatio-temporal context. Similar to ARMA models, it involves three fundamental steps: model identification, parameter estimation, and diagnostic checking. Based on the significant spikes observed in the STACF and STPACF plots, the STARMA(1,0,1) model was selected. The parameters of this model were estimated using the maximum likelihood method, and the estimates, along with their standard errors and p-values, are provided in Table 5. These estimated parameters were incorporated into the model to generate forecasts. For diagnostic verification, the Multivariate Box-Pierce non-correlation test was employed, confirming that the residuals were uncorrelated. The forecasting performance of the model is summarized in Table 6.

Table 5: STARMA Model parameters

Spatial lag	Slag 0		Slag 1		Slag 2	
	AR	MA	AR	MA	AR	MA
Parameters	-0.66 (0.023)	0.119 (0.010)	0.171 (0.052)	0.213 (0.0157)	0.79 (0.089)	0.11 (0.116)
Probability	<0.001	<0.001	0.013	0.004	<0.001	0.010

The Multivariate Box-Pierce non-correlation test of residuals yielded a Chi-square statistic of
69.86 with a p-value of 0.31, indicating no significant autocorrelation among the residuals. Th
e values presented within parentheses denote the corresponding standard errors. To assess and
compare the forecasting performance of the ARIMA and STARMA models, the Mean Absol
ute Percentage Error (MAPE) was computed and is reported in Table 6. The results reveal tha
t the STARMA model consistently produced lower MAPE values across all study locations. T
his consistent performance suggests that the STARMA model outperforms the conventional B
ox-Jenkins ARIMA model in all cases considered.

Table 6: Modeling Performance in terms of MAPE



Sl. No	Location	ARIMA	STARMA
1	Gulbarga	2.54	1.30
2	Bijapur	2.73	1.29
3	Raichur	2.36	1.24
4	Bagalkot	2.80	1.49
5	Belgaum	3.42	2.07
6	Dharwad	3.31	1.69
7	Gadag	2.97	1.56
8	Koppal	2.89	1.41
9	Bellary	2.45	1.24

R cods to implement STARMA model

```
#install.packages("starma")
```

```
#install.packages("spdep")
rm(list = ls())
library(starma)
library(spdep)
library(forecast)
w0.mat=as.matrix(read.table(file="wo.txt",header=TRUE))
w1.mat=as.matrix(read.table(file="w1.txt",header=TRUE))
w0.mat
w1.mat
wlist =list(order0=w0.mat, order1=w1.mat)
wlist
st=as.matrix(read.table(file="ukavg.txt",header=TRUE))  # data read
st
stcor.test(st, wlist) #spatial corr
stacf(st, wlist, tlag.max=36)
stpacf(st, wlist, tlag.max=36)
#model fitting
#ar <- matrix(c(1, 1, 1, 0), 1,1)
#ma <- matrix(c(0, 1), 1, 2)
model=starma(st, wlist, ar = 1, ma = 1)
model
ab=summary(model)
ab
capture.output(ab, file = "myfile.txt")
res=model$residuals
stcor.test(res, wlist)
```

Suggested Readings

- Box, G.E.P. and Jenkins, G. (1970). Time series analysis, Forecasting and control, Holden-Day, San Francisco, CA.
- Ding, Q., X. Wang, X. Zhang, and Z. Sun. (2011). Forecasting Traffic Volume with Space–Time ARIMA Model. Advanced Materials Research, 156–57, 979–83.
- Pfeifer, P.E., and Bodily, S.E. (1990). A test of space-time ARMA modeling and forecasting with an application to real estate prices, International Journal of Forecasting, 16, 255-272.
- Pfeifer, P.E., and Deutsch, S.J. (1980). A Comparison of Estimation Procedures for the Parameters of the STAR Model. Communication in Statistics, simulation and Comput., B9(3), 255-270.
- Pfeifer, P.E., and Deutsch, S.J. (1980a). A three-stage iterative procedure for space-time modeling. Technometrics, 22(1), 35-47.
- Pfeifer, P.E., and Deutsch, S.J. (1981). Variance of the Sample-Time Autocorrelation Function of Contemporaneously Correlated Variables. SIAM Journal of Applied Mathematics, Series A, 40(1), 133-136.
- Rathod, S., Gurung,B., Singh, K.N. and Ray, M. (2018). An improved Space- ime Autoregressive Moving Average (STARMA) model for Modelling and Forecasting of Spatio-Temporal time-series data. JISAS.

Vector Autoregressive Model

Prabhat Kumar, Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar
ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115
Email: prabhatkkv@gmail.com

1. Introduction

The Vector Autoregressive (VAR) model is a powerful multivariate time series modeling technique that captures the linear interdependencies among multiple time-dependent variables. Unlike univariate time series models that handle a single variable, VAR models are particularly useful when the objective is to model and forecast more than one variable simultaneously, especially when these variables influence each other. In a VAR model, each variable in the system is expressed as a linear function of its own past values and the past values of all other variables in the system. This makes VAR especially suitable for modeling complex systems in which feedback among variables is present, such as in economics, finance, or agriculture.

2. Basic Concepts

The VAR model was introduced by Christopher Sims in 1980 as an alternative to the traditional structural economic models that imposed strong theoretical restrictions. One of the key features of the VAR model is that it treats all variables in the system as endogenous by default, meaning that it does not impose any distinction between dependent (endogenous) and independent (exogenous) variables initially. This makes the model flexible and data-driven, allowing the interrelationships among variables to emerge naturally through estimation. Because of this property, VAR is especially useful in macroeconomic modeling, financial time series analysis, and agricultural systems where multiple variables are mutually influencing each other over time.

3. Mathematical Formulation

For two variables (Y_1 and Y_2), a VAR(1) model looks like:

$$Y_{1t} = a_{10} + a_{11}Y_{1(t-1)} + a_{12}Y_{2(t-1)} + e_{1t}$$

$$Y_{2t} = a_{20} + a_{21}Y_{1(t-1)} + a_{22}Y_{2(t-1)} + e_{2t}$$

Here, Y_{1t} and Y_{2t} represent the values of the two variables at time ttt, the aija_{ij}aij coefficients are parameters to be estimated, and e_{1t}, e_{2t} are white noise error terms. The model shows how each variable depends not only on its own lagged values but also on the lagged values of the other variable.

General VAR(p) Model (for k variables):

Let $Y_t = (y_{1t}, y_{2t}, \ldots, y_{nt})$ denote an $(n \times 1)$ vector of time series variables. The basic p-lag vector autoregressive VAR (p) model has the form:

$$Y_t = A + B_1 Y_{t-1} + B_2 Y_{t-2} +, \ldots , B_p Y_{k-p} + \in_t \tag{7}$$

where, A is $(n \times 1)$ vector of intercepts, B_i (i=1, 2, …, p) is $k \times k$ matrices of parameters and $\in_t \sim iidN(0, \Sigma)$ (Lama et al. 2016). The number of parameters to be estimated in the VAR model is $k(1 + kp)$which increases with the number of variables (k) and number of lags (p).

Illustrative Example:

Here we have two variables:

- Y_1: Delhi Tomato Price
- Y_2: Lucknow Tomato Price

A VAR(1) model would look like:

$$Y_{1t} = a_{10} + a_{11} Y_{1(t-1)} + a_{12} Y_{2(t-1)} + e_{1t}$$

$$Y_{2t} = a_{20} + a_{21} Y_{1(t-1)} + a_{22} Y_{2(t-1)} + e_{2t}$$

Here, both variables are **explained by each other's past**, without assuming which one is exogenous.

In this system, both the Tomato prices are influenced by their own past values and by each other's past values. Importantly, the model does not assume both causes price or vice versa in advance; instead, it lets the data determine the direction and strength of the interrelationships.

4. Steps in Building a VAR Model

Building a VAR model involves a structured sequence of steps to ensure reliable estimation and forecasting. The first and most critical step is to check whether the time series data are stationary, as VAR models require stationary input. This is typically assessed using unit root tests such as the Augmented Dickey-Fuller (ADF) test. If any series is found to be non-stationary, it must be transformed—most commonly through differencing—to achieve stationarity.

Once stationarity is ensured, the next step is to determine the optimal lag length for the model. This is crucial because including too few lags may omit important dynamics, while too many can lead to overfitting. Lag selection is guided by information criteria such as the Akaike Information Criterion (AIC), the Bayesian Information Criterion (BIC), or the Hannan-Quinn Criterion (HQIC), which balance model fit with complexity. With the lag length decided, the VAR model is then estimated using Ordinary Least Squares (OLS) method. Each equation in the system is estimated separately, taking advantage of the fact that OLS remains efficient in this setup due to the identical regressors across equations.

Finally, after the model has been estimated, it can be used to forecast future values of the variables. VAR models are particularly valuable when the time series under study influence each other, as they can capture and utilize these interdependencies in the forecasting process.

5. Why Use VAR?

VAR models are especially useful because they can capture the complex dynamic interdependencies among multiple time series. They are particularly well-suited for forecasting, policy analysis, and simulations. In applied research, VAR has been widely used in macroeconomic modeling (e.g., studying the relationship between inflation, interest rate, and GDP), in financial markets (e.g., modeling stock prices and returns), and in agriculture (e.g., analyzing the relationship between rainfall, fertilizer use, and crop yields). VAR provides a framework where researchers can model systems of equations without requiring strong assumptions about which variables are exogenous or endogenous.

6. Limitations

Despite their flexibility, VAR models come with certain limitations. One major issue is that the model requires all variables to be stationary. Non-stationary series must be transformed, which may lead to loss of long-run relationships unless a cointegrated VAR (like VECM) is used. Another problem is the large number of parameters, especially as the number of variables and lag length increases. This can lead to overfitting, especially in small samples. Additionally, VAR models do not imply causality — just because one variable helps predict another does not mean it causes it. Therefore, further testing such as Granger causality is necessary to establish directional relationships.

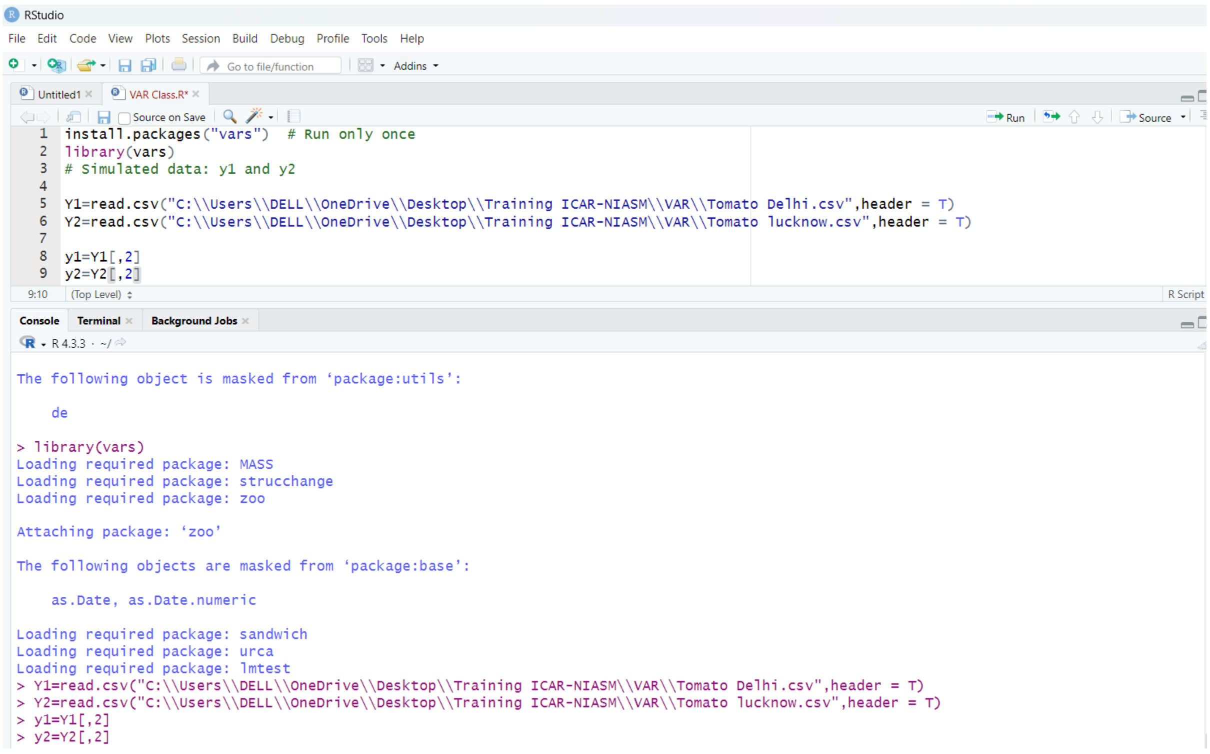
7. Tools for Implementation

VAR models can be implemented using various statistical software. In R, the vars package is commonly used, which includes functions like VAR() for model, and for predict() for forecasting. In Python, the statsmodels.tsa.api.VAR module provides similar functionality. Both tools allow for comprehensive analysis and visualization of multivariate time series using the VAR framework.

R Practical

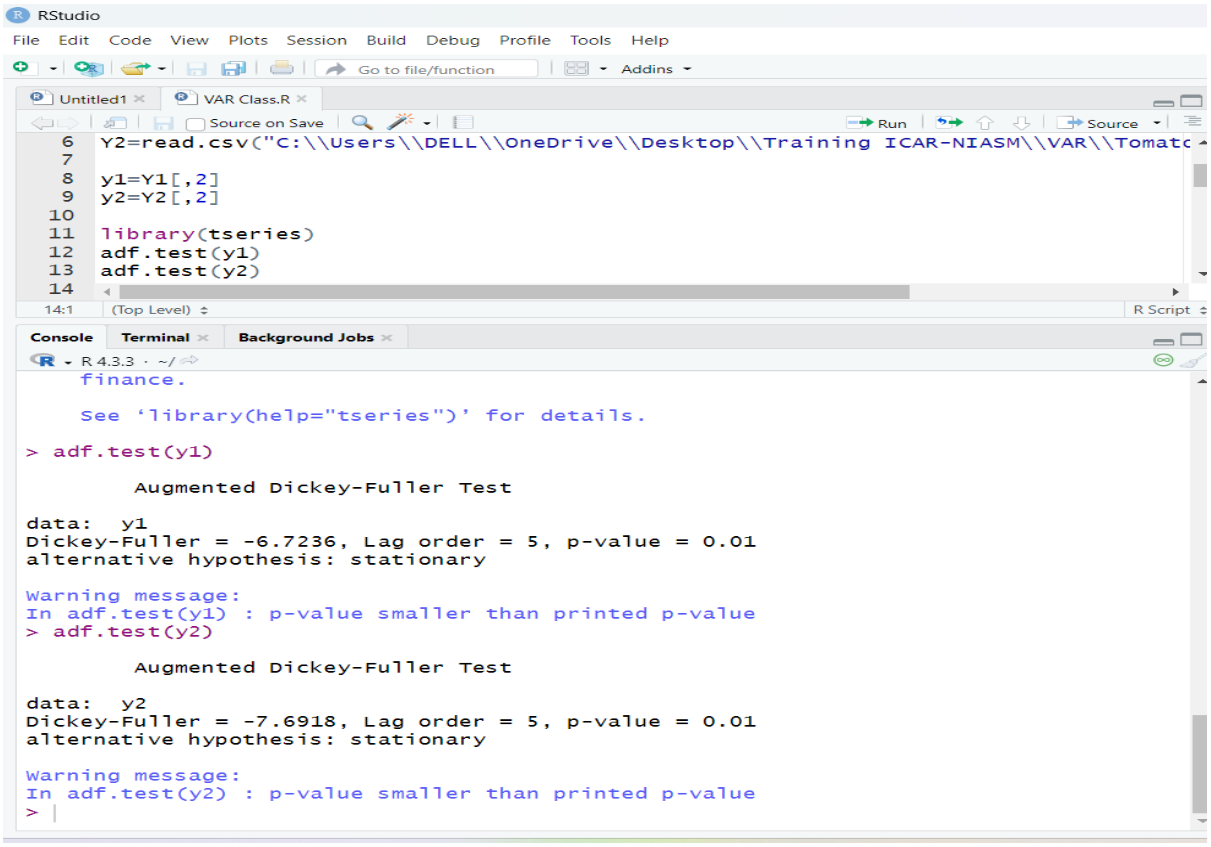
Data Used:

- Source: Simulated or actual tomato price data from two Indian cities:
    - Tomato_Delhi.csv
    - Tomato_Lucknow.csv
- Structure: Each CSV contains price data (assumed to be in column 2).



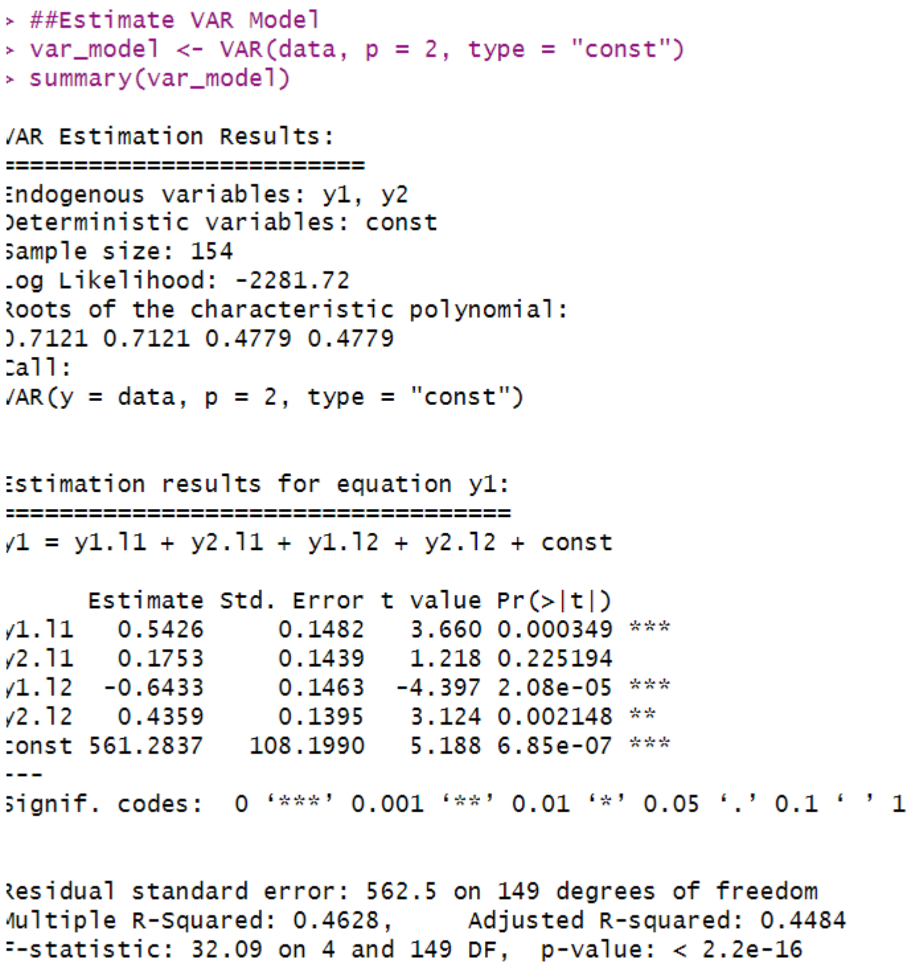
```
install.packages("vars")  # Run only once
library(vars)
# Simulated data: y1 and y2

Y1=read.csv("C:\\Users\\DELL\\OneDrive\\Desktop\\Training ICAR-NIASM\\VAR\\Tomato Delhi.csv",header = T)
Y2=read.csv("C:\\Users\\DELL\\OneDrive\\Desktop\\Training ICAR-NIASM\\VAR\\Tomato lucknow.csv",header = T)

y1=Y1[,2]
y2=Y2[,2]
```

```
The following object is masked from 'package:utils':

    de

> library(vars)
Loading required package: MASS
Loading required package: strucchange
Loading required package: zoo

Attaching package: 'zoo'

The following objects are masked from 'package:base':

    as.Date, as.Date.numeric

Loading required package: sandwich
Loading required package: urca
Loading required package: lmtest
> Y1=read.csv("C:\\Users\\DELL\\OneDrive\\Desktop\\Training ICAR-NIASM\\VAR\\Tomato Delhi.csv",header = T)
> Y2=read.csv("C:\\Users\\DELL\\OneDrive\\Desktop\\Training ICAR-NIASM\\VAR\\Tomato lucknow.csv",header = T)
> y1=Y1[,2]
> y2=Y2[,2]
```



```
Y2=read.csv("C:\\Users\\DELL\\OneDrive\\Desktop\\Training ICAR-NIASM\\VAR\\Tomatc

y1=Y1[,2]
y2=Y2[,2]

library(tseries)
adf.test(y1)
adf.test(y2)
```

```
    finance.

    See 'library(help="tseries")' for details.

> adf.test(y1)

        Augmented Dickey-Fuller Test

data:  y1
Dickey-Fuller = -6.7236, Lag order = 5, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(y1) : p-value smaller than printed p-value
> adf.test(y2)

        Augmented Dickey-Fuller Test

data:  y2
Dickey-Fuller = -7.6918, Lag order = 5, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(y2) : p-value smaller than printed p-value
> 
```



```
> ##Estimate VAR Model
> var_model <- VAR(data, p = 2, type = "const")
> summary(var_model)

/AR Estimation Results:
=========================
:ndogenous variables: y1, y2
)eterministic variables: const
;ample size: 154
.og Likelihood: -2281.72
Roots of the characteristic polynomial:
).7121 0.7121 0.4779 0.4779
:all:
/AR(y = data, p = 2, type = "const")


:stimation results for equation y1:
===================================
y1 = y1.l1 + y2.l1 + y1.l2 + y2.l2 + const

      Estimate Std. Error t value Pr(>|t|)
y1.l1   0.5426     0.1482   3.660 0.000349 ***
y2.l1   0.1753     0.1439   1.218 0.225194
y1.l2  -0.6433     0.1463  -4.397 2.08e-05 ***
y2.l2   0.4359     0.1395   3.124 0.002148 **
:onst 561.2837   108.1990   5.188 6.85e-07 ***
---
;ignif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Residual standard error: 562.5 on 149 degrees of freedom
Multiple R-Squared: 0.4628,     Adjusted R-squared: 0.4484
=-statistic: 32.09 on 4 and 149 DF,  p-value: < 2.2e-16
```

Stationarity Check (ADF Test):

Performed using the adf.test() from the tseries package.

Variable	ADF Test Statistic	p-value	Stationarity
y1	-6.7236	0.01	Stationary
y2	-7.6918	0.01	Stationary

Conclusion: Both series are stationary at 1% level, meaning no differencing is needed.

VAR Estimation Results for y1 (Delhi):

Equation:

y1_t = β_1·y1_{t-1} + β_2·y2_{t-1} + β_3·y1_{t-2} + β_4·y2_{t-2} + constant

Coefficient	**Estimate**	**p-value**	**Significance**
y1.1	0.5426	0.000349	***
y2.1	0.1753	0.2252	Not Sig.
y1.2	-0.6493	2.08e-05	***
y2.2	0.4359	0.0021	**
const	561.2837	6.85e-07	***

- Adjusted R^2: 0.4484
- F-statistic: Highly significant (< 2.2e-16)
- **Interpretation***:* Prices in Delhi are significantly influenced by their own lags and Lucknow's second lag.

VAR Estimation Results for y2 (Lucknow):

Equation:
y2_t = β_1·y1_{t-1} + β_2·y2_{t-1} + β_3·y1_{t-2} + β_4·y2_{t-2} + constant

Coefficient	**Estimate**	**p-value**	**Significance**
y1.1	0.2449	0.0948	. (marginal)
y2.1	0.6669	5.48e-06	***
y1.2	-0.6802	5.14e-06	***
y2.2	0.2808	0.0423	*
const	698.3248	7.98e-10	***

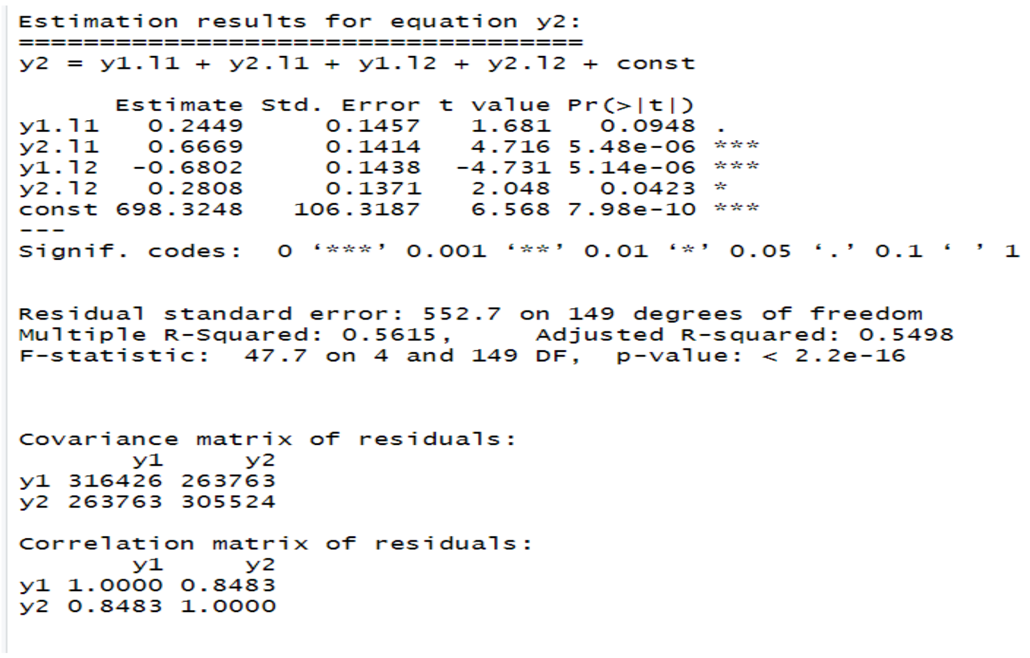
- **Adjusted R^2**: 0.5498
- **F-statistic**: < 2.2e-16
- **Interpretation***:* Lucknow prices are strongly affected by Delhi's 2nd lag and its own past values.

Residual Diagnostics:

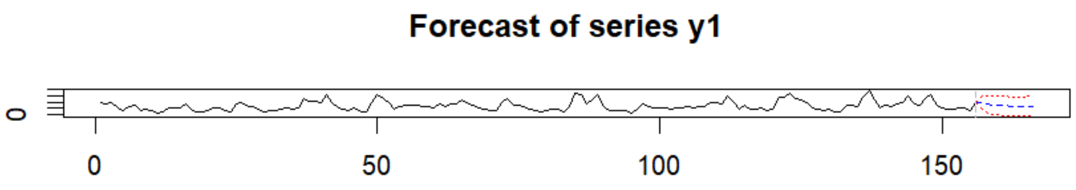
- Residual Correlation:
    - y1 and y2: 0.8483 → Strong positive correlation between model residuals.
- Covariance Matrix: Shows non-zero interaction between variables → joint modeling appropriate.

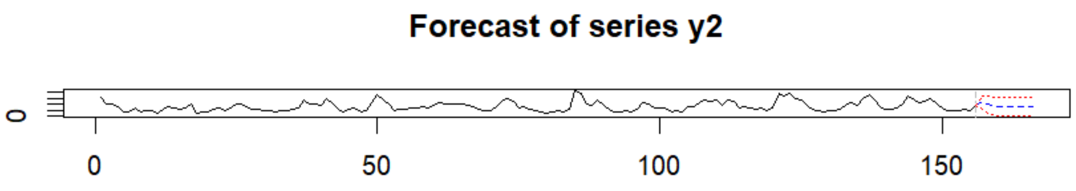
Conclusion:

- VAR modeling provides powerful insights into interconnected price behavior over space and time.
- Strong spatial linkages suggest price transmission between Delhi and Lucknow.
- This model can be extended to other crops and linked with climatic parameters to assess and forecast abiotic stress impacts.



```
Estimation results for equation y2:
====================================
y2 = y1.l1 + y2.l1 + y1.l2 + y2.l2 + const

      Estimate Std. Error t value Pr(>|t|)
y1.l1   0.2449     0.1457   1.681   0.0948 .
y2.l1   0.6669     0.1414   4.716 5.48e-06 ***
y1.l2  -0.6802     0.1438  -4.731 5.14e-06 ***
y2.l2   0.2808     0.1371   2.048   0.0423 *
const 698.3248   106.3187   6.568 7.98e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1


Residual standard error: 552.7 on 149 degrees of freedom
Multiple R-Squared: 0.5615,     Adjusted R-squared: 0.5498
F-statistic:  47.7 on 4 and 149 DF,  p-value: < 2.2e-16



Covariance matrix of residuals:
       y1     y2
y1 316426 263763
y2 263763 305524

Correlation matrix of residuals:
       y1     y2
y1 1.0000 0.8483
y2 0.8483 1.0000
```





Suggestion Reading:

- Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.
- Lama, A., Jha, G. K., Gurung, B., Paul, R. K., and Sinha, K. (2016). VAR-MGARCH models for volatility modelling of pulses prices: An Application. *J. Indian Soc. Agric. Stat*, 145-151
- Kumar, R. R., and Jha, G. K. (2017). Examining the co-movement between energy and agricultural commodity prices in India. *Journal of the Indian Society of Agricultural Statistics*, 71(3), 241-252
- Sathianandan T V. 2007. Vector time series modeling of marine fish landings in Kerala. *Journal of the Marine Biological Association of India*, 49(2): 197–205
- Yashavanth, B. S., Singh, K. N., Paul, A. K., and Paul, R. K. (2017). Forecasting prices of coffee seeds using vector autoregressive time series model. *Indian Journal of Agricultural Sciences*, 87(6), 754-758

Introduction to Functional Time Series Analysis

Roshini Priya C H[1] and Santosha Rathod[2]

[1]Professor Jayashankar Telangana Agricultural University, Hyderabad -500030
[2]ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115
Email: roshini7sn@gmail.com

1. Introduction

A time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Functional data often arise from measurements obtained by separating an almost continuous time record into natural consecutive intervals, for example days. Functional time series consist of random functions observed at regular time intervals. Examples of functional time series whose sample elements are recorded sequentially over time are frequently encountered in many disciplines. For example, in demography and epidemiology, researchers observe age-specific mortality rate or fertility rate curves over many years, and are interested in forecasting future mortality/fertility rate curves. Functional time series consist of random functions observed at regular time intervals. The functions thus obtained from a timeseries $\{X_k, k \epsilon \mathrm{Z}\}$ where each X_k is a (random) function $X_k(t)$, $t \epsilon [a, b]$. Functional data pertains to datasets in which each observation represents a function defined over a continuous domain. FDA deals with data that are naturally viewed as functions, such as curves or surfaces, and is crucial for analysing high-dimensional data efficiently.

A functional time series (FTS) arises when functional objects (curves) are collected sequentially over time. Functional time series (FTS) analysis is an emerging area in statistics designed for data where each observation is a function, such as daily or seasonal curves of crop yields, rather than a single value. Functional data analysis (FDA) arises naturally in this context to exploit the information recorded over a continuum such as time or space. In contrast to conventional scalar or multivariate data, functional data retains the complete functional characteristics of the observations, encompassing their shape, evolution, and variability. FDA is applied widely across fields such as medicine, finance, agriculture, and engineering, enabling more accurate predictions and insights by leveraging the functional nature of the data. Functional data analysis also involves estimating functional parameters describing data that are not themselves functional, and estimating a probability density function for rainfall data is an example. A theme in functional data analysis is the use of information in derivatives.

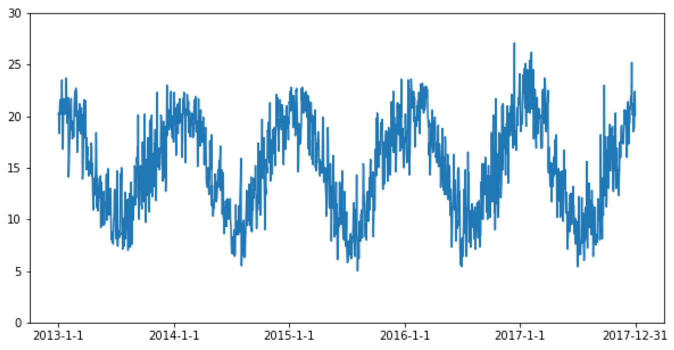
Why Use FTS?

- Handles high-dimensional data efficiently
- Reduces noise and enhances interpretability
- Better suited for seasonal and long-term forecasting

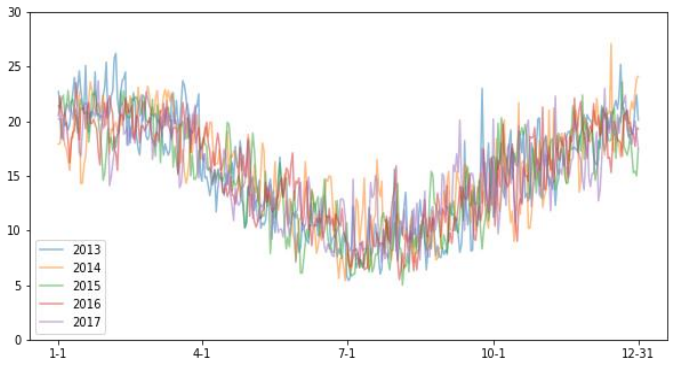
Functional Data Analysis (FDA) enables to model, analyse, and interpret continuous variation, uncovering patterns and relationships that traditional methods may miss. By representing data as functions, FDA allows for operations like differentiation, integration, and smoothing, which facilitate deeper exploration of data structure and variation. By extending time series analysis principles to functional data, FTS incorporates temporal dependencies and patterns that manifest both within and between these observed curves. Among the numerous existing contributions, one-step-ahead functional time series forecasting, that is, one-step-ahead prediction of a curve-valued time series, has been applied in several practical studies. Like many classical methods for time series forecasting, for example, those based on the Auto Regressive Integrated Moving Average (ARIMA) family models, the benchmark methods for FTS forecasting are based on fitting some statistical model of some approximated data generation. The most common is to fit ARIMA or Variational Auto Regressive (VAR) models to the Functional Principal Components Scores (FPC) of the functions, see Functional Auto Regressive (FAR) model.

A FTS is considered (weakly) stationary if it satisfies two conditions: (i) the mean function, denoted as μ_t, remains independent of time, that is, $\mu_t = \mu$. (ii) The autocovariance operator at lag h, denoted as C_h, solely depends on the time distance and is represented as $C_h := C_{t,t+h} = C_{0,h}$.

Model using multivariate time series with as many dimensions as observations per year, such that every observation of the time series corresponds to the data collected during the entire year: Y(t) = (X(t,1), …, X(t,d)). But the dimension is very high (365 dimensions to be precise). We can reduce the dimension by reducing the frequency of the observations but we lose information. A final approach is to consider the data as a functional time series Y(t, x), where we have a function Y(.,x) for every year t with Y(t,i) = X(t,i). In this case, the yearly temperature is viewed as a function in time and every observation corresponds to a function, which describes the yearly temperature.



Daily temperature in Sydney from 2013 to 2017; x-axis: time, y-axis: temperature in Celsius;



Temperature in Sydney for different years; x-axis: time, y-axis: temperature in Celsius;

A wide array of modelling approaches has been developed for FTS, reflecting the complexity and richness of functional data. These models are typically classified into several categories:

- **Parametric models**, such as functional autoregressive (FAR) and functional moving average (FMA) models, which extend classical time series models to the functional domain by capturing linear temporal dependence between curves.
- **Nonparametric models**, including kernel regression and smoothing methods, which make minimal assumptions about the underlying process and rely on data-driven techniques to capture complex dependencies.
- **Semiparametric models**, which combine linear and nonlinear components to flexibly model both structured and unstructured variation in the data.
- **Dimension reduction and score-based models**, where functional principal component analysis (FPCA) is used to extract key features (scores) from the curves, and these scores are then forecasted using standard time series methods.
- **Multivariate and grouped FTS models**, which handle multiple related functional time series simultaneously, capturing both within- and between-group dependencies

Exploring Variability in Functional Data:

Any data analysis begins with the basics: estimating means and standard deviations.

Functional Principal Components Analysis:

Goal of FPCA:

- Reduce the infinite-dimensional functional data into a finite number of components.
- Find a set of orthogonal basis functions (eigenfunctions) that capture the most variability.

Optimal choice of the number of components K needed for the approximation which gives the best trade-off between bias and variance.

There are several ad hoc procedures that are routinely applied in multivariate PCA, such as the scree plot or the fraction of variance explained by the first few PC components, which can be directly extended to the functional setting.

Basis Function Systems for Constructing Functions

We use a set of functional building blocks $\emptyset(k)$; $k = 1; : : : ;K$ called *basis functions*,

which are combined linearly. That is, a function $x(t)$ defined in this way is expressed

in mathematical notation as

$$x(t) = \sum_{k=1}^{K} c_k \ \emptyset_t(t) = c'\,\emptyset(t), \qquad \ldots\ldots\ldots\ldots\ldots(1)$$

and called a *basis function expansion*. The parameters $c_1, c_2, \ldots, c_K$ are the *coefficients*

of the expansion. We often want to consider a sample of N functions,

$x(t) = \sum_{k=1}^{K} c_{1k} \ \emptyset_k(t)$, $i=1,2,3,\ldots,N$ and in this case matrix notation for (1) becomes

$$\mathrm{x}(t) = \mathbf{C}\emptyset(t);$$

where $\mathrm{x}(t)$ is a vector of length N containing the functions $x_i(t)$, and the coefficient

matrix C has N rows K columns.

The notion of a basis system is hardly new; a polynomial such as $x(t) = 18t^4 - 2t^3 + \sqrt{17}t^2 + \pi/2$ is just such a linear combination of the *monomial* basis functions$1; t; t^2; t^3;$ and t^4 with coefficients $\pi/2$, 0, $\sqrt{17}$, -2, and 18, respectively. Within the monomial basis system, the single basis function 1 is often needed by itself, and it the called as *constant* basis system. But polynomials are of limited usefulness when complex functional shapes are required. Therefore we need heavy lifting two basis systems: *Splines* and *Fourier Series*. These two systems often need to be supplemented by the *constant* and *monomial* basis systems.

For each basis system we need a function in either R or MATLAB to define a specific set of K basis functions $\emptyset_k$. These are the create functions. Here are the calling statements of the create functions in R that set up constant, monomial, Fourier and spline basis systems, omitting arguments that tend only to be used now and then as well as default values:

basisobj = create.constant.basis(rangeval)

basisobj = create.monomial.basis(rangeval, nbasis)

basisobj = create.fourier.basis(rangeval, nbasis, period)

basisobj = create.bspline.basis(rangeval, nbasis, norder, breaks)

rangeval argument specifies the lower and upper limits of the values of argument t and is a vector object of length 2. For example, if we need to define a basis over the unit interval [0*;*1], we would use a statement like *rangeval* = c(0,1).

The second argument *nbasis* specifies the number *K* of basis functions. It does not appear in the constant basis call because it is automatically 1.

Fourier Series for Periodic Data and Functions:

Many functions are required to repeat themselves over a certain period *T*, as would be required for expressing a seasonal trend in a long time series. Fourier basis functions are arranged in successive sine/cosine pairs, with both arguments within any pair being multiplied by one of the integers 1*, 2 ,3,….*up to some upper limit *m*. If the series contains both elements of each pair, as is usual, the number of basis functions is *K* = 1+2*m.*

Only two pieces of information are required to define a Fourier basis system:

- the number of basis functions *K* and
- the period *T*

Example: daybasis65 = create.fourier.basis(c(0,365), 65)

Note that these function calls use the default of *T* =365, but if we wanted to specify some other period *T*, we would use   create.fourier.basis(c(0,365), 65, T)

Spline Series for Nonperiodic Data and Functions:

Splines are piecewise polynomials. Spline bases are more flexible and therefore more complicated than finite Fourier series. They are defined by the range of validity, the knots, and the order. There are many different kinds of splines.

Break Points and Knots:

Splines are constructed by dividing the interval of observation into subintervals, with boundaries at points called *break points* or simply *breaks*. Over any subinterval, the spline function is a polynomial of fixed degree or order, but the nature of the polynomial changes as one passes into the next subinterval. The term *degree* to refer the highest power in the polynomial. The *order* of a polynomial is one higher than its degree. For example, a straight line is defined by a polynomial of degree one since its highest power is one, but is of order two because it also has a constant term. A spline basis is actually defined in terms of a set of *knots*.

Order and Degree:

Order four splines are often used, consisting of cubic polynomial segments (degree three), and the single knot per break point makes the function values and first and second derivative values match.

To summarize, spline basis systems are defined by the following:

- the break points defining subintervals,
- the degree or order of the polynomial segments, and
- the sequence of knots.

The number K of basis functions in a spline basis system is determined by the relation

number of basis functions = order + number of interior knots

By interior here we mean only knots that are placed at break points which are not either at the beginning or end of the domain of definition of the function.

Example:

13 order four B-splines corresponding to nine equally spaced interior knots over the interval [0;10], constructed in R by the command

```
splinebasis = create.bspline.basis(c(0,10), 13)
```

The B-spline basis system has a property that is often useful: the sum of the B-spline basis function values at any point t is equal to one.

Other Basis Systems

The *exponential basis*, a set of exponential functions, exp(*akt*), each with a different rate parameter , and created with function   create.exponential.basis.

²The *polygonal basis*, defining a function made up of straightline segments, and created with function   create.polygonal.basis.

The *power basis*, consisting of a sequence of possibly non integer powers and even negative powers, of an argument *t*. These bases are created with the function create.power.basis.

Methods for Functional Basis Objects:

Basis *evaluation* functions

```
basismatrix = eval.basis(tvec, mybasis)

basismatrix = eval_basis(tvec, mybasis)
```

where argument tvec is a vector of *n* argument values within the range used to define the basis, and argument mybasis is the name of the basis system that you have created. The resulting basismatrix is *n* by *K*.

Adding Coefficients to Bases to Define Functions:

Coefficient Vectors, Matrices and Arrays:

Once we have selected a basis, we have only to supply coefficients in order to define an object of the *functional data* class (with class name fd). If there are *K* basis functions, we need a coefficient vector of length *K* for each function that we wish to define. If only a single function is defined, then the coefficients are loaded into a vector of length *K* or a matrix with *K* rows and one column. If *N* functions are needed, say for a sample of functional observations of size *N*, we arrange these coefficient vectors in a *K* by *N* matrix.

Example: coefficients for mean temperature for each of the 35 weather stations organized into the 65 by 35 matrix coefmat:

```
tempfd = fd(coefmat, daybasis65)
```

Labels for Functional Data Objects:

Adding labels to functional data objects is a convenient way to supply the information needed for graphical displays.

```
fdnames = vector("list", 3)
```

Methods for Functional Data Objects:

As for the basis class, there are similar generic functions for printing, summarizing and testing for class and identity for functional data objects.

There are, in addition, some useful methods for doing arithmetic on functional data objects and carrying out various transformations. For example, we can take the sum, difference, power or pointwise product of two functions with commands like

```
fdsumobj = fdobj1 + fdobj2
fddifobj = fdobj1 - fdobj2
fdprdobj = fdobj1 * fdobj2
fdsqrobj = fdobjˆ2
```

The mean of a set of functions is achieved by a command like

```
fdmeanobj = mean(fdobj)
```

Smoothing Using Regression Analysis:

The Linear Differential Operator or Lfd Class:

The concept of a “derivative” could itself be extended by proposing linear combinations of derivatives, called *linear differential operators*.

Smoothing is supported using the Lfd class that expresses the concept of a linear differential operator. An important special case is the *harmonic acceleration* operator that we will use extensively with Fourier basis functions to smooth periodic data.

Regression Splines: Smoothing by Regression Analysis

When smoothing function x is defined as a basis function expansion (3.1), the least squares estimation problem becomes

$$\text{SSE(c)} = \sum_j^n [y_j - \sum_k^K c_k \emptyset_k(t_j)]^2 = \sum_j^n [y_j - \emptyset(t_j)c]^2$$

The least squares estimation process can be defended on the grounds that it tends to give nearly optimal answers relative to "best" estimation methods so long as the true error distribution is fairly short-tailed and departures from the other assumptions are reasonably mild.

Data Smoothing with Roughness Penalties*:*

The roughness penalty approach uses a large number of basis functions, possibly extending to one basis function per observation and even beyond, but at the same time imposing smoothness by penalizing some measure of function complexity.

Choosing a Roughness Penalty:

We define a measure of the *roughness* of the fitted curve, and then minimize a fitting criterion that trades off curve roughness against lack of data fit. Whatever roughness penalty we use, we add some multiple of it to the error sum of squares to define the compound fitting criterion.

$$F(c) = \sum_j [y_j - x(t_j)]^2 + \lambda \int [D^2 x(t)]^2 dt$$

where $x(t) = c' \emptyset(t)$. The *smoothing parameter* λ specifies the emphasis on the second term penalizing curvature relative to goodness of fit quantified in the sum of squared residuals in the first term. As λ moves from 0 upward, curvature becomes increasingly penalized. With l sufficiently large, $D^2(x)$ will be essentially 0.

Details of **fdPar** Class and **smooth.basis** Function:

The fdPar class:

fdPar(fdobj=NULL, Lfdobj=NULL, lambda=0, estimate=TRUE, penmat=NULL)

The arguments are as follows:

fdobj A functional data object, functional basis object, a functional parameter object or

a matrix. If it a matrix, it is replaced by fd(fdobj). If class(fdobj) == 'basisfd', it

is converted to an object of class fd with a coefficient matrix consisting of a single column of zeros.

Lfdobj Either a nonnegative integer or a linear differential operator object. If NULL, Lfdobj depends on fdobj[['basis']][['type']]: bspline Lfdobj = int2Lfd(max(0, norder-2)),

where norder = norder(fdobj).

fourier Lfdobj is a harmonic acceleration operator set up for the period used to define the basis. anything else Lfdobj <- int2Lfd(0)

lambda: A nonnegative real number specifying the amount of smoothing to be applied to the estimated functional parameter estimate.

penmat: A roughness penalty matrix. Including this can eliminate the need to compute this matrix over and over again in some types of calculations

Exploring the Variation:

Functional Principal Component Analysis:

In functional PCA, there is an eigenfunction associated with each eigenvalue, rather than an eigenvector. These eigenfunctions describe major variational components. Applying a rotation to them often results in a more interpretable picture of the dominant modes of variation in the functional data, without changing the total amount of common variation.

Principal component analysis is implemented in the functions pca.fd in R.

pca.fd(fdobj, nharm = 2, harmfdPar=fdPar(fdobj), centerfns = TRUE)

The first argument is a functional data object containing the functional data to be analysed, and the second specifies the number of principal components to be retained. The third argument is a functional parameter object that provides the information necessary to smooth the eigenfunctions if necessary.

Function pca.fd in R returns an object with the class name pca.fd, so that it is effectively a constructor function. Here are the named components for this class.

harmonics A functional data object for the ` harmonics or eigenfunctions x_j. values. The complete set of eigenvalues m_j. scores The matrix of scores c_{ij} on the principal components or harmonics. Varprop A vector giving the proportion $\mu_j / \sum \mu_j$ of variance explained by each eigenfunction.

meanfd A functional data object giving the mean function.

Tests on Stationarity and Independence

In time series analysis, two functional observations are considered independent if their joint probability can be expressed as a product of individual probabilities. A time series is deemed stationary when its statistical properties, such as distribution, remain unchanged over time. For functional data analysis, stationarity can be assessed using the CUSUM (Cumulative Sum) statistic, which helps in identifying weak stationarity. To test the assumption of independence, a Portmanteau-type test is commonly employed, allowing researchers to confirm or refute the null hypothesis of independence.

Testing for weak stationarity

In time series analysis, assessing stationarity is a fundamental step, as it determines whether temporal dynamics must be explicitly modeled. A stationary time series is one whose statistical properties remain constant over time, implying that its behavior is predictable and consistent. When a series is stationary, we do not need to adjust for time-dependent structural changes, thereby simplifying the modeling process.

However, stationarity is inherently difficult to assess directly, especially when working with finite samples. As a practical alternative, researchers often evaluate statistical moments—such as the mean, variance, and autocovariance structure—to serve as proxies. The underlying intuition is that if these moments remain constant over time, the distribution of the time series can be assumed to be time-invariant, and hence, the process is likely to be stationary.

Consequently, rather than testing for strict stationarity, it is common to test for weak (or second-order) stationarity, which requires that the mean, variance, and autocovariances are not functions of time. If these conditions are satisfied, the time series is considered weakly stationary and suitable for many traditional time series modeling approaches such as ARIMA, VAR, or STARMA.

For a functional time series *Y(1)(x), Y(2)(x), …, Y(n)(x)*, this translates into the hypotheses

$$H_0 : \mathbb{E}[Y(1)] = \cdots = \mathbb{E}[Y(n)] \quad \text{vs.} \quad H_1 : \mathbb{E}[Y(i)] \neq \mathbb{E}[Y(1)] \qquad (1)$$

for some $i \in \{ 2, \ldots , n \}$ and

$$H_0^{(h)} : \mathbb{E}[Y(1)Y(1+h)] = \cdots = \mathbb{E}[Y(n-h)Y(n)]$$
$$\text{vs.} \quad H_1^{(h)} : \mathbb{E}[Y(i)Y(i+h)] \neq \mathbb{E}[Y(1)Y(1+h)]$$

for some $i \in \{ 2, \ldots , n\text{-}h\}$.

The time series *Y(i)* is weakly stationary if the null hypotheses are valid for any positive integer *h*. Note that the first- and second-order moments of *Y(i)* are functions themselves. So equality of two functions depends on the function space. In the space of continuous functions, for example, two functions *f* and *g* are equal if they are equal in any point *x*, so if *f(x)=g(x)*. Contrarily, two functions are equal in the space of square-integrable functions L^2 if they coincide in almost every point (w.r.t. the Lebesgue measure).

As in the univariate scenario, we can employ the CUSUM statistic, which basically compares the average of the first with the average of the remaining observations. The (functional) CUSUM statistic is defined as

$$C(u,x) = \frac{1}{n}\sum_{i=1}^{\lfloor un \rfloor} Y(i)(x) - \frac{u}{n}\sum_{i=1}^{n} Y(i)(x).$$

Under the null hypothesis (and weak assumptions), $\sqrt{n}\, C(u, x)$ converges weakly to a centered Gaussian process *B(u, x)* with unknown covariance function in the space $L^2([0,1]^2)$ with norm $||.||_2$. Contrarily, $\sqrt{n}\, C(u, x)$ deviates to $+\infty$ or $-\infty$ under the alternative. So if $\sqrt{n}\, C(u, x)$ deviates too much from its limit *B(u, x)*, we can reject H_0.

A Portmanteau-type test

Just as with stationarity, assessing stochastic independence directly is challenging, so we rely on the autocovariance structure of the time series as a proxy to evaluate the extent of its dependence. For analytical convenience, it is assumed that the time series is both stationary and mean-centered, i.e., *E[Y(i)]=0* . This assumption can be verified using a procedure similar to the one described. The primary focus remains on autocovariances at lower lags h. In alignment with the classical Portmanteau test framework, the following hypotheses are considered:

$$H_0 : \max_{h=1}^{H} \|\mathbb{E}[[Y(1)Y(h+1)]\|_2 = 0 \quad \text{vs.} \quad H_1 : \max_{h=1}^{H} \|\mathbb{E}[[Y(1)Y(h+1)]\|_2 > 0.$$

As before, the second order moments of a functional time series are functions themselves, so we formulate the hypotheses in terms of their norms.

```
import pandas as pd
import numpy as np
from scipy.signal import correlate2d
```

```
df = pd.read_csv('Sydney.csv')  # Load data set
df.interpolate(inplace=True)     # Impute missing values
df.drop(df.columns[[0,1,6,7]],axis=1,inplace=True)   # Drop unused columns
df = df[df['Year']<2018]    # Drop observations from 2018
# Restructure data: rows correspond to different observations, columns are the different days
df = df.pivot(index='Year', columns=['Month','Day'], values='Minimum temperature (Degree C)')
df.drop((2, 29), axis=1, inplace=True)  # Drop 29th of February
df.index = df.index - min(df.index)   # Change index
```

Testing for stationarity of the mean:

In order to test for stationarity of the mean, we define three auxiliary functions to calculate the CUSUM statistic, the _L²-_norm and bootstrap replicates to approximate the quantile.

```
def calculate_cusum(X):
    n = X.shape[0]
    X_cusum = ( np.cumsum(X) - np.tensordot(np.arange(1,n+1)/n, np.sum(X), axes=0) ) / n
    return X_cusum
def calculate_l2_norm(X):
    l2_norm = np.sqrt((X**2).mean())
    return l2_norm
def generate_bootstrap_replicate(X):
    n, d = X.shape
    random_multipliers = np.random.randn(n)
    # Calculate local mean
    kernel = np.ones(2*bw+1).reshape((2*bw+1,1))
    conv_loc_mean = correlate2d(X,kernel,mode='full')[bw:-bw]
    weights = 1/np.convolve(np.ones(2*bw+1), np.ones(conv_loc_mean.shape[0]))[bw:-bw]
    local_mean = np.multiply(conv_loc_mean, weights[:, np.newaxis])
    # Calculate bootstrap replicate
    conv_arr = correlate2d(X - local_mean,np.ones(m).reshape((m,1)),mode='full')[m-1:]/
     np.sqrt(m)
    scalar_prod = np.multiply(conv_arr, random_multipliers[:, np.newaxis])
    bootstrap_replicate = scalar_prod.cumsum(axis=0)/np.sqrt(n)
    l2_norm = calculate_l2_norm(bootstrap_replicate)
    return l2_norm
 n = df.shape[0]
test_statistic = np.sqrt(n) * calculate_l2_norm(calculate_cusum(df).to_numpy())
m = 5
bw = 25
K = 1000
alpha = 0.05
```

```
bootstrap_replicates = np.zeros(K)
for k in range(K):
    bootstrap_replicates[k] = generate_bootstrap_replicate(df)
quantile_approx = np.sort(bootstrap_replicates)[round((1-alpha)*K)]
print('Test Statistic: ' + str(round(test_statistic,3)))
print('Approximated quantile: ' + str(round(quantile_approx,3)))
if test_statistic > quantile_approx:
    print('The null hypothesis can be rejected')
else:
   print('The null hypothesis cannot be rejected')
```

- ✓ Test Statistic: 2.11
- ✓ Approximated quantile: 1.84
- ✓ The null hypothesis can be rejected
- ✓ The output suggests that we can reject the null hypothesis of a constant mean function. Thus, it is unlikely that the temperature was stationary in Sydney from 1859 to 2017, which suggests a change in climate.

Forecasting Methods

Functional Auto Regressive Moving Average Method:

Research in functional data analysis has led to *functional ARIMA* (FARIMA) models, which generalise ARIMA concepts to infinite-dimensional settings. These models are mathematically more complex and typically require specialised statistical software and expertise.

Key Steps in Practice:

1. **Preprocessing:** Ensure each function is well-defined and aligned (e.g., same domain, normalisation).
2. **Dimension Reduction:** Use FPCA or similar techniques to reduce each function to a set of scalar scores.
3. **Stationarity Check:** Check and, if necessary, difference the score series to achieve stationarity.
4. **Model Fitting:** Fit ARIMA models to each principal component score series.
5. **Forecasting:** Predict future scores using the fitted ARIMA models.
6. **Reconstruction:** Combine the predicted scores with the principal components to reconstruct the forecasted function.

Approach	Description
FPCA + ARIMA on Scores	Reduce functions to scores, fit ARIMA, reconstruct functions
Nonparametric/Projection Methods	Use projections or nonparametric techniques for curve forecasting

R code for Functional ARIMA:

```
required_packages <- c("lubridate", "forecast", "fda", "fda.usc","dplyr","tidyr","readxl",
                       "ggpplot2","ftsa","funtimes","STFTS")
install_if_missing <- function(packages) {
  new_packages <- packages[!(packages %in% installed.packages()[, "Package"])]
  if(length(new_packages)) {
    install.packages(new_packages, dependencies = TRUE)
  }
}
# Install missing packages
install_if_missing(required_packages)
library(readxl)
library(dplyr)
library(tidyr)
library(lubridate)
library(fda)
library(ftsa)
library(ggplot2)
library(data.table)
library(fda.usc)
library(forecast)
library(funtimes)
library(STFTS)
# Set working directory (assuming the file is there)
setwd("C:/Users/pc/Downloads/")
# Load the Excel file and sheet
df <- read_excel("Tomato.xlsx", sheet = 1)
# Data Cleaning and Preparation for Functional Data
df$date <- as.Date(df$Date, format="%Y-%m-%d")
df_processed <- df %>%
  mutate(
    year = year(Date),
    day_of_year = yday(Date)
  ) %>%
  filter(year %in% 2018:2024) # Filter for the specific years 2018-2024
# Reshape to matrix: Years as rows, Days of Year as columns
arrival_matrix_daily <- df_processed %>%
  pivot_wider(
    id_cols = year,
    names_from = day_of_year,
    values_from = Arrivals,
    values_fn = mean, # Use mean in case of multiple entries per day/year
    names_sort = TRUE
  ) %>%
  arrange(year)
rownames(arrival_matrix_daily) <- arrival_matrix_daily$year
arrival_matrix_daily <- arrival_matrix_daily %>%
  dplyr::select(-year) %>%
  as.matrix()
print("Dimensions of initial arrival_matrix_daily (Years x Days):")
print(dim(arrival_matrix_daily))
# Likely 7x366 if 2020 or 2024 is present
# Remove the last column if there's a 366th day (leap year) to make it consistent
# This assumes you want a 365-day domain for all functions
if (ncol(arrival_matrix_daily) > 365) {
  arrival_matrix_daily <- arrival_matrix_daily[, 1:365]
}
print("Dimensions after removing possible 366th day (Years x Days):")
print(dim(arrival_matrix_daily))  # Should now be 7x365
# Define evaluation points (days 1 to 365)
argvals <- 1:ncol(arrival_matrix_daily)
rangeval <- range(argvals)

# Create Fourier basis
nbasis <- 21
fourier_basis <- create.fourier.basis(rangeval = rangeval, nbasis = nbasis)

# Smooth the data to create functional data objects (7 functions)
fd_smooth <- smooth.basis(argvals = argvals, y = t(arrival_matrix_daily), fdParobj =
fourier_basis)$fd
plot(fd_smooth,main="Smoothened functions")
print("Number of functional observations after smoothing:")
print(length(fd_smooth$coefs[1,])) # Should be 7
fd_matrix <- eval.fd(eval_points, fd_smooth)
print("Dimensions of evaluated functional data matrix (Days x Years):")
print(dim(fd_matrix)) # Should be 365x7
fts_obj <- fts(eval_points, y = fd_matrix)
print("Number of functional observations in fts object:")
print(dim(fts_obj$y)[2])
spatial_diff2_matrix <- apply(fd_matrix, 2, diff, differences = 1)
print(dim(spatial_diff2_matrix))
spatial_argvals <- argvals[-(1:2)] # Remove the first two points
result_spatial_diff2 <- T_stationary(
  sample = spatial_diff2_matrix, # Pass the 363x7 matrix
  L = 21,              # Number of basis functions (adjust based on 363 domain?)
  J = 500,             # Truncation level (adjust based on 363 domain?)
  MC_rep = 1000,        # Number of Monte Carlo replications
  cumulative_var = 0.90,# Variance explained for dimension reduction within
  Ker1 = FALSE,
  Ker2 = TRUE,
  h = ncol(spatial_diff2_matrix)^0.5, # Default h calculation is correct for N=7
  pivotal = FALSE,
  use_table = FALSE,
  significance = "5%"
)
cat("\nStationarity test p-value for the SECOND SPATIAL DIFFERENCE:",
result_spatial_diff2$p.value, "\n")
class(spatial_diff2_matrix) #"matrix" "array"
train_matrix <- spatial_diff2_matrix[, 1:6]  # 363 x 6
test_matrix  <- spatial_diff2_matrix[, 7, drop = FALSE]
spatial_argvals <- 1:nrow(spatial_diff2_matrix)  # 1:363
fts_train <- fts(spatial_argvals, y = train_matrix)
fts_test  <- fts(spatial_argvals, y = test_matrix)
# Fit model to training data
fit_model <- ftsm(fts_train)
# Forecast the next curve (test year)
fc <- forecast(fit_model, h = 1,method = "arima")
```

Conclusion:

In the present study, relevant techniques were applied to develop a functional ARIMA model for forecasting Tomato Arrivals. The comparison study revealed that FARIMA outperforms classical ARIMA model in forecasting Tomato Arrivals.

References:

- ✓ Hyndman, R.J. and Ullah, M.S. 2007. Robust forecasting of mortality and fertility rates. A functional data approach. *Computational Statistics & Data Analysis*. *51*(10):4942-4956.
- ✓ Ramsay, J.O. and Silverman, B.W. (2005). *Functional Data Analysis*. Springer.
- ✓ Ramsay, J.O., Hooker, G. and Graves, S. (2009) *Functional Data Analysis with R and MATLAB* Springer.
- ✓ Kokoszka, P. and Reimherr, M. (2017). *Introduction to Functional Data Analysis*.

Trend Impact Analyss and its Application

Mrinmoy Ray[1], Ramasubramanian V[2], Santosha Rathod[3], Gayathri Chitikela[4], Ponnaganti Navyasree[3]
1-AKMU, ICAR-Indian Agricultural Research Institute, New Delhi
2-ICAR-National Academy of Agricultural Research management, Hyderabad
3-ICAR-National Institute of Abiotic Stress Management, Baramati
4-Professor Jayashankar Telangana Agricultural University, Hyderabad, India
Email: mrinmoy4848@gmail.com

Introduction

Trend Impact Analysis (TIA) is an advanced forecasting technique that extends traditional time series analysis by explicitly accounting for the potential effects of unexpected or external events, known as interventions on an existing trend. Unlike purely statistical extrapolation methods, which assume that past patterns will continue unchanged, trend impact analysis recognizes that real-world time series are often disrupted by unforeseen forces such as policy changes, economic shocks, technological innovations, natural disasters, pandemics, or other rare but influential events. In agricultural and environmental sciences, trend impact analysis is especially important because external interventions; like the introduction of a new crop variety, sudden pest outbreaks, extreme weather events, or government policy reforms can dramatically alter the trajectory of production, prices, or yields. By combining quantitative time series models with structured expert judgment or explicit intervention variables, TIA provides a practical way to integrate both historical data and anticipated future disruptions into forecasting. The core idea of trend impact analysis is to identify the point of intervention, measure its effect on the mean level or trend of the series, and model the pattern of this impact over time. This can be achieved through parametric statistical methods such as the ARIMA Intervention Model, where the intervention is built directly into the ARIMA framework by adding an intervention component with appropriate indicator variables. Depending on the nature of the event, the effect may be immediate and temporary (pulse), sudden and permanent (step), or gradual but increasing (ramp).

Trend impact analysis is widely used in agricultural research, social sciences, economics, and policy planning to assess the consequences of unique events on production, market dynamics, supply chains, or resource use. It not only provides more realistic forecasts but also helps decision-makers understand possible future scenarios under different assumptions. In modern applications, trend impact models can also be combined with machine learning techniques, such as Artificial Neural Networks (ANNs) and hybrid approaches, to capture non-linear patterns and complex interactions when interventions occur. This lecture note introduces the ARIMA Intervention Model, explains its components, types of interventions, coding of indicator variables, and demonstrates how it can be implemented using open-source tools like the forecast package in R. It also touches on advanced extensions, such as the NARX (Nonlinear Autoregressive Model with Exogenous Inputs), which integrates intervention effects into neural network frameworks for more flexible modeling of dynamic systems impacted by external forces.

Trend Impact Analysis plays an important role in understanding and forecasting climatic variables that are often disrupted by unexpected natural events or human activities. Climate time series such as rainfall, temperature, drought indices, or humidity can show abrupt changes due to phenomena like extreme weather events, major floods, prolonged droughts, or policy actions like large-scale afforestation or emission reduction programs. By applying the ARIMA Intervention Model, researchers can identify when such interventions or events occur and measure how they shift the mean level or trend of a climatic variable over time. For more complex and nonlinear climate dynamics, the NARX (Nonlinear Autoregressive Model with Exogenous Inputs) is useful for modeling situations where the impact of an external factor unfolds in a nonlinear way. For example, gradual land use change, deforestation, or changes in irrigation patterns may have delayed and nonlinear effects on local temperature and rainfall. By combining lagged climate variables with external drivers, the NARX model captures these intricate relationships, helping researchers develop better forecasts and design climate adaptation strategies.

ARIMA Intervention Model

ARIMA Intervention model was developed by Box and Tiao (1975). Time series intervention analysis is an application of modelling procedures for incorporating the effect of exogeneous forces or interventions in time series analysis. This intervention can be government policies, strikes, earthquakes, price changes, floods, pandemic and other irregular events. It causes unusual changes in time series. So, simply we can say that intervention analysis in time series refers to the analysis of how mean level of a series change after an intervention. An intervention model is given by

$$Y_t = \frac{\omega(B)}{\delta(B)} B^b I_t + \frac{\theta(B)}{\phi(B)} \varepsilon_t$$

$Y_t =$ [Intervention component] * I_t + ARIMA model

Where Y_t =dependent variable

I_t = indicator variable coded according to the type of intervention.

$\delta(B) = 1+ \delta_1 B + \ldots \delta_r B^r$ – slope parameter.

$\omega(B)= \omega_0+ \omega_1 B + \ldots \omega_s B^s$ – impact parameter.

$\phi(B) = 1- \phi_1 B- \phi_2 B^2 - \ldots - \phi_p B^p$- Autoregressive parameter.

$\theta(B) = 1- \theta_1 B- \theta_2 B^2 - \ldots - \theta_q B^q$ – Moving average parameter.

b = delay parameter, B=Backshift operator i.e. $B^a Y_t = Y_{t-a}$, ε_t = White noise or error term.

Types of Intervention

Time series interventions are broadly classified as step intervention, pulse/point intervention and ramp intervention based on nature and duration of interventions effects.

Step intervention:

It happens at a certain point of time and continues to exists in the subsequent time periods. The step intervention’s impact may remain constant over time, or it may increase or decrease. this type of intervention occurs in agriculture when a new variety, pesticide, or economic policy is introduced.

Pulse Intervention

It happens only for a short period of time, but the impact of these type of intervention can last only for that time period or may last for a longer period of time. For example; in agriculture these types of interventions are seen where there is an extreme drought, flood, or insect-pest infestation.

Ramp Intervention

It happens at a specific point in time and continues to exist with rising severity in the subsequent time periods. The impact of this action will continue to increase over time. For example, the price of agricultural commodity increases over a period of time.

Indicator variables:

The range of values that an intervention variable can take is generally determined by the type of intervention. For step intervention

$I_t = 0, t < T', 1, t \geq T'$

Where T' is the time of intervention when it is first occurred.

For Pulse intervention

$I_t = 0, t \neq T'$

$1, t = T'$

For Ramp intervention

$I_t = 0, t < T'$

$t\text{-}T'+1, t \geq T'$

The Table 3.3 shows an example of indicator coding for types of interventions, assuming that the intervention took place at 4th time point. Fitting the intervention model follows the same three steps as the ARIMA model *i.e.* identification, estimation, diagnostic checking.

The "forecast" package (Hyndman *et al*. 2008) in R software was used to build for ARIMA intervention model. The intervention parameter indicates the change, either the impact is positive, negative or no impact due to occurrence and spread of covid -19 pandemic. A model was considered valid when all the coefficients were significant and the residuals were found to be non-autocorrelated by means of Ljung-Box test. For selection of suitable candidate models, the loglikelihood value, minimum Akaike Informative Criteria (AIC) and Bayesian Informative Criteria (BIC) were used to select the best model.

Table: Values of intervention variable under different functions

Time t	Step intervention I_t	Pulse intervention I_t	Ramp intervention I_t

1	0	0	0
2	0	0	0
3	0	0	0
4= T′	1	1	1
5	1	0	2
6	1	0	3
7	1	0	4
8	1	0	5

Neural Network Intervention Model

The Nonlinear autoregressive exogeneous model is a recurrent dynamic neural network. The model works same as Artificial Neural Networks (ANN) with exogenous variables which is discussed below. Lagged values of the intervention variable were considered as exogeneous variables. Since we have used intervention component in ANN, which is nomenclature as ANN Intervention model henceforth (Vega *et al* 2001). The classical ANN model allows making forecasts based on only past values of the forecast variable. The model assumes that future values of a variable depend on its past values, as well as on the values of past exogeneous variables. The ANN Intervention model is an extended version of the ANN model, where it includes other independent (predictor) variables called as intervention variable, the model is also referred to as the vector ANN model.

ANN forecasting models typically assume that each observed value is an unknown nonlinear function F of c lags $t_1, t_2, \ldots, t_c$, for a given univariate time series $\{x_t, t = 1,2, \ldots, n\}$, where $x_t \in R$,

$$x_t = F(x_{t-t1}, x_{t-t2}, \ldots, x_{t-tc}) + \varepsilon_t$$

Where the error ε_t is error of zero mean. Next, we assume that m interventions have been observed throughout time periods $r_1, r_2, \ldots, r_m$. Depending on the nature of the interventions., we define m auxiliary variables $\delta_1^t, \delta_2^t, \ldots, \delta_m^t$. As a result, we can investigate a nonlinear forecasting model with c lags $t_1, t_2, \ldots, t_c$ and m interventions;

$$x_t = F(x_{t-t1}, x_{t-t2}, \ldots, x_{t-tc}, \delta_1^t, \delta_2^t, \ldots, \delta_m^t) + \varepsilon_t$$

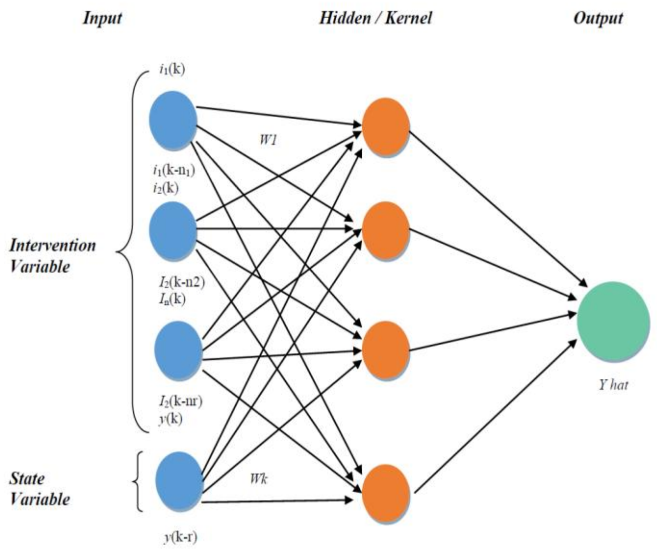


Fig.: Diagram of ANN with Intervention variable

`Illustration

The time series intervention model based TIA has been employed for envisioning crop yield scenarios for maize, potato, rice, tomato, okra, cabbage, mustard yield.

Table1: Parameters for Maize Yield Scenario

Initial Impact (percentage)	10
Maximum Impact (percentage)	25
Steady State Impact (percentage)	20
Time to maximum impact (year)	7

Time to steady state impact (year)	15
Unprecedented technology	Bt Maize

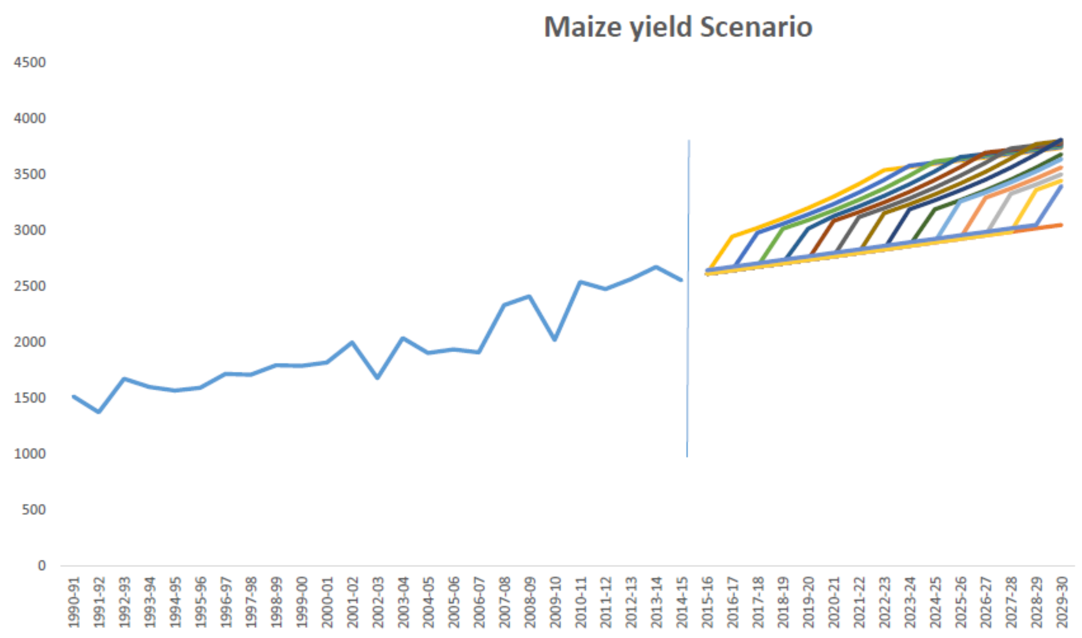


Fig : Maize yield scenarios

Table2: Parameters for Potato Yield Scenario

Initial Impact (percentage)	5
Maximum Impact (percentage)	20
Steady State Impact (percentage)	13
Time to maximum impact (year)	5

Time to steady state impact (year)	8
Unprecedented technology	Transgenic potato

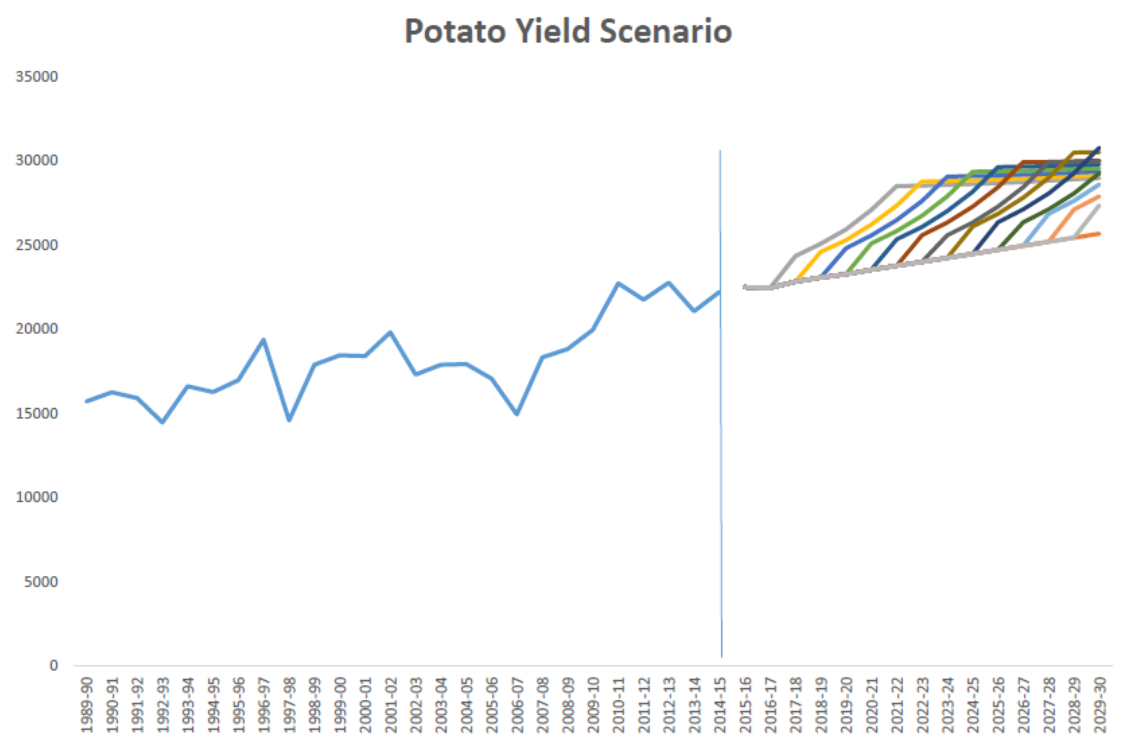


Fig 4: Potato yield scenarios

Table: Parameters for Rice Yield Scenario

Initial Impact (percentage)	7
Maximum Impact (percentage)	20
Steady State Impact (percentage)	18

Time to maximum impact (year)	7
Time to steady state impact (year)	15
Unprecedented technology	Golden Rice

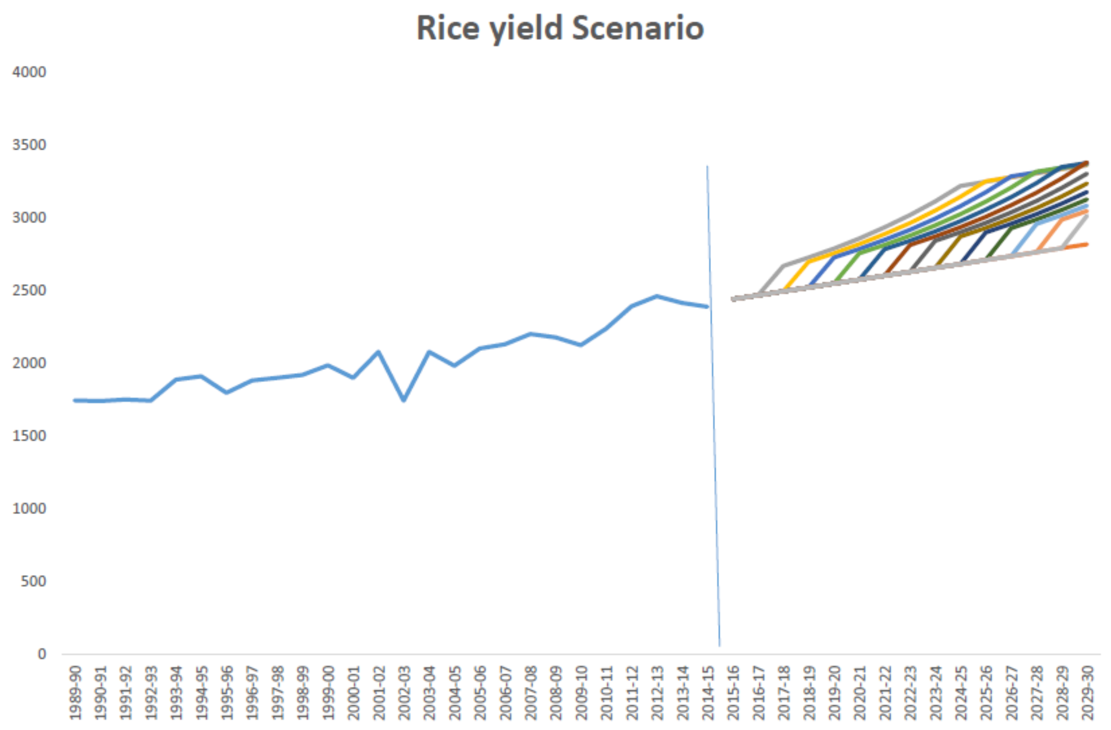


Fig.: Rice yield scenarios

Table4: Parameters for tomato Yield Scenario

Initial Impact (percentage)	6
Maximum Impact (percentage)	10
Steady State Impact (percentage)	9
Time to maximum impact (year)	5
Time to steady state impact (year)	10

Unprecedented technology	Transgenic tomato

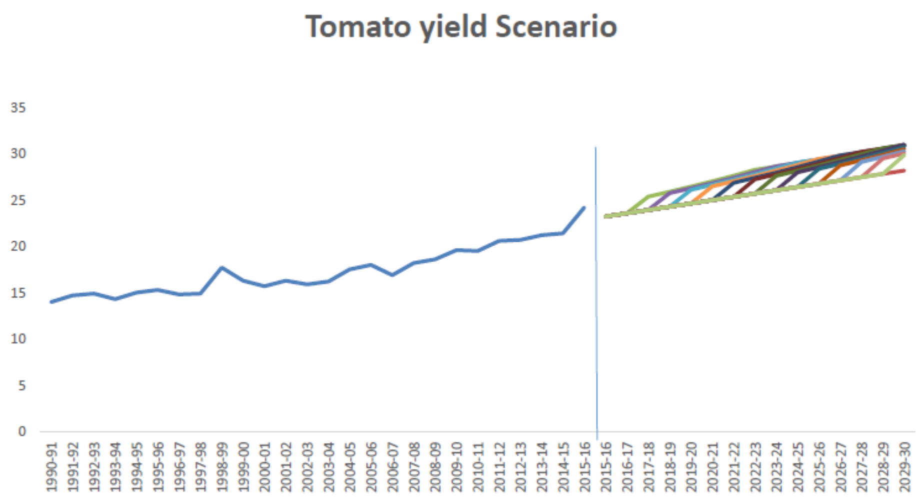


Fig.: Tomato yield scenarios

Table: Parameters for okra Yield Scenario

Initial Impact (percentage)	2
Maximum Impact (percentage)	10
Steady State Impact (percentage)	7
Time to maximum impact (year)	10
Time to steady state impact (year)	5
Unprecedented technology	Transgenic okra

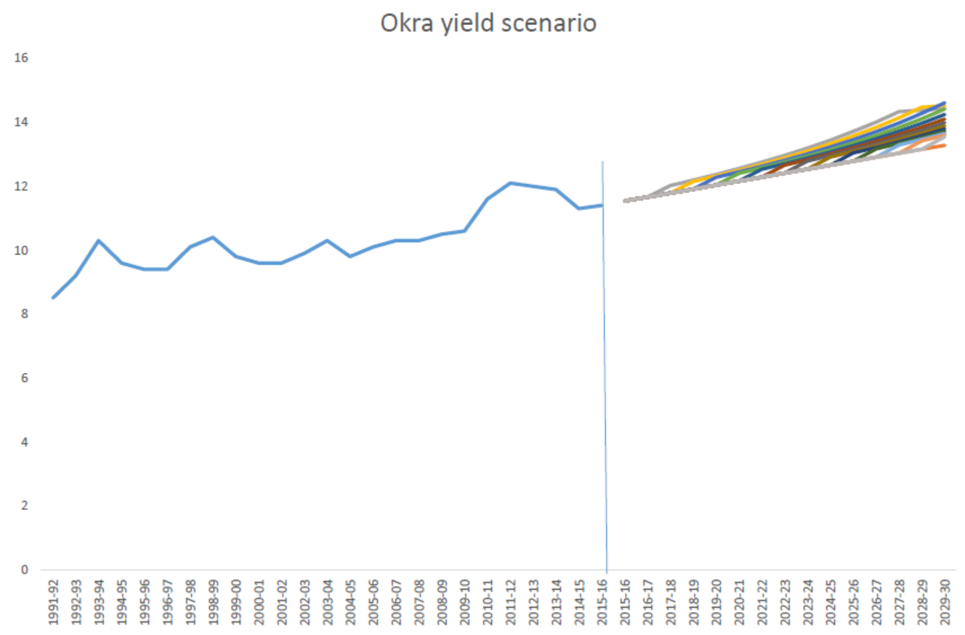


Fig .: Okra yield scenarios

Table: Parameters for Cabbage yield Scenario

Initial Impact (percentage)	4
Maximum Impact (percentage)	15
Steady State Impact (percentage)	12
Time to maximum impact (year)	8
Time to steady state impact (year)	10
Unprecedented technology	Transgenic cabbage

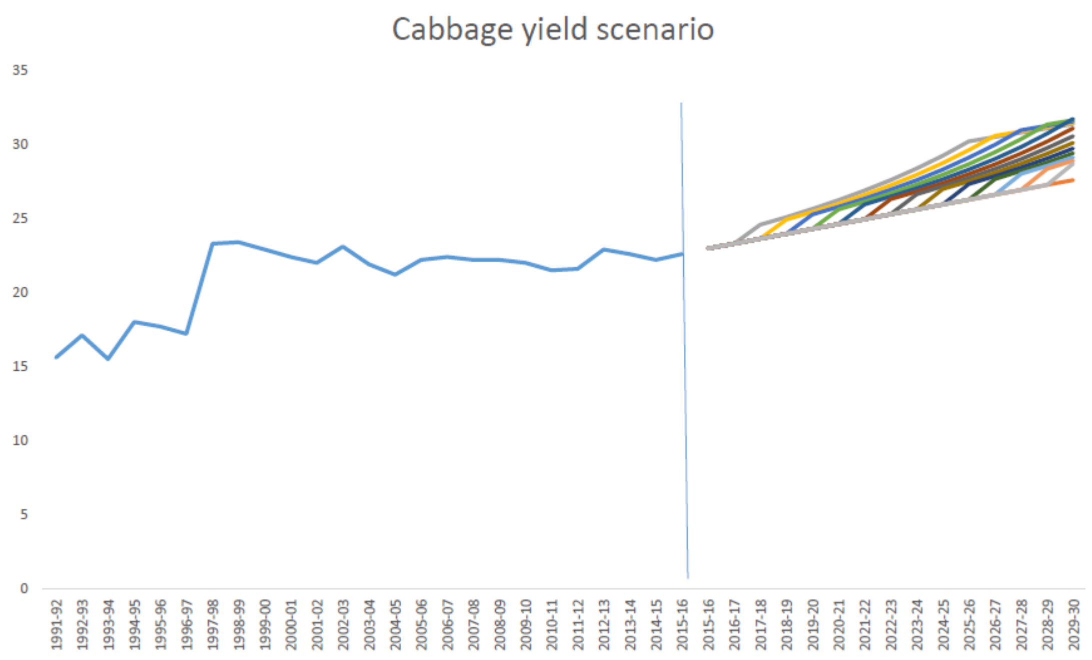


Fig: Cabbage yield scenarios

Table : Parameters for Mustard yield Scenario

Initial Impact (percentage)	30
Maximum Impact (percentage)	50
Steady State Impact (percentage)	45
Time to maximum impact (year)	3
Time to steady state impact (year)	10
Unprecedented technology	Bt Mustard

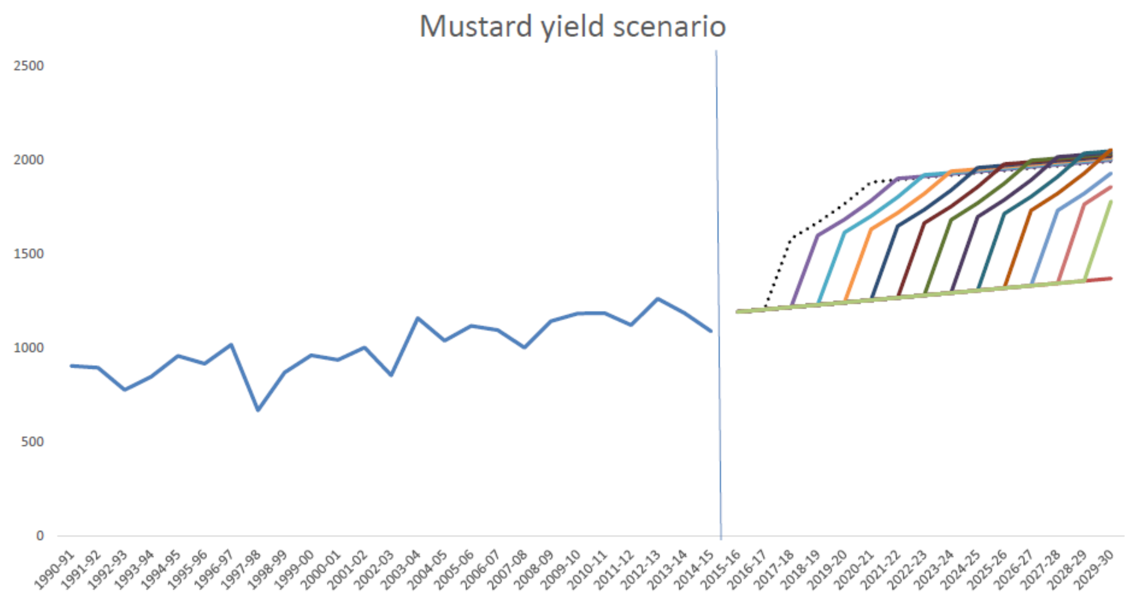


Fig 9: Mustard yield scenarios

Conclusion

In this study, by integrating Delphi, GOS tree and time series intervention-based TIA a methodology has been proposed for envisioning crop yield scenarios. The proposed approach has been employed for envisioning crop yield scenarios of maize, potato, rice, tomato, okra, cabbage, mustard at All-India level considering the impact of Bt technology.

Trend Impact Analysis (TIA) offers a valuable extension to classical time series forecasting methods by explicitly integrating the effects of unexpected interventions or exogenous forces that can significantly alter the trajectory of a variable of interest. Unlike traditional time series models that assume continuity of past trends, intervention-based models like the ARIMA Intervention Model and its nonlinear counterpart, NARX, help capture the real impact of sudden or gradual external events, resulting in more realistic and actionable forecasts.

In this study, by combining structured expert judgment (Delphi), GOS tree analysis, and time series intervention modeling, a robust TIA framework was developed for envisioning future crop yield scenarios. This integrated approach was successfully applied to major crops — including maize, potato, rice, tomato, okra, cabbage, and mustard at the All-India level to assess the impact of Bt technology adoption. The results demonstrated that the trend intervention-based models consistently outperformed classical time series models in capturing shifts caused by technological interventions, providing more accurate and policy-relevant insights for agricultural planning and technology foresight.

These findings highlight that TIA, especially when combined with hybrid modeling techniques and expert inputs, can serve as an effective decision-support tool for researchers, policymakers, and planners to understand possible future scenarios and design resilient strategies in the face of climatic and technological disruptions.

R codes:

```
rm(list=ls())
library(forecast)
library(tseries)
library(TSA)
library(ggplot2)
library(tidyverse)
library(lmtest)
g=read.table(file="Gudumalkapur.txt",header=T)
head(g)
dim(g)
Box.test(g$Arrivals)
bdsTest(g$Arrivals, m = 3, eps = NULL, title = NULL, description = NULL)
a1=g$Arrivals[1:2856]
a2=g$Arrivals[2857:2887]
i1=g$Int[1:2856]
i2=g$Int[2857:2887]
Box.test(a1)
acf(a1)
pacf(a1)
############## ARIMA Fitting #########
m1=auto.arima(a1)
coeftest(m1)
accuracy(m1)
Box.test(m1$residuals)
fitted1=m1$fitted
write.csv(as.data.frame(fitted1), file="ARIMA_Fitted.csv")
f1=forecast(m1, h=30)
f11=data.frame(f1)
f12=f11$Point.Forecast
mape1=abs(a2-f12)/abs(a2)
mape11=mean(mape1)*100
mape11
write.csv(as.data.frame(f12), file="ARIMA_Forecasted.csv")
#################### ANN ##########
m2=nnetar(a1,2, P=1, 5, repeats=25, xreg=NULL, lambda=NULL, model=NULL,
subset=NULL, scale.inputs=TRUE,  maxit=150)
```

```
m2
accuracy(m2)
fitted2=m2$fitted
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
Box.test(m2$residuals)
f2=forecast(m2, h=30)
f21=data.frame(f2)
f22=f21$Point.Forecast
mape2=abs(a2-f22)/abs(a2)
mape21=mean(mape2)*100
mape21
write.csv(as.data.frame(f22), file="ANN_Forecasted.csv")
############# ARIMA Int ###########
m3=auto.arima(a1, xreg=i1)
coeftest(m3)
accuracy(m3)
fitted3=m3$fitted
write.csv(as.data.frame(fitted3), file="ARIMA_Int_Fitted.csv")
Box.test(m3$residuals)
f3=forecast(m3, h=30, xreg=i2)
f31=data.frame(f3)
f32=f31$Point.Forecast
mape3=abs(a2-f32)/abs(a2)
mape31=mean(mape3)*100
mape31
write.csv(as.data.frame(f32), file="ARIMA_Int_Forecasted.csv")
############ ANN_Int##########
m4=nnetar(a1,2, P=1, 5, repeats=25, xreg=i1, maxit=150)
m4
accuracy(m4)
fitted4=m4$fitted
write.csv(as.data.frame(fitted4), file="ANN_Int_Fitted.csv")
Box.test(m4$residuals)
f4=forecast(m4, h=30, xreg=i2)
f41=data.frame(f4)
f42=f41$Point.Forecast
mape4=abs(a2-f42)/abs(a2)
mape41=mean(mape4)*100
mape41
write.csv(as.data.frame(f42), file="ANN_Int_Forecasted.csv")
##########Significance Comparison ##########
########## For testing set ######
dm.test(m1$residuals, m2$residuals)
dm.test(m1$residuals, m3$residuals)
dm.test(m1$residuals, m4$residuals)
dm.test(m2$residuals, m3$residuals)
dm.test(m2$residuals, m4$residuals)
dm.test(m3$residuals, m4$residuals)
```

######## You have to do it for testing set also #####

Suggested Readings

- Agami, N., Omran, A., Saleh, M. & El-Shishiny, H. (2008). A enhanced approach for trend impact analysis. Technological Forecasting & Social Change, 75, 1439-1450.
- Asan, S. S. and Asan U. (2007). Qualitative cross-impact analysis with time consideration. Technological Forecasting & Social Change, 74, 627-644.
- Bañuls, V.A. and Turoff, M. (2011). Scenario construction via Delphi and cross-impact analysis. Technological Forecasting and Social Change. 78(9), 1579-1602.
- Gordon, T. J. (2002). Trend Impact Analysis. AC/UNV Millennium project, Futures Research Methodology-V 2.1(CD-ROM)
- Ray, M., Rai, A., Singh, K. N., V., Ramasubramanian and Kumar, A. (2017). Technology forecasting using time series intervention based trend impact analysis for wheat yield scenario in India. Technological Forecasting and Social Change, 118, 128-133.
- Kane, J. (1972). A primer for a new cross-impact language- KSIM, Technological Forecasting and Social Change, 4, 129-142.
- Godet, M. (1976). Scenarios of air transport development to 1990 by SMIC 74 – A new cross impact method, Technological forecasting and social change, 9, 279-288.
- Ramasubramanian, V., Ananthan, P.S., Krishnan, M. and Vinay, A. (2017). Technology forecasting in fisheries sector: Cross impact analysis and substitution modeling, Journal of the Indian Society of Agricultural Statistics, 71(3), 231–239.
- Ray, M., Ramasubramanian, V., Singh, K.N., Rathod, S. and Shekhawat, R. S. (2022).Technology Forecasting for Envisioning Bt Technology Scenario in Indian Agriculture. Agricultural Research, https://doi.org/10.1007/s40003-022-00612-z
- Chitikela, G., Rathod, S. & Vijayakumar, S. Change point-driven interrupted time series and machine learning models for forecasting indian food grain production. Discov Food 5, 68 (2025). https://doi.org/10.1007/s44187-025-00350-5
- Rathod, S., Chitikela, G., Bandumula, N., Ondrasek, G., Ravichandran, S., & Sundaram, R. M. (2022). Modeling and Forecasting of Rice Prices in India during the COVID-19 Lockdown Using Machine Learning Approaches. *Agronomy*, *12*(9), 2133. https://doi.org/10.3390/agronomy12092133
- Chitikela, G., Admala, M., Ramalingareddy, V. K., Bandumula, N., Ondrasek, G., Sundaram, R. M., & Rathod, S. (2021). Artificial-Intelligence-Based Time-Series Intervention Models to Assess the Impact of the COVID-19 Pandemic on Tomato Supply and Prices in Hyderabad, India. *Agronomy*, *11*(9), 1878. https://doi.org/10.3390/agronomy11091878

Time Series Analysis: Cointegration Analysis

Kanchan Sinha

ICAR-Indian Agricultural Statistics Research Institute

Library Avenue, Pusa, New Delhi-110012

Email: kanchan.sinha@icar.gov.in

1. Introduction

In econometrics, cointegration analysis is a powerful tool used to estimate and test long-run equilibrium relationships among non-stationary time series variables such as income and consumption, interest rates of different maturities, or stock prices. Its primary significance lies in its ability to address challenges posed by the use of non-stationary data, which is common in macroeconomic and financial time series. When two or more non-stationary series are cointegrated, they share a common stochastic trend, indicating the existence of a meaningful long-term relationship, despite short-term deviations.

A time series is considered stationary when its mean and variance remain constant over time, and the covariance between values depends only on the lag between time points, not on the actual time at which the covariance is computed. In contrast, non-stationary series exhibit time-varying means or variances, complicating statistical inference and model validity. Therefore, econometric models dealing with such series must be carefully specified to yield valid economic interpretations.

To address non-stationarity, a common approach is to apply differencing, which transforms a trending series into a stationary one. The number of differencing steps required to achieve stationarity defines the order of integration. A series that becomes stationary after first differencing is termed integrated of order one, denoted as I(1), while a stationary series without differencing is I(0).

Cointegration analysis allows researchers to identify and model long-run relationships without discarding essential information, unlike traditional approaches such as regression on first-differenced data, which may lead to loss of long-term dynamics. Moreover, earlier methods like price correlation coefficients could be misleading when applied to non-stationary data due to the presence of unit roots, and may yield spurious relationships.

In the context of market integration, cointegration plays a crucial role in understanding price transmission mechanisms across regional or international markets. For instance, strong cointegration between domestic and world prices suggests efficient market integration, low trade barriers, and synchronized price movements. Conversely, weak cointegration may imply fragmented markets and significant price disparities.

Additionally, cointegration inherently implies the existence of Granger causality among variables indicating that prices in one market may help predict price changes in another. This property is particularly useful for policymakers and market analysts to examine price leadership, information flow, and the direction of causality between integrated markets.

Thus, cointegration analysis not only enhances the statistical robustness of time series models involving non-stationary variables but also provides meaningful economic insights into the long-run co-movements and causal interrelationships between key economic indicators.

2. Model Specification

2.1 Vector Autoregressive(VAR) process

A VAR is a simple extension of the *AR(k)* framework and is given by:

$$Y_t = \delta + A_1 Y_{t-1} + A_2 Y_{t-2} + \cdots + A_k Y_{t-k} + u_t \qquad \text{(i)}$$

where, $u_t \sim IN(0, \Sigma)$

where, $Y_t = (Y_{1t}, Y_{2t}, \ldots, Y_{nt})'$ is $(\mathrm{n} \times 1)$ random vector of endogenous variables, each of the A_i is an $(n \times n)$ matrix of parameters, δ is a fixed $(n \times 1)$ vector of intercept terms. Finally, $u_t = (u_{1t}, u_{2t}, \ldots, u_{nt})'$ is a n-dimensional white noise or innovation process, i.e., $E(u_t) = 0$, $E(u_t, u_t{}') = \Sigma$ and $E(u_t, u_s{}') = 0$ for $s \neq t$. The covariance matrix Σ is assumed to be non-singular.

2.2 Cointegration process

Cointegration analysis is used to examine whether long-run equilibrium relationships exist between two or more series. The long-run relationship is given as:

$$P_t^1 = \alpha_0 + \alpha_1 P_t^2 + \varepsilon_t \qquad \text{(ii)}$$

Let P_t^1 and P_t^2 denote the prices of a given commodity in two distinct markets. If the error term ε_t is stationary, it implies that the market prices are cointegrated. Cointegration analysis captures the long-run equilibrium relationship between price series, even though short-term deviations may occur. Johansen's multivariate cointegration technique is employed to assess the presence of cointegration between the two price series. Prior to applying the cointegration test, it is essential to verify the stationarity of the series. This is done using the Augmented Dickey-Fuller (ADF) test, which examines stationarity by regressing the original price series with an intercept, trend, the first differences, and lagged differences. Variables that are integrated to the same order are eligible for cointegration testing. The unit root test helps determine the order of integration—for example, a variable integrated of order one is denoted as I(1), while integration of order ppp is denoted as I(p). The ADF unit root test can be formulated as follows:

$$\Delta y_t = \beta_1 + \beta_2 t + \delta y_{t-1} + \sum_{i=1}^{m} \alpha_i \Delta y_{t-i} + \varepsilon_t \qquad \text{(iii)}$$

where Δy_t is a vector to be tested for cointegration, t is time or trend variable. Δy_t is the first difference ie., ($\Delta y_t = y_t - y_{t-1}$), ε_t is a white noise term. The null hypothesis that, H_0: $\delta = 0$; signifying unit root, states that the time series is non-stationary while the alternative hypothesis, H_1: $\delta < 0$, signifies that the time series is stationary, thereby rejected the null hypothesis. Since ADF tests tell us whether a time series is integrated or not, therefore the test is known as a "Test for integration".

2.3 Johansen's Cointegration Tests

A cointegrated system can be written as:

$$\Delta y_t = \sum_{i=1}^{k} \Gamma_i \, \Delta y_{t-i} + \alpha\beta' y_{t-k} + \varepsilon_t \qquad \text{(iv)}$$

where y_t is the price series, Δy_t is the first difference i.e., ($\Delta y_t = y_t - y_{t-1}$), and the matrix $\alpha\beta'$ is $n\ x\ n$ with rank $(0 \le r \le n)$, which is the rank of linear independent cointegration relations in the vector space of matrix. The Johansen's method of cointegrated system is a restricted maximum likelihood method with rank restriction on matrix $\Pi = \alpha\beta'$. The rank of Π can be obtained by using λ_{trace} or λ_{max} test statistics. The test statistics can be written as:

$$\lambda_{trace} = -T \sum_{i=r+1}^{n} \ln\,(1 - \widehat{\lambda_\iota}) \;\; \forall\, r = 0, 1, \dots, n-1 \qquad \text{(v)}$$

The estimated eigenvalues $\widehat{\lambda_\iota}$'s represent the magnitude of correlation between the differenced terms and the error-correction components. To determine the number of cointegrating relationships, the Johansen cointegration test is applied by evaluating the following hypotheses: the null hypothesis H_0: $rank\ of\ \Pi = r$ and under alternative hypothesis, H_1: $rank\ of\ \Pi > r$, where rrr denotes the number of cointegrating vectors. This test is conducted under the assumption that the cointegrating equation contains only an intercept (i.e., no deterministic trend), whereas the original series may exhibit a trend due to non-constant mean and variance over time, indicating non-stationarity.

Granger Causality Test

Once cointegration between the time series is established, the Granger causality test is applied to investigate the direction of causality between the variables. If two markets are cointegrated, it implies a long-run equilibrium relationship, and typically, the price in one market is found to Granger-cause the price in the other market and/or vice versa. The Granger causality test thus provides further insight into the dynamics of price transmission, indicating whether and in which direction the causal influence flows between the series.

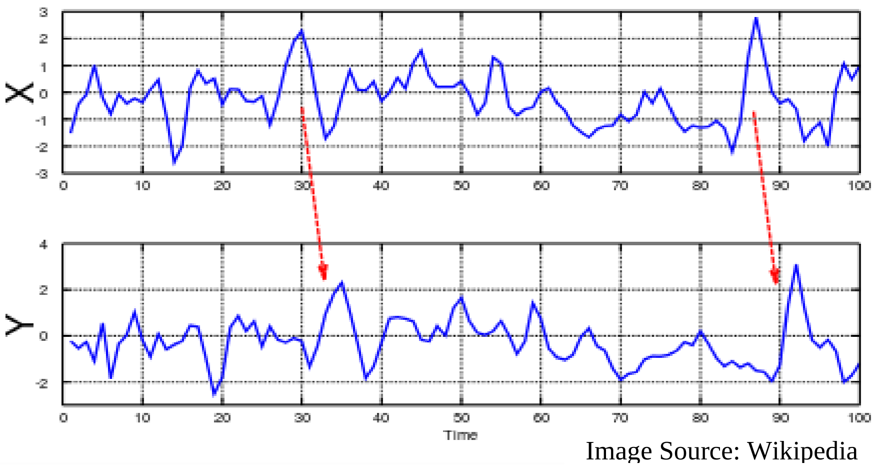


Image Source: Wikipedia

Figure: Time series *X* Granger-causes time series *Y*; the patterns in *X* are approximately repeated in *Y* after some time lag (two examples are indicated with arrows). Therefore, past values of *X* can be used for the prediction of future values of *Y*.

A VAR (2) model is applied in order to assess the causality of the price series.

$$\begin{pmatrix} y_t \\ x_t \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} + \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} y_{t-1} \\ x_{t-1} \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{bmatrix} \begin{bmatrix} y_{t-2} \\ x_{t-2} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{bmatrix} \qquad \text{(vi)}$$

The matrix relation can be written in individual form as:

$$y_t = a + c_{11}y_{t-1} + c_{12}x_{t-1} + d_{11}y_{t-2} + d_{12}x_{t-2} + \varepsilon_{1t} \qquad \text{(vii)}$$

$$x_t = b + c_{21}y_{t-1} + c_{22}x_{t-1} + d_{21}y_{t-2} + d_{22}x_{t-2} + \varepsilon_{2t} \qquad \text{(viii)}$$

The restrictions imposed to test the causality can be described as:

lags of *y* do not explain the value of *x* so, $c_{21} = 0$ and $d_{21} = 0$

lags of x do not explain the value of y so, $c_{12} = 0$ and $d_{12} = 0$

Hence, the null hypothesis for Granger causality test is defined as:

$H_0: c_{12} = d_{12} = 0$ (x_t does not Granger cause y_t)

$H_0: c_{21} = d_{21} = 0$ (y_t does not Granger cause x_t)

2.5 Vector Error Correction Model (VECM)

If the time series are found to be cointegrated, a Vector Error Correction Model (VECM) is estimated. The VECM can be viewed as an extension of the Vector Autoregressive (VAR) model, augmented by an error correction term that captures deviations from the long-run equilibrium. The VECM possesses two essential characteristics:

First, it is dynamic in nature, incorporating both lagged values of the dependent and explanatory variables. This allows the model to capture short-run adjustments that arise from past disequilibria and current changes in the explanatory variables.

Second, the VECM framework explicitly reveals the long-run cointegrating relationship among the variables through the error correction term. This term quantifies the speed at which the system returns to equilibrium after a short-term shock.

Equation (ix) presents the structure of a VECM involving three variables. The model specification includes a constant term, the error correction component, lagged endogenous variables, and a stochastic error term, thereby providing a comprehensive depiction of both short- and long-run dynamics.

$$\begin{bmatrix} \Delta P_t^B \\ \Delta P_t^C \\ \Delta P_t^H \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} + \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} ECT_{-1} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} \Delta P_{t-1}^B \\ \Delta P_{t-1}^C \\ \Delta P_{t-1}^H \end{bmatrix} + \begin{bmatrix} \varepsilon_t^{P^B} \\ \varepsilon_t^{P^C} \\ \varepsilon_t^{P^H} \end{bmatrix} \tag{ix}$$

In equation (ix), P_t^B, P_t^C and P_t^H represents time series datasets from three different markets. The Vector Error Correction Model (VECM) is a powerful framework for analyzing both short-run dynamics and long-run equilibrium relationships among cointegrated time series variables. This representation is particularly valuable because it enables the estimation of how quickly variables adjust toward their long-term equilibrium path following a deviation.

A central feature of the VECM is the Error Correction Term (ECT), whose coefficient (denoted typically as αi\alpha_iαi) captures the speed of adjustment back to the long-run equilibrium after a shock. For the model to be valid and economically meaningful, the ECT coefficient must be negative and statistically significant. A negative and significant ECT implies that when the system deviates from equilibrium, it self-corrects in subsequent periods, restoring the balance over time. On the other hand, the coefficients of lagged explanatory variables in the VECM describe the short-run adjustments, revealing how immediate past changes in one variable affect current values of others

3. Conclusion

In a developing economy like India, understanding market integration is essential for formulating effective agricultural marketing policies, enhancing marketing efficiency, and guiding farmers in production planning and crop diversification towards high-value commodities. The concepts of cointegration and VECM modeling provide valuable tools for quantifying the degree of market integration by identifying long-term price relationships and information transmission between markets.

Such studies help researchers and policymakers assess whether markets move together in the long run and how quickly they respond to temporary shocks. They offer crucial insights into market efficiency, price stabilization, and the effectiveness of policy interventions, ultimately benefiting all stakeholders in the agricultural supply chain—from producers and traders to consumers and regulators.

4. Suggested Readings

Engle, R.F. and Granger, C.W. J. (1987) Cointegration and error correction: Representation, estimation and testing, *Econometrica*, **50**, 987-1007

Sinha, K, Paul, R.K. and Bhar, L. M. (2016) Price Transmission and Causality in major onion markets of India. *Journal of the Society for Application of Statistics in Agriculture and Allied Sciences (SASAA)*, **1(2)**, 35-40

Paul, R. K. and Sinha, K. (2015) Spatial market integration among major coffee markets in India, *Journal of the Indian Society of Agricultural Statistics*, **69 (3)**, 281-287

Sahu, P.K., Dey, S., Sinha, K. Singh, H. and Narsimaiaha, L. (2019) Cointegration and Price Discovery Mechanism of Major Spices in India, *American Journal of Applied Mathematics and Statistics*, **7(1)**, 18-24

Long Memory Time-Series Models

Ranjit Kumar Paul
ICAR-Indian Agricultural Statistics Research Institute
Library Avenue, Pusa, New Delhi-110012
Email: ranjitstat@gmail.com

Introduction

While many economic time series are inherently non-stationary and commonly require differencing to achieve stationarity, the conventional Box-Jenkins ARIMA methodology—which assumes that differencing of integer order is sufficient—may not always yield the best model fit, especially in the absence of seasonal components. The traditional assumption is that once the appropriate number of integer differences is applied, the resulting series will exhibit rapidly decaying autocorrelations, allowing it to be adequately captured by a stationary and invertible ARMA model. This approach has been widely applied in agriculture (Paul and Das, 2010; 2013; Paul et al., 2013a; 2013b).

However, empirical evidence suggests that some time series do not contain a further unit root yet continue to exhibit persistent dependence over time—a phenomenon known as long memory. Such series may not be well-described by ARMA or ARIMA models. In these cases, a more flexible modeling approach is required—specifically, models that allow for fractional differencing. The Autoregressive Fractionally Integrated Moving Average (ARFIMA) model addresses this issue by incorporating a non-integer differencing parameter d. This parameter quantifies the degree of long-range dependence in the data, where values of d different from zero imply the presence of long memory. The magnitude of d reflects the strength of this memory, and its non-integer nature has led to its association with fractal structures in time series.

Notably, ARFIMA models offer an effective alternative to conventional ARIMA models for series exhibiting such persistent dependencies. Estimating the long memory parameter d using modern techniques—such as the wavelet-based approach—has gained traction in recent econometric research. Key contributions in the field include Robinson (1995) and Baillie et al. (1996), who surveyed long memory modeling in econometrics, while Beran (1994) provided insights across other disciplines. Despite the growing literature, long memory analysis in the context of agricultural commodity markets remains underdeveloped. One of the early studies by Helms et al. (1984) applied classical rescaled range (R/S) analysis to a limited data set of a single commodity. Given the complex dynamics and persistence often observed in agricultural price series, there is a compelling need for further investigation into the long memory behavior of agricultural markets, using ARFIMA models and wavelet-based techniques.

Long Memory Process

Long memory in time-series data refers to the persistence of autocorrelations across long time lags (Robinson, 1995). In the context of time-series modeling, long memory implies that observations are not independent; instead, each observation is influenced by events that occurred in the distant past (Jin and Frechette, 2004). This is in contrast to short memory processes, where the impact of past events diminishes rapidly. The autocorrelation function (acf) of a time-series y_t is defined as

$$\rho_k = cov(y_t, y_{t-1})/var(y_t) \tag{1}$$

for integer lag k. A covariance stationary time-series process is expected to have autocorrelations such that $\lim_{k\to\infty} \rho_k = 0$. Most of the well-known class of stationary and invertible time-series processes have autocorrelations that decay at exponential rate, so that $\rho_k \approx |m|^k$, where $|m|<1$ and this property is true, for example, for the well-known stationary and invertible ARMA(p,q) process. For long memory processes, the autocorrelations decay at an hyperbolic rate which is consistent with $\rho_k \approx Ck^{2d-1}$, as k increases without limit, where C is a constant and d is the long memory parameter.

Testing of Long Memory

The Hurst exponent (H), derived from the rescaled range (R/S) analysis, is a widely used statistical measure for detecting the presence of long memory or long-range dependence in a time-series. Originally introduced by H.E. Hurst in hydrology, the method was later extended and applied to economic and financial time-series by Booth et al. (1982) and Helms et al. (1984). For a given time-series, the Hurst exponent quantifies the degree of long-term, non-periodic dependence, reflecting how long the memory or persistence of past values influences future observations. Specifically, the Hurst exponent indicates the average duration over which a time series remains correlated. The R/S analysis first estimates the range R for a given n:

$$R(n) = \max_{1\le j\le n} \sum_{j=1}^{n} \left(Y_j - \bar{Y}\right) - \min_{1\le j\le n} \sum_{j=1}^{n} \left(Y_j - \bar{Y}\right) \tag{2}$$

where $R(n)$ is the range of accumulated deviation of $Y(t)$ over the period of n and $\bar{Y}$ is the overall mean of the time-series. Let $S(n)$ be the standard deviation of Y_t over the period of n. For a given n, there exists a statistic

$$Q(n) = R(n)/S(n) \tag{3}$$

Here, n is the time scale to split total observations T into int[T/n] segments where int[.] denotes the integer part of [.]. There will be int[T/n] estimates of $R(n)/S(n)$ for a given n. The final $R(n)/S(n)$ is the average of int[T/n]'s $R(n)/S(n)$**.** As n increases**,** the following holds:

$$R(n)/S(n) = \alpha n^{H}, \ \alpha \ \text{ is a constant.}$$

or

$$\log(R(n)/S(n)) = \log\alpha + H\log n \tag{4}$$

Thus, H is a parameter that relates mean R/S values for subsamples of equal length of the series to the number of observations within each equal length subsample. H is always greater than 0. When $0.5<H<1$, the long memory structure exists. If $H \geq 1$, the process has infinite variance and is nonstationary. If $0<H<0.5$, anti-persistence structure exists. If $H=0.5$, the process is white noise. The relationship between Hurst exponent and long memory parameter is: $H=1-d$,

ARFIMA Model

Fractional integration serves as the fundamental conceptual framework for characterizing long memory in time-series data, particularly in financial and economic applications. Unlike traditional integer-order integration, which assumes time-series are integrated of order zero [I(0)] or one [I(1)], fractional integration provides a more flexible approach by allowing the order of integration, denoted by d to take on non-integer (fractional) values. This generalization is especially useful for modeling series that exhibit long-range dependence, where the autocorrelations decay at a slower, hyperbolic rate rather than the exponential decay observed in short memory processes. The most commonly used model incorporating fractional differencing is the Autoregressive Fractionally Integrated Moving Average (ARFIMA) model, denoted by ARFIMA(p,d,q),

p: order of the autoregressive (AR) component,

d: fractional differencing parameter (order of integration),

q:order of the moving average (MA) component.

$$(1-L)^{\mathrm{d}}\varphi(L)y_t = \theta(L)u_t \tag{5}$$

where u_t is an independently and identically distributed (i.i.d.) random variable with zero mean and constant variance, L denotes the lag operator; and $\varphi(L)$ and $\theta(L)$ denote finite polynomials in the lag operator with roots outside the unit circle. For $d = 0$, the process is stationary, and the effect of a shock $u(t)$ on $y(t + j)$ decays geometrically as j increases. For d = 1, the process is said to have a unit root, and the effect of a shock $u(t)$ on $y(t + j)$ persists into the infinite future. In contrast, fractional integration defines the function $(1 - L)^{-d}$ for noninteger values of the fractional differencing parameter d.

For $-0.5 < d < 0.5$ the process $y(t)$ is stationary and invertible. For such processes, the effect of a shock $u(t)$ on $y(t+j)$ decays as j increases, but the rate of decay is much slower than for a process integrated of order zero.

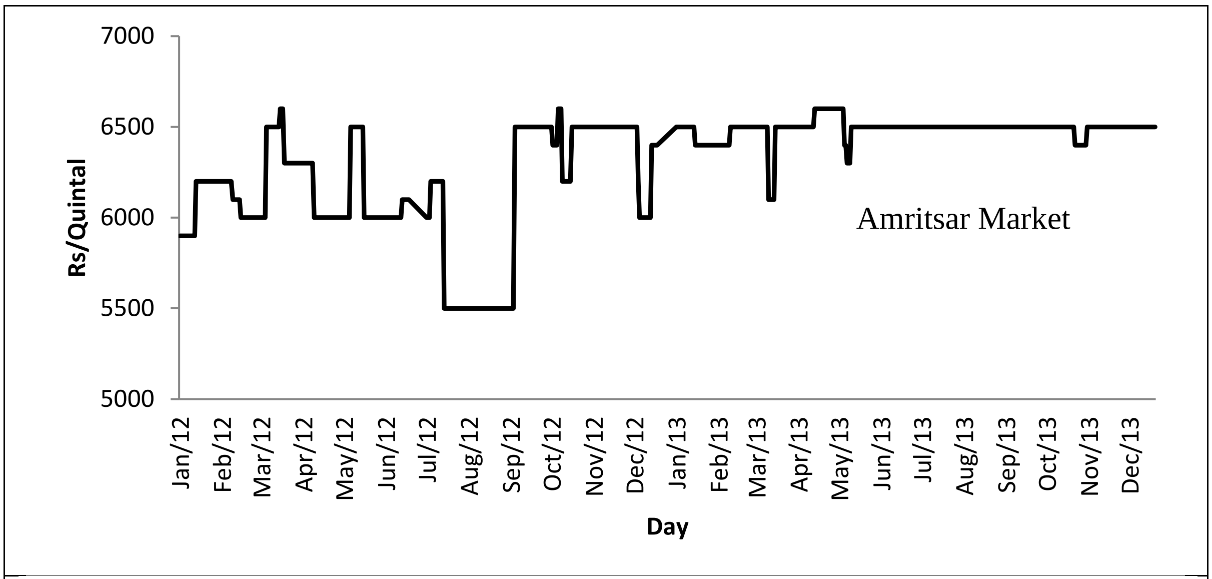
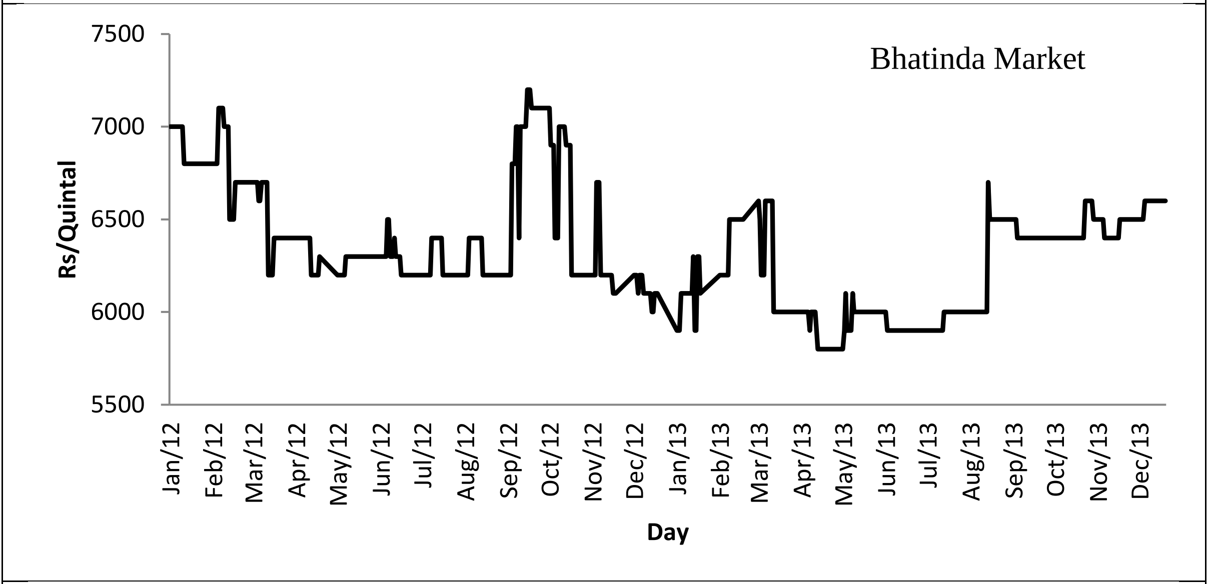
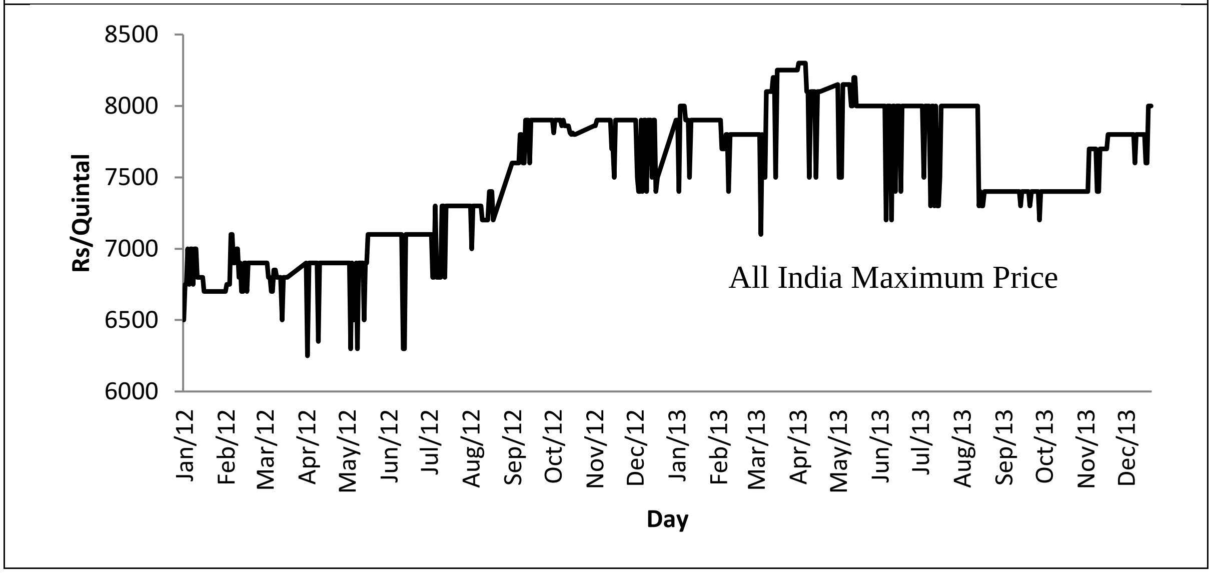
In stationary time series, the autocorrelation function (ACF) decays geometrically, while in fractionally integrated processes, it decays hyperbolically, indicating long memory. The sign of autocorrelations aligns with the sign of the differencing parameter d. Thus, ARFIMA(p, d, q) models effectively capture long memory more efficiently than high-order ARMA models.Correct specification of p and q is crucial. As noted by Robinson (2003), under- or over-specifying AR or MA orders can lead to inconsistent estimation of both short-term coefficients and the long memory parameter d, causing model misidentification.

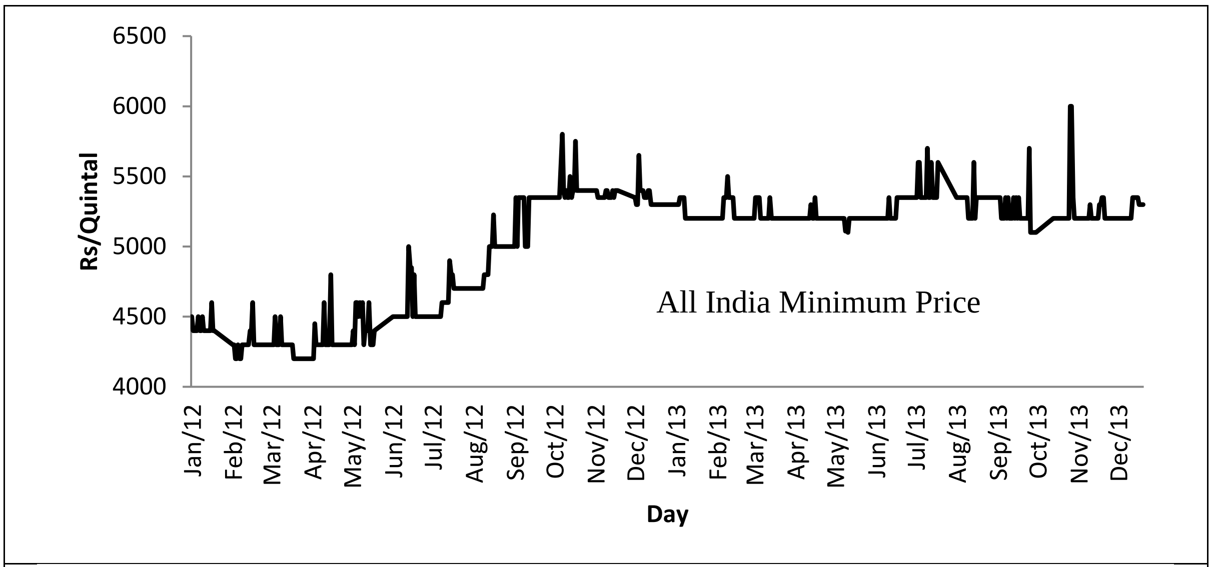
Estimation of long memory parameter

For estimating the long memory parameter, GPH estimator proposed by Geweke and Porter-Hudak (1983) is used in the present investigaton. Robinson (1995), Hurvich *et al*. (1998) and Tanaka (1999) have analyzed the GPH estimate in detail. Under the assumption of normality for y_t, it has been proved that the estimate is consistent and asymptotically normal.

Illustration (Paul, 2014)

Daily wholesale prices of pigeon pea (Arhar) from Amritsar, Bhatinda, and all-India maximum, minimum, and modal prices from Jan 1, 2012 to Dec 31, 2013 were sourced from the Ministry of Consumer Affairs, Govt. of India. Data from Jan 2012 to Oct 2013 were used for model development, and the rest for validation. Figure 1 shows that the series appear stationary. To confirm, ADF (Said & Dickey, 1984) and PP (Philips & Perron, 1988) unit root tests were conducted. As per Table 1, all series are stationary. If trends are present, a test with trend is applied; otherwise, a test with mean only is used.


7000
6500
6000
5500
5000
Rs/Quintal
Amritsar Market
Jan/12
Feb/12
Mar/12
Apr/12
May/12
Jun/12
Jul/12
Aug/12
Sep/12
Oct/12
Nov/12
Dec/12
Jan/13
Feb/13
Mar/13
Apr/13
May/13
Jun/13
Jul/13
Aug/13
Sep/13
Oct/13
Nov/13
Dec/13
Day

7500
7000
6500
6000
5500
Rs/Quintal
Bhatinda Market
Jan/12
Feb/12
Mar/12
Apr/12
May/12
Jun/12
Jul/12
Aug/12
Sep/12
Oct/12
Nov/12
Dec/12
Jan/13
Feb/13
Mar/13
Apr/13
May/13
Jun/13
Jul/13
Aug/13
Sep/13
Oct/13
Nov/13
Dec/13
Day

8500
8000
7500
7000
6500
6000
Rs/Quintal
All India Maximum Price
Jan/12
Feb/12
Mar/12
Apr/12
May/12
Jun/12
Jul/12
Aug/12
Sep/12
Oct/12
Nov/12
Dec/12
Jan/13
Feb/13
Mar/13
Apr/13
May/13
Jun/13
Jul/13
Aug/13
Sep/13
Oct/13
Nov/13
Dec/13
Day



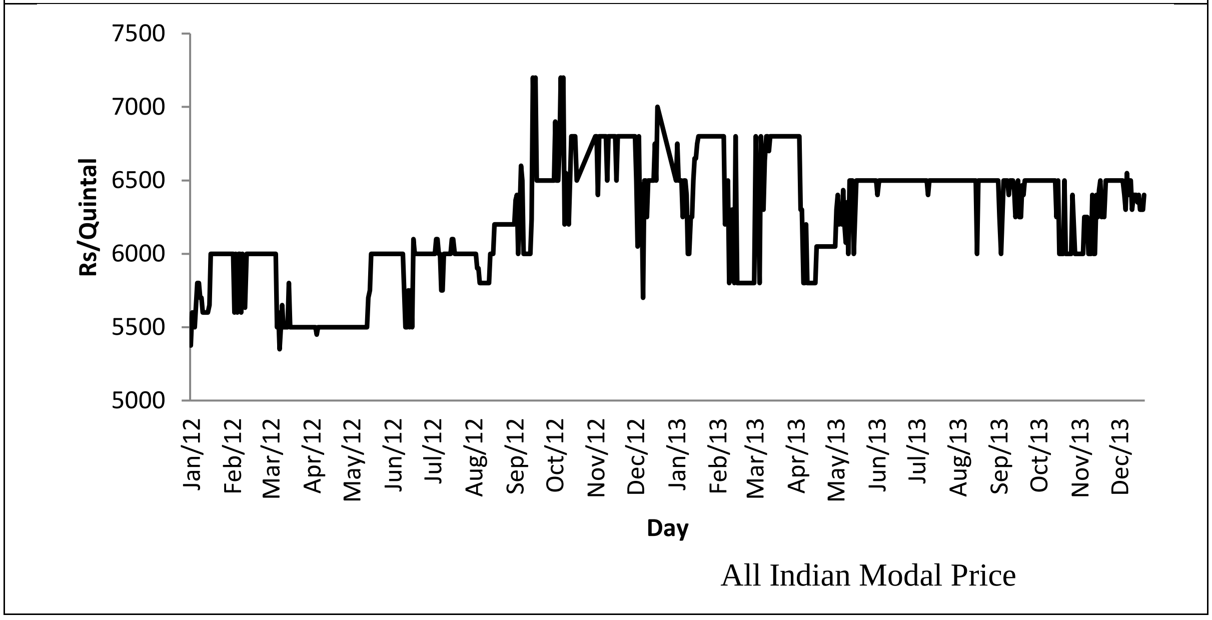


Fig. 1: Time series plot of wholesale prices of Pigeon Pea in different markets

Structure of Autocorrelations

For a linear time series model, such as the autoregressive integrated moving average (ARIMA(p,d,q)) process, the patterns of autocorrelations (ACF) and partial autocorrelations (PACF) help determine the plausible model structure. This information is also crucial when modeling nonlinear dynamics. The presence of long-lasting autocorrelations suggests that the underlying process may be nonlinear with time-varying variances. A key property of long memory processes is that dependence between distant observations remains significant. For the daily wholesale price series, autocorrelations were estimated up to 100 lags (j = 1,...,100). The

ACF plots, shown in Figure 2, indicate that these autocorrelations decay hyperbolically rather than exponentially, and do not display any clear seasonal or periodic patterns. The magnitude of the autocorrelations does not diminish rapidly as the lag increases, confirming the absence of short-term cycles and highlighting the long memory nature of the data.

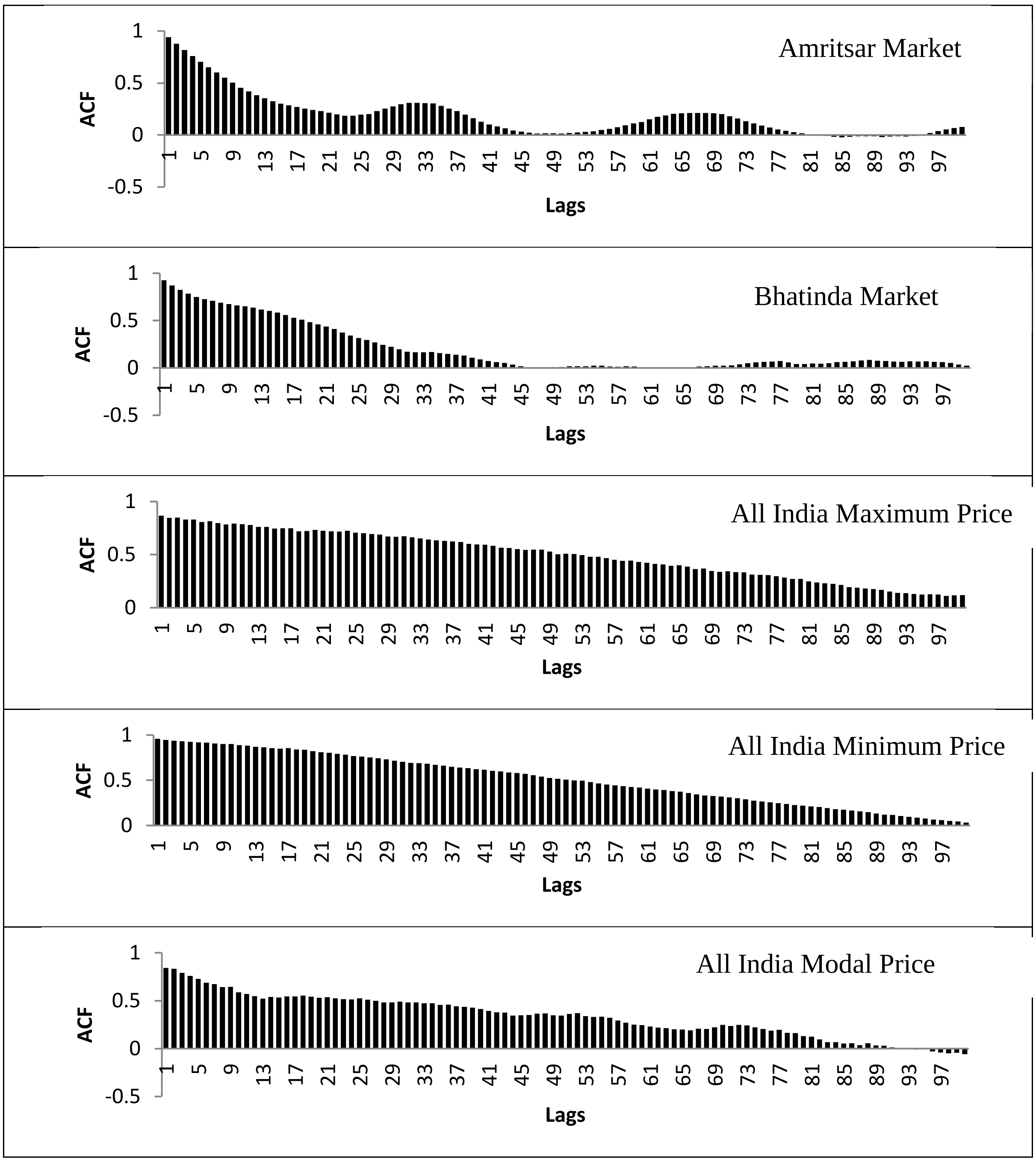


Fig. 2: Correlogram of time series data of wholesale prices of Pigeon Pea in different markets

Testing Stationarity

ADF Test

The ADF test tests the null hypothesis that a time series y_t is I(1) against the alternative that it is I(0), assuming that the dynamics in the data have an ARMA structure. The ADF test is based on estimating the test regression

$$\Delta y_t = \boldsymbol{\beta}'\mathbf{D_t} + \pi y_{t-1} + \sum_{j=1}^{p} \psi_j \Delta y_{t-j} + \varepsilon_t$$

In this context, $\mathbf{D_t}$ represents a vector of deterministic components, such as constants and linear trends. The model includes ppp lagged difference terms to capture and approximate the autoregressive moving average (ARMA) structure of the error process. The optimal lag length p is chosen to ensure that the residual term ut is free from serial correlation. Additionally, the error term is assumed to be homoskedastic.

Under the null hypothesis, the series is integrated of order zero, denoted as I(0), which corresponds to the condition π=0. The Augmented Dickey-Fuller (ADF) test statistic used to test this null hypothesis is the conventional t-statistic applied to the coefficient π. The ADF test was conducted on the dataset under consideration, and the results are presented in Table 1.

Phillips-Perron (PP) Unit Root Tests

The Phillips-Perron (PP) unit root test differs from the ADF test primarily in the treatment of serial correlation and heteroskedasticity in the error terms. While the ADF test incorporates a parametric autoregressive framework to model the ARMA structure in the error process, the PP test allows for a more flexible approach by not requiring such parametric adjustments within the test regression.

The test regression for the PP approach is specified as follows:

$$\Delta y_t = \boldsymbol{\beta}'\mathbf{D_t} + \pi y_{t-1} + u_t$$

where the error term ut is assumed to be I(0) and may exhibit heteroskedasticity. The PP test accounts for both serial correlation and heteroskedasticity in the residuals ut by applying non-parametric corrections directly to the test statistics. Despite these corrections, under the null hypothesis π=0, the asymptotic distribution of the PP test statistic remains identical to that of the ADF t-statistic.

One of the key advantages of the PP test is its robustness to general forms of heteroskedasticity in the error term. Moreover, unlike the ADF test, the PP test does not require the user to pre-specify the lag length for the regression, making it more flexible in certain empirical applications. The PP test was also applied to the current dataset, and the results are likewise reported in Table 1.

Table 1: Stationarity testing

Market		ADF Test Statistic				PP Test Statistic			
		Single Mean	With Trend	Probability		Single Mean	With Trend	Probability	
				Single Mean	With Trend			Single Mean	With Trend
Amritsar		5.46	7.08	0.0238	0.0283	-3.29	-3.76	0.0165	0.0199
Bhatinda		6.85	9.43	0.0010	0.0010	-3.70	-4.27	0.0047	0.0039
All India	Maximum	13.11	42.71	<.0001	<.0001	-5.12	-9.23	<.0001	<.0001
	Minimum	8.32	11.55	0.0402	0.0010	-3.53	-4.81	0.0195	0.0006
	Modal	15.43	27.00	<.0001	<.0001	-5.55	-7.35	<.0001	<.0001

The most widely used method to estimate the fractional integration parameter d is the ARFIMA time series approach (Robinson, 2003). Various ARFIMA model specifications were estimated, and the best model was selected based on the minimum AIC value. The estimated parameters and corresponding t-statistics for the selected ARFIMA models are reported in Table 2. The results indicate evidence of long memory in five price series, with $0<d<0.5$. Positive and significant d values suggest persistence—characterized by positive autocorrelations and low-frequency variance. When d is significantly positive, it may imply that the series has infinite conditional variance. The estimated d values range from 0.052 to 0.489, with the All India Maximum Price series exhibiting the strongest long memory. These findings confirm that autocorrelations decay hyperbolically with increasing lag length.

Table 2: Parameter estimates of ARFIMA Model

Market	Parameters	Estimate	Probability
Amritsar	d	0.077	0.001
	AR1	0.915	< 0.001
Bhatinda	d	0.052	0.040
	AR1	1.6154	< 0.001
	AR2	-0.623	< 0.001
	MA1	0.821	< 0.001
Maximum Price	d	0.489	< 0.001
	AR1	-0.223	< 0.001
	AR2	-0.128	0.0168
Minimum Price	d	0.093	< 0.001
	AR1	1.1467	< 0.001
	AR2	-0.149	< 0.001
	MA1	0.784	< 0.001

Modal Price	d	0.477	< 0.001
	AR1	-0.157	0.008
	AR2	0.183	< 0.001

Validation

One-step-ahead forecasts of wholesale prices and their corresponding standard errors were generated using the naïve approach for the period from November 1, 2013 to December 31, 2013 (covering 40 data points, excluding market holidays) based on the fitted ARFIMA model. A notable feature of the fitted model is that all observed values fall within one standard error of their respective forecasts.To evaluate forecast accuracy, Relative Mean Square Prediction Error (RMSPE), Mean Absolute Prediction Error (MAPE), and Relative Mean Absolute Prediction Error (RMAPE) were calculated using standard formulae and are presented in Table 3.

$$\text{MAPE} = 1/40\sum_{i=1}^{40}\left|y_{t+i} - \hat{y}_{t+i}\right|$$

$$\text{RMSPE} = 1/40\sum_{i=1}^{40}\left\{\left(y_{t+i} - \hat{y}_{t+i}\right)^2 / y_{t+i}\right\}$$

$$\text{RMAPE} = 1/40\sum_{i=1}^{40}\left\{\left|y_{t+i} - \hat{y}_{t+i}\right| / y_{t+i}\right\}\times 100$$

Table : Validation of Models

Market	MAPE	RMSPE	RMAPE (%)
Amritsar	195.964	204.773	3.5
Bhatinda	323.303	333.535	4.8
Max Price	352.963	366.503	4.7
Min Price	168.629	194.520	3.3
Modal Price	173.679	177.470	3.1

A perusal of above table indicates that in all the price series data, RMAPE is less than 5% indicating the accuracy of the models.

R code for application of ARFIMA model

```
library(forecast)

library(tseries)

data<-read.delim("clipboard")

###### ARFIMA model
```

```
ts<-as.ts(data[,2]) #convert to time series
acf(ts)  # ACF plot
pacf(ts) # PACF plot
adf_test<-adf.test(ts) # stationary plot
train<-ts[c(1:(length(ts)*0.9))] #train data
test<-ts[-c(1:(length(ts)*0.9))] #test data
model<-arfima(ts, drange=c(0, 0.5), estim=c("mle","ls")) #model
Forecast<-forecast(model, h=length(test)) # future forecast
accuracy(Forecast, x=test) ## accuarcy measure
plot(Forecast) ##plot
```

References

Baillie, R. T., Bollerslev, T., Mikkelsen, H.O. (1996). Fractionally integrated generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, **74**, 3–30.

Beran, J. (1994). Maximum likelihood estimation of the differencing parameter for invertible short and long memory autoregressive integrated moving average models. *Journal of the Royal Statistical Society*, **B 57** (4) 659-672

Booth, G.G., F.R. Kaen, and P.E. Koveos. (1982). R/S Analyses of Foreign Exchange Rates under Two International Monetary Regimes. *Journal of Monetary Economics*, **10**, 407-415.

Geweke, J., Porter-Hudak, S. (1983). The estimation and application of long-memory time-series models. *Journal of Time series Analysis*, **4**, 221–238.

Helms, B.P., F.R. Kaen, and R.E. Rosenman. (1984). Memory in Commodity Futures Contracts. *The Journal of Futures Markets*, **10**, 559-567.

Hurvich, C.M., Deo, R. and Brodsky, J. (1998). The mean squared error of Geweke and Porter-Hudak's estimator of the memory parameter of a long-memory time-series. *Journal of Time series Analysis*, **19**, 19-46.

Jin, H. J., and Frechette, D. (2004). Fractional integration in agricultural futures price volatilities. *American Journal of Agricultural Economics*, **86**, 432-443.

Paul, R. K. and Das, M. K. (2010). Statistical modelling of Inland fish production in India. *Journal of the Inland Fisheries Society of India*, **42** (2), 1-7

Paul, R. K. and Das, M. K. (2013). Forecasting of average annual fish landing in Ganga Basin. *Fishing chimes*, **33** (3), 51-54

Paul, R. K., Prajneshu and Ghosh, H. (2013a). Statistical modelling for forecasting of wheat yield based on weather variables. *Indian Journal of Agricultural Sciences*, **83**(2), 180-183.

Paul, R. K., Panwar, S., Sarkar, S. K., Kumar, A. Singh, K. N., Farooqi, S. and Chaudhary, V. K. (2013b). Modelling and Forecasting of Meat Exports from India. *Agricultural Economics Research Review*, **26** (2), 249-256.

Paul, R. K. (2014). Forecasting Wholesale Price of Pigeon Pea Using Long Memory Time-Series Models. Agricultural Economics Research Review, **27**(2), 167-176.

Phillips, P.C.B. and P. Perron (1988). Testing for Unit Roots in Time Series Regression. *Biometrika*, **75**, 335-346.

Robinson, P.M. (1995). Log-periodogram regression of time-series with long-range dependence. *The Annals of Statistics*, **23**, 1048–1072.

Tanaka, K. (1999). The nonstationary fractional unit root. *Econometric Theory*, **15**, 549- 582.

Wavelets Time-Series Analysis

Ranjit Kumar Paul

ICAR-Indian Agricultural Statistics Research Institute
Library Avenue, Pusa, New Delhi-110012
Email: ranjitstat@gmail.com

Introduction

The Autoregressive Integrated Moving Average (ARIMA) methodology developed by Box et al. (2007) has remained one of the most dominant parametric approaches for analyzing time-series data over the past few decades. In this framework, various explanatory variables influence the model "implicitly" through the use of past values of the response variable. However, in many practical situations, especially in complex phenomena, it becomes difficult to assume an appropriate parametric form. In such scenarios, nonparametric approaches offer better flexibility and modeling capability. One such powerful and emerging nonparametric approach is Wavelet Analysis (Vidakovic, 1999; Percival and Walden, 2000). Despite the growing number of theoretical papers on wavelet methods, their actual implementation and application to empirical data still pose significant challenges to researchers.

Wavelet analysis can be conducted in two primary ways: in the time domain and the frequency domain. Time-domain wavelet analysis typically involves techniques like wavelet thresholding. For instance, Sunilkumar and Prajneshu (2004) effectively applied wavelet thresholding for modeling and forecasting monthly rainfall across meteorological subdivisions in Eastern Uttar Pradesh, India. On the other hand, frequency-domain wavelet analysis is suitable for identifying and analyzing trends and cycles within time-series data. Almasri et al. (2008) proposed a new statistical test based on wavelet decomposition for detecting the presence of a trend in time-series data. A key difficulty in trend testing arises due to the presence of autocorrelation among residuals, which makes standard tests based on ordinary least squares (OLS) regression unreliable. In such cases, the autocovariance structure often exhibits slow decay, indicating long memory in the series. Wavelet analysis proves useful here because it can better match the structure of long-memory processes.

One of the main advantages of wavelet transformation is that it changes the behavior of autocovariance functions. While the original time-domain series may be strongly autocorrelated, the transformed wavelet series often shows much faster hyperbolic decay in autocovariances. This makes the transformed series almost uncorrelated in the wavelet domain, thereby improving the reliability of statistical tests and trend estimation. In fact, this property of wavelets makes them highly useful for modeling long memory and complex temporal dependencies. Rainfall plays a critical role in determining agricultural performance in any country, and particularly in India where monsoon rainfall largely influences crop production and food security. Therefore, accurate modeling and prediction of rainfall is of utmost importance for agricultural planning, decision-making, and policy formulation. In the Indian context, Rajeevan et al. (2004) have provided a comprehensive review of rainfall prediction models using multiple and power regression techniques, highlighting various modifications that have been made over time, particularly in identifying relevant predictor variables and improving model performance.In recent years, there has been growing interest in the application of wavelet-based models to study agricultural time-series data. Several studies have employed wavelet methods to improve forecasting accuracy and capture hidden structures in data. Examples include the works of Anjoy and Paul (2017), Anjoy et al. (2017), Paul et al. (2013), Paul (2015), Paul and Birthal (2015), Sarkar et al. (2019), Paul et al. (2020), Paul et al. (2021), Singla et al. (2021), Paul and Garai (2021), Paul and Mitra (2021), Paul and Garai (2022), and Paul et al. (2022). These studies reflect a wide array of applications ranging from price forecasting to climate modeling and agricultural risk analysis. The present lecture aims to focus on the application of wavelet analysis in the frequency domain, particularly for estimation and testing of significant trends in India's monsoon rainfall data covering the period from 1979 to 2006. This approach will not only help in detecting long-term trends in the monsoon but also assist in better understanding of the rainfall dynamics that influence Indian agriculture.

Basics of Wavelets

The term *wavelet* refers to a class of basic functions characterized by a unique mathematical structure, which underpins their key properties and broad applicability in statistical analysis. Wavelets serve as fundamental building blocks, much like the sine and cosine functions in Fourier analysis. Similar to these trigonometric functions, a wavelet oscillates around zero, a property that qualifies it as a "wave."

However, unlike sine and cosine waves that extend indefinitely, wavelet functions exhibit a localized nature their oscillations diminish and converge to zero. This dampening behavior gives rise to the term *wavelet*, indicating a small or finite-duration wave.

Formally, let $\psi(.)$ be a real-valued function defined over the real line R. To qualify as a wavelet, this function must satisfy two fundamental properties:

1. Admissibility condition – ensuring the existence of the inverse wavelet transform.
2. Zero mean – the function must integrate to zero over the entire real line.

These foundational properties allow wavelets to efficiently represent localized features in both time and frequency domains, making them powerful tools for time series analysis, signal processing, and data compression.

(i) The integral of $\psi(.)$ is zero:

$$\int_{-\infty}^{\infty} \psi(u)\, du = 0$$

(ii) The square of $\psi(.)$ integrates to unity:

$$\int_{-\infty}^{\infty} \psi^2(u)\, du = 1$$

Then the function $\psi(.)$ is called a wave.

Discrete Fourier transform

The transformation of a function into its wavelet components shares many similarities with its transformation into Fourier components. An understanding of wavelet analysis typically begins with a discussion of the classical Fourier transformation. The concept, introduced by the French mathematician Jean-Baptiste Fourier, establishes that any square-integrable function defined on the interval [−π,π] can be decomposed into a series of component functions derived from standard trigonometric bases. Specifically, a function fff is said to belong to the square-integrable space $L^2[a, b]$ if it satisfies the condition of finite energy over the interval, expressed as:

$$\int_a^b f^2(x) dx < \infty$$

Fourier's results states that any function $f \in L^2[-\pi, \pi]$ can be expressed as an infinite sum of dilated cosine and sine functions given by

$$f(x) = \frac{1}{2} a_0 + \sum_{j=1}^{\infty} \left(a_j \cos(jx) + b_j \sin(jx) \right) \tag{1}$$

where

$$a_j = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos jx.dx \qquad j = 0, 1, 2,\ldots$$

$$b_j = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin jx.dx \qquad j = 1, 2,\ldots$$

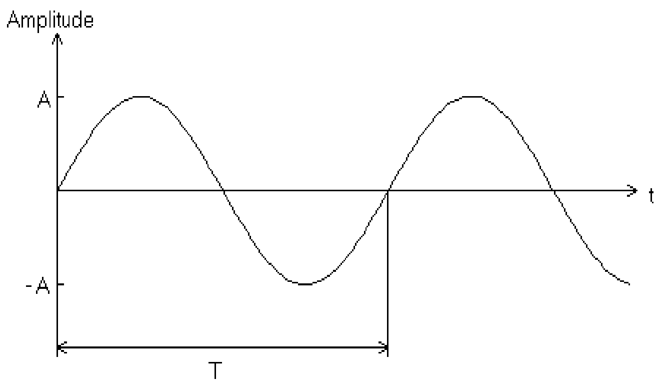
The series expansion is regarded as a transform, taking a function f into a set of coefficients a_j and b_j. The *Fourier series expansion* is extremely useful in that any L^2 function can be written in terms of very simple building block functions: sines and cosines, because the set of functions $\{\sin(j.), \cos(j.), j=1,2,\dots\}$, together with the constant function, form a *basis* for the function space $L^2[-\pi, \pi]$ which is orthonormal. A sequence of functions $\{f_j\}$ are orthonormal if the f_j*'s* are pairwise orthogonal and if $\|f_j\|=1$, for all j.

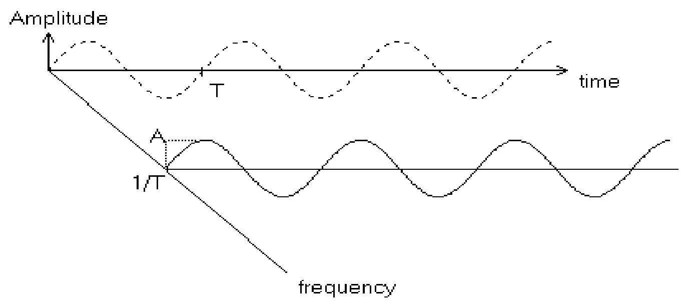
Wavelet analysis versus Fourier analysis

Wavelet analysis and Fourier analysis share a fundamental similarity in that both techniques aim to express a function as a linear combination of basis functions. In Fourier analysis, these basis functions are complex exponentials of the form {eiwx=coswx+isinwx}, while in wavelet analysis, the basis functions are wavelets denoted as $\{\psi_{j,k}\}$. A key distinction between the two lies in their indexing: Fourier basis functions are indexed by a single frequency parameter ω, whereas wavelet basis functions are indexed by two parameters—scale (j) and position (k). This means that wavelets provide a much richer and more flexible set of basis functions compared to the relatively limited set in Fourier analysis. The essential difference lies in how the two analyses handle frequency and time (or location). In classical Fourier analysis, the sine and cosine functions offer precision in the frequency domain, but they lack localization in time. That is, while they identify which frequencies are present in a signal, they cannot determine when these frequencies occur. Wavelets, on the other hand, offer dual localization. Through translation (shifting in time) and dilation (scaling in frequency), wavelet basis functions can capture both the frequency content and its location in time. This makes wavelet analysis particularly powerful for studying signals that exhibit time-varying behavior. One of the standout features of wavelet transforms is their locality. The wavelet coefficients are dependent only on the behavior of the function in the vicinity of each time point. As a result, if the function contains abrupt changes, discontinuities, or spikes—known as singularities—these features will only affect the wavelet transform locally, around the singularity. This is in stark contrast to Fourier transforms, which are global in nature: a single discontinuity in the signal can influence the Fourier coefficients across the entire domain. Therefore, when analyzing data that exhibit local irregularities or non-stationary patterns, wavelet analysis provides a more appropriate and effective tool.

Time domain versus Frequency domain

The most widely used method to represent signals and waveforms is in the time domain. However, many analytical techniques work primarily in the frequency domain. Understanding how a signal appears in the frequency domain can be somewhat challenging. In essence, the frequency domain is just an alternative representation of a signal. To illustrate, imagine a simple sine wave as an example. This sine wave is usually plotted on a time-amplitude graph, which defines the time plane. Now, if we introduce another axis to represent frequency, the sine wave can be visualized in three dimensions.





In time-frequency analysis, the frequency-amplitude plane serves a role analogous to that of the time-amplitude plane in representing a signal. The frequency plane is orthogonal to the time plane, and both intersect along the common amplitude axis. When the frequency spectrum of a signal is displayed, it is essentially a representation of this frequency plane.A time-domain signal can be seen as the projection of a sinusoidal wave onto the time-amplitude plane. Meanwhile, the sinusoid itself exists at a specific distance along the frequency axis, corresponding to its frequency, which is the inverse of its period. The projection of this waveform onto the frequency plane appears as a vertical line at the given frequency, rising to a height equal to the sinusoid's amplitude. Hence, the sinusoid simultaneously exhibits a waveform in the time domain and a spike in the frequency domain.

Discrete Wavelet Transform (DWT)

Wavelet transforms have evolved through two major approaches: the Continuous Wavelet Transform (CWT) and the Discrete Wavelet Transform (DWT). The CWT is suited for analyzing continuous signals defined over the real number line, while the DWT is designed for discrete signals—those indexed over integers, such as digital time-series data.

The DWT decomposes a time-series into components associated with both low and high frequency bands, enabling a multi-resolution representation of the data. This allows for effective modeling and analysis, especially in the presence of time-varying patterns, by using the inverse DWT to reconstruct the original signal from its components.

Several features make DWT a powerful tool for time-series analysis:

Time-Scale Localization:

DWT re-expresses a time-series through coefficients associated with specific dyadic scales2^{j-1} and corresponding time positions. These coefficients preserve all information in the original series, allowing for perfect reconstruction.

Energy Decomposition:

DWT partitions the signal energy across different scales and times, much like analysis of variance (ANOVA) in statistics. This makes it useful for identifying how energy (or variability) is distributed across frequencies and over time.

Decorrelation Capability:

DWT effectively decorrelates many types of real-world time-series, especially those common in physics, engineering, and finance. This property makes it valuable in statistical modeling, where uncorrelated components simplify analysis.

Computational Efficiency:

The DWT can be computed efficiently using a recursive method known as the Pyramid Algorithm, which is even faster than the widely known Fast Fourier Transform (FFT).

In summary, the DWT serves as a versatile and computationally efficient method for analyzing time-series, capturing both short-term fluctuations and long-term trends by localizing features in both time and frequency domains.

The first stage for computing the DWT simply consists of transforming the time-series $\mathbf{X}$ of length $N = 2^J$ into the $N/2$ first level wavelet coefficients $\mathbf{W}_1$ and the $N/2$ first level scaling coefficients **V**$_1$. Precisely, to obtain unit scale wavelet coefficients, time-series $\{X_t : t = 0,\ldots, N-1\}$ is circularly filtered with filter h_l, $l = 1, 2, \ldots, L-1$, where L is the width of the filter and must be an even integer. For h_l to have width L, it must satisfy the conditions: $h_0 \neq 0$ and $h_{L-1} \neq 0$. Now define $h_l = 0$ for $l < 0$ and $l \geq L$ so that h_l is actually an infinite sequence wit at most L nonzero values. A wavelet filter must satisfy the following three basic properties:

$$\sum_{l=0}^{L-1} h_l = 0,\ \sum_{l=0}^{L-1} h_l^2 = 1 \text{ and } \sum_{l=0}^{L-1} h_l h_{l+2n} = \sum_{l=-\infty}^{\infty} h_l h_{l+2n} = 0,$$

for all nonzero integers n. Compute

$$2^{1/2}\tilde{W}_{1,t} = \sum_{l=0}^{L-1} h_l X_{(t-l)\bmod N}\ ,\ t = 0, 1, \ldots, N-1. \tag{2}$$

Now define $N/2$ wavelet transforms for unit scale corresponding to $t=0,\ldots,N/2-1$ as

$$W_{1,t} = 2^{1/2}\tilde{W}_{1,2t+1} = \sum_{l=0}^{L-1} h_l X_{(2t+1-l)\bmod N}\ , \tag{3}$$

This procedure is called "Downsampling" procedure. To obtain first stage scaling coefficients, define scaling filter $g_l = (-1)^{l+1} h_{L-1-l}$.
Then the first level scaling coefficients are

$$V_{1,t} = 2^{1/2}\tilde{V}_{1,2t+1} = \sum_{l=0}^{L-1} g_l X_{(2t+1-l)\bmod N} \tag{4}$$

The second stage of Pyramid algorithm consists of treating $\{V_{1,t}\}$ in the same way as $\{X_t\}$ was treated in the first stage. Then we circularly filter $\{V_{1,t}\}$ separately with $\{h_l\}$ and $\{g_l\}$ and subsample to produce two new series, namely

$$W_{2,t} = \sum_{l=0}^{L-1} V_{1,(2t+1-l)\bmod N/2} \tag{5}$$

$$V_{2,t} = \sum_{l=0}^{L-1} V_{1,(2t+1-l)\bmod N/2}\ ,\ t=0,1,\ldots,N/4-1. \tag{6}$$

Above procedure is repeated J times to obtain 2^J DWT's. There are $J-2$ subsequent stages to the Pyramid algorithm. For $j = 3,\ldots, J$, the j^{th} stage transforms **V**$_{j-1}$ of length $N/2^{j-1}$ into **W**$_j$ and **V**$_j$ each of length $N/2^j$. At the j^{th} stage, the elements of **V**$_{j-1}$ are filtered separately with wavelet filter $\{h_l\}$, and scaling filter $\{g_l\}$. The filter outputs are subsampled to form respectively **W**$_j$ and **V**$_j$. The elements of **V**$_j$ are called the scaling coefficients for level j, while those of **W**$_j$ contain the desired wavelet coefficients for level j. At the end of J^{th} stage, the DWT coefficient **W** is formed by concatenating the $J + 1$ vectors.

Let **P** be an $N \times N$ real valued matrix defining the DWT and satisfying the orthonormality property **P**`**P** = **I**$_N$, where **I**$_N$ is the $N\times N$ identity matrix. Then the DWT (***W***) of the time-series vector **X** may be computed by **W** = **P X**. Now the elements of the vector **W** are decomposed into J+1 subvectors. The first J subvectors contains all of the DWT coefficients for scale τ_j. Then **W** can be written as

$$\mathbf{W} = [\mathbf{W}_1' \, \mathbf{W}_2' \ldots \mathbf{W}_J' \, \mathbf{V}_J']'$$

Multiresolution Analysis (MRA)

Consider the wavelet synthesis of **X**

$$\mathbf{X} = \mathbf{P}'\mathbf{W} = \sum_{\mathbf{j}=\mathbf{1}}^{\mathbf{J}} \mathbf{P}_\mathbf{j}' \mathbf{W}_\mathbf{j} + \mathbf{Q}_\mathbf{J}' \mathbf{V}_\mathbf{J}, \tag{7}$$

where $\mathbf{P}_j$ and $\mathbf{Q}_J$ matrices are defined by partitioning the rows of **P** commensurate with the partitioning of **W** into $\mathbf{W}_1$, …, $\mathbf{W}_J$ and $\mathbf{V}_J$. Thus the $N/2 \times N$ matrix $\mathbf{P}_1$ is formed from the n = 0 up to n = N/2-1 rows pf **P**; the $N/4 \times N$ matrix $\mathbf{P}_2$ is formed from the n = N/2 up to n = 3N/4-1 rows; and so forth, until we come to the $1 \times N$ matrices $\mathbf{P}_J$ and $\mathbf{Q}_J$, which are the last two rows of **P**.

Thus

$$\mathbf{P} = [P_1 \, P_2 \ldots P_J \, Q_J]'$$

Now define $\mathbf{D}_j = \mathbf{P}`_j \, \mathbf{W}_j$ for $j = 1,\ldots, J$, which is an N dimensional column vector whose elements are associated with changes in **X** at scale τ_j; i.e., $\mathbf{W}_j = \mathbf{P}_j\mathbf{X}$ represents the portion of the analysis **W** = **PX** attributable to scale τ_j, while $\mathbf{P}`_j \, \mathbf{W}_j$ is the portion of the synthesis **X** = **P`W** attributable to scale τ_j. Let $\mathbf{S}_J = \mathbf{Q}`_J\mathbf{V}_J$ which has all its elements equal to the sample mean $\overline{X}$. Then it can be seen that

$$\mathbf{X} = \sum_{j=1}^{J} D_j + \mathbf{S}_J, \tag{8}$$

which defines a multiresolution analysis (MRA) of **X**; *i.e.*, the time-series **X** is expressed as the sum of a constant vector $\mathbf{S}_J$ and J other vectors $\mathbf{D}_j$, $j = 1,\ldots, J$ each of which contains a time-series related to variations in **X** at a certain scale. $\mathbf{D}_j$ is called the jth level wavelet detail.

Estimation of Trend by Wavelets

Sometimes it is important to decompose a time-series into different components of variations like, low frequencies (trend), and high-frequency (noise) components. And the multiresolution analysis is used for decomposing and describing the low frequencies and high-frequency components in the data in a scale by scale basis. Consider the following model for a time-series data $\{X_t\}$:

$$X_t = \mu + T_t + Z_t, \quad t = 0, \ldots, N-1, \tag{9}$$

where μ is a constant term, T_t is an unknown deterministic polynomial trend function of order r, Z_t is a residual term which is a long-memory process defined by $(1-B)^{\delta} Z_t = \varepsilon_t$, where, δ is the long memory parameter, $\{\varepsilon_t\}$ is a Gaussian white noise process with mean zero and $\sigma_{\varepsilon}^2 > 0$. Here, B, is the back shift operator such that $BZ_t = Z_{t-1}$.

Now, since $\mathbf{W} = \left[\mathbf{W}_1' \, \mathbf{W}_2' \ldots \mathbf{W}_J' \, \mathbf{V}_J'\right]'$, the vector $\mathbf{W}$ can be written as sum of two vectors: $\mathbf{W} = \mathbf{W_w} + \mathbf{W_s}$, where $\mathbf{W_w}$ is an $N \times 1$ vector containing the wavelet coefficients and zeros at all other locations, and $\mathbf{W_s}$ is an $N \times 1$ vector containing the scaling coefficients and zeros at all other locations. Since $\mathbf{X} = \mathbf{P`W}$, therefore,

$$\mathbf{X} = \mathbf{P`W} = \mathbf{P`\,W_s} + \mathbf{P`\,W_w} = \hat{T} + \hat{Z}, \tag{10}$$

where $\hat{T}$ is an estimator of the polynomial trend T at level J, while $\hat{Z}$ is the estimate of residual Z. The issue of choosing the level of the estimate depends on the goal of application. J should be chosen small for detecting the local trends and cycles. In other applications, J is set to be large, if the aim is to detect the global trend.

The orthonormality of the matrix $\mathbf{P}$ implies that the DWT is an energy preserving transform so that

$$\|\mathbf{X}\|^2 = \|\mathbf{W}\|^2 = \sum_{t=1}^{N} X_t^2 \tag{11}$$

Given the structure of the wavelet coefficients, the energy in $\mathbf{X}$ is decomposed, on a scale by scale basis, via

$$\|\mathbf{X}\|^2 = \|\mathbf{W}\|^2 = \sum_{j=1}^{J} \|\mathbf{W}_j\|^2 + \|\mathbf{V}_J\|^2 \tag{12}$$

so that $\|\mathbf{W}_j\|^2$ represents the contribution to the energy of $\{X_t\}$ due to changes at scale τ_j. whereas $\|\mathbf{V}_J\|^2$ represents the contribution due to variations at scale τ_J. So the estimated variance of the time-series in terms of wavelet and scaling coefficients can be expressed as:

$$\sigma_X^2 = \frac{1}{N}\sum_{t=1}^{N}\left(X_t - \bar{X}\right)^2 = \frac{1}{N}\|\mathbf{W}\|^2 - \bar{X}^2 = \frac{1}{N}\sum_{j=1}^{J}\|\mathbf{W}_j\|^2 + \frac{1}{N}\|\mathbf{V}_J\|^2 - \bar{X}^2$$

$$= \sum_{j=1}^{J} \hat{v}_X^2(\tau_j) + \hat{\sigma}_{S_J}^2 \tag{13}$$

where $\hat{v}_X^2(\tau_j)$ is the estimated variance of the wavelet coefficients at scale τ_j, and $\hat{\sigma}_{S_J}^2$ is the estimated variance of the trend.

For testing the null hypothesis $H0$: *Trend* $= 0$, Almasri *et al*. (2008) proposed a test statistic that can discriminate between this null hypothesis and the alternative hypothesis $H1$: *Trend* $\neq 0$ is defined as follows:

$$G = \frac{\hat{\sigma}^2_{S_J}}{\sum_{j=1}^{J} \hat{v}^2_X(\tau_j)} \tag{14}$$

The test statistic $(N - N/2^J)/(N/2^J - 1)\mathbf{G}$ will follow an F distributed with $(N/2^J - 1)$ and $(N - N/2^J)$ degrees of freedom, under the assumption of normally distributed scaling coefficients. However, when errors deviate from normality or exhibit dependency, the true distribution of this test statistic becomes unknown. In such cases, it is essential to compute empirical critical values through simulation experiments to better understand the distributional behavior of the statistic under non-standard conditions. Wavelet-based estimation has a distinct advantage over the Fourier transform because of its localization in both time and frequency domains. This characteristic allows the wavelet estimates to vary with time t which is particularly useful for analyzing long memory processes. Such processes often manifest as localized trends and cycles, which may eventually disappear, making them difficult to capture with global methods like Fourier analysis. Wavelets, however, can isolate these transient features effectively across different scales J providing richer insights into the structure of variability. An important aspect of wavelet analysis is the selection of an appropriate wavelet filter. The choice depends on the structural characteristics of the data under investigation. For instance, the Haar wavelet, a piecewise constant function, is well-suited for detecting structural breaks or sharp discontinuities in a series. This is because it preserves such features without smoothing them out. On the other hand, smoother wavelets like those with length $L > 2$ (e.g., Daubechies wavelets) provide better continuity but may blur discontinuities, making them less effective for change-point detection. In general, wavelets with wider support (large L) offer smoother approximations but lower spatial localization, whereas wavelets with narrow support (small L) are highly localized but less smooth.

Basis Functions

Just as any two-dimensional vector (x,y) can be decomposed into a linear combination of the basis vectors $(1,0)$ and $(0,1)$, functions can also be expressed as linear combinations of basis functions. In Fourier analysis, these basis functions are sines and cosines, which satisfy the property of orthogonality—their inner product over a given interval sums to zero when different. By selecting appropriate combinations of these trigonometric functions, one can represent a wide class of functions. Wavelet analysis follows a similar principle using wavelet families to construct orthonormal bases. A wide variety of wavelet families have been developed, each suited to different types of data and analysis goals. Two of the most widely used wavelet systems are the Haar wavelet and the Daubechies wavelet. These enable the generation of orthonormal wavelet bases tailored to specific function spaces, allowing for flexible and efficient representation and analysis of complex signals.

The Haar System

The simplest wavelet basis for L^2(R) is the Haar basis. The Haar function is a bonafide wavelet, though not used much in practice, uses a mother wavelet given by

$$\psi(x) = \begin{cases} 1, & 0 \le x < ½, \\ -1, & ½ \le x \le 1, \\ 0, & \text{otherwise} \end{cases}$$

The Haar wavelet is piecewise constant over intervals of length one-half and can be expressed by a picture as follows (Fig.1).

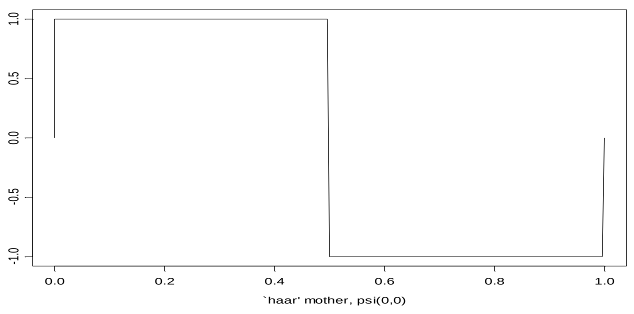


Fig. 1. The Haar function

Haar wavelets are characterized by their compact support, meaning they vanish outside a finite interval, which allows for good time localization. However, when analyzing functions that require higher levels of regularity or smoothness, the Haar system becomes unsuitable. This is primarily because Haar wavelets lack continuous differentiability and exhibit poor decay of coefficients at infinity, making them less effective for representing smooth functions. These limitations reduce their applicability in many data analysis contexts. To address these drawbacks, Daubechies (1992) introduced a family of smooth wavelet bases by replacing the Haar scaling function with one exhibiting greater regularity. The resulting Daubechies wavelets offer significantly improved behavior for analyzing smooth or more complex functions, thereby broadening their utility in practical applications.

Daubechies Wavelet Bases

By imposing an appealing set of regularity conditions, Daubechies (1992) came up with a useful class of wavelet filters, all of which yield a DWT in accordance with the notion of differences of adjacent averages. The definition for this class of filters can be expressed in terms of the squared gain function for the associated Daubechies scaling filters g_l, $l = 0, \ldots, L-1$:

$$G^D(f) \equiv 2\cos^L(\pi f)\sum_{l=0}^{L/2-1}\binom{L/2-1+l}{l}\sin^{2l}(\pi f),$$

where L is a positive even integer.

Using the relationship $H^D(f) = G^D(f+1/2)$, the corresponding Daubechies wavelet filters have squared gain functions satisfying

$$H^D(f) \equiv 2\sin^L(\pi f)\sum_{l=0}^{L/2-1}\binom{L/2-1+l}{l}\cos^{2l}(\pi f)$$

$H^D(.)$ can be considered as the squared gain function of the equivalent filter for a filter cascade. Apart from the above, there are other families of smooth wavelet bases that provide compactly supported orthonormal wavelets and are continuously differentiable, like those proposed by Stromberg, Meyer and Battle (Ogden, 1997).

An Illustration (Ghosh *et al* (2010), and Paul *et al* (2011))

To estimate the trend using wavelet methodology, Indian monsoon rainfall data from 1879 to 2006 is used.Monsoon rainfall is calculated as the total daily rainfall from June 1st to September 30th each year.The data set is collected from the official website of the Indian Institute of Tropical Meteorology, Pune (www.tropmet.res.in). This rainfall data shows cyclical fluctuations along with a possible declining trend. The trend in monsoon rainfall has been estimated using both the ARIMA method and twavelet approach. Various types of wavelets have been applied to analyze the data on a scale-by-scale basis. This helps to highlight the localized variations present in the dataset.

Modelling of rainfall data in the framework of autoregressive process

Assuming presence of deterministic linear trend in the rainfall series, following model is fitted:

$$Y_t = \mu + \delta t + \varepsilon_t,\ t = 1, 2, \ldots, T \qquad (18)$$

where ε_t's are uncorrelated with zero mean and constant variance σ_ε^2. Let

$$\hat{e}_t = Y_t - \hat{\mu} - \hat{\delta}\, t$$

The fitted trend equation is obtained as:

$$Y_t = 863.718 - 0.234\, t$$
$$(14.226) \quad (0.191)$$

where the values within brackets ( ) denote corresponding standard errors of estimates. The trend is not significant at 5% level of significance. The graph of trend is displayed in Fig. 3.

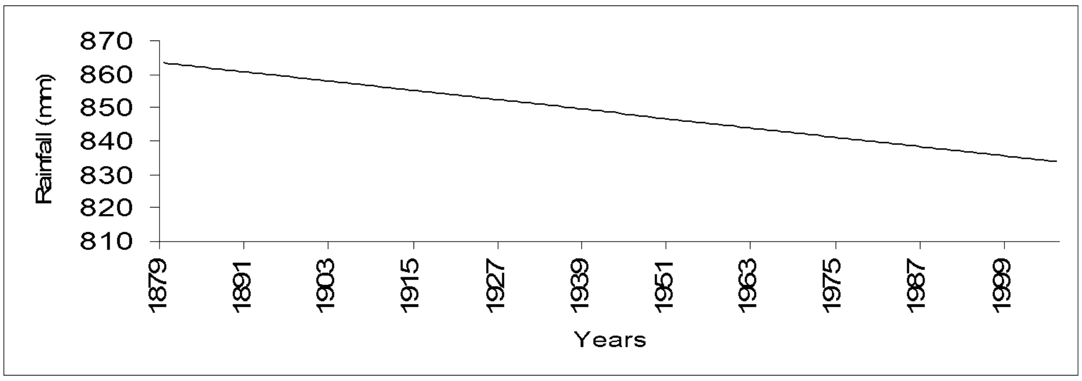


Fig. 3. Trend in Indian monsoon rainfall data

Trend analysis through wavelet approach

The discrete wavelet transform (DWT) and multiresolution analysis (MRA) were performed using both the Haar wavelet and the Daubechies 4 (D4) wavelet. The resulting DWT coefficients are illustrated in Figures 5 and 6. These coefficients represent differences of various orders of weighted averages of segments of the time series X_t localized in time. The wavelet coefficients are displayed as upward or downward bars, whose magnitudes correspond to the strength of the localized features. At level 1 (the lowest resolution), the number of coefficients is half the original number of data points, and this number continues to halve at each subsequent level (Nason and Sachs, 1999).

The upper levels of the plot contain high-frequency components, while the lower levels represent low-frequency (smooth) components. Since wavelet coefficients vary over time, they effectively capture time-localized changes in the series. Abrupt structural changes or discontinuities can be detected by observing vertical clusters of large coefficients across resolution levels. These coefficients allow the original time series to be perfectly reconstructed using the inverse discrete wavelet transform (IDWT). These patterns are further corroborated by the multiresolution analysis (MRA) of the time series, as shown in Figures 7 and 8.

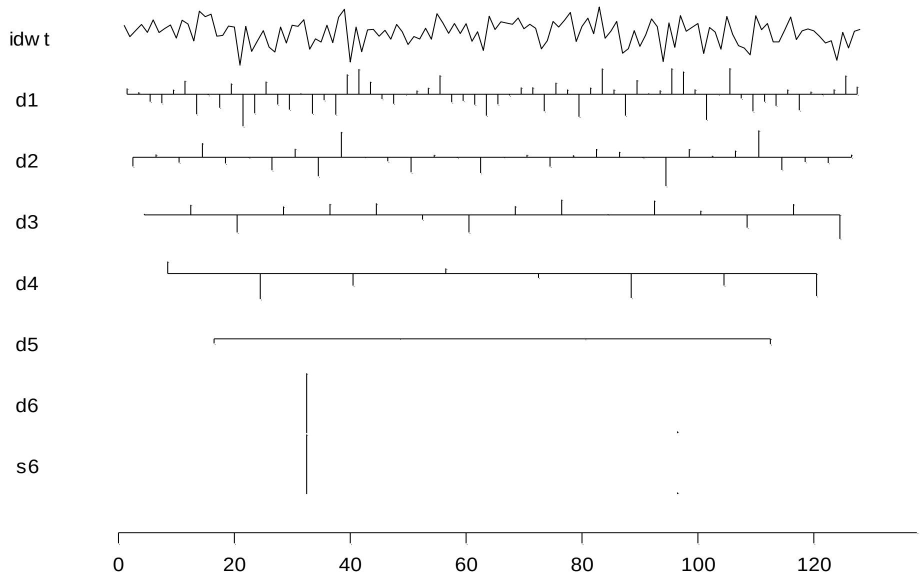


Fig. 5. DWT by D4 wavelet at level 6

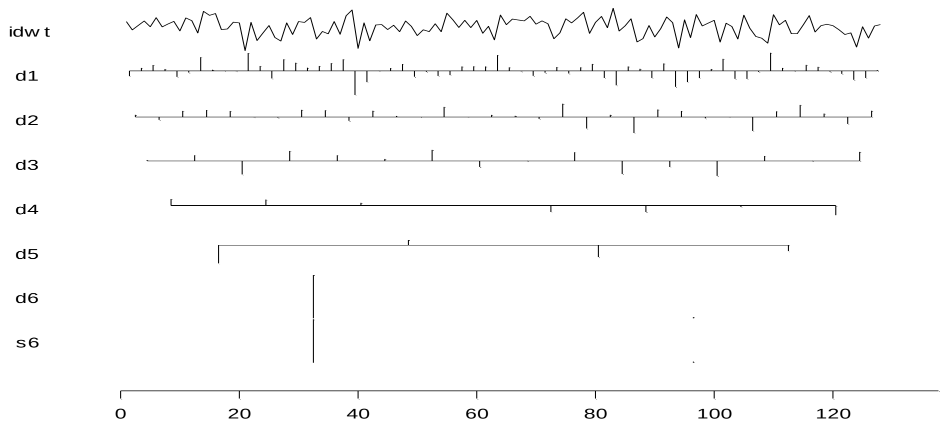


Fig. 6. DWT by Haar wavelet at level 6

The estimate of trend of the rainfall data computed by Haar and D4 wavelets for the levels 6 are given below (Figure 9-10). As the level increases the declining global trend present in the data set is depicted clearly.

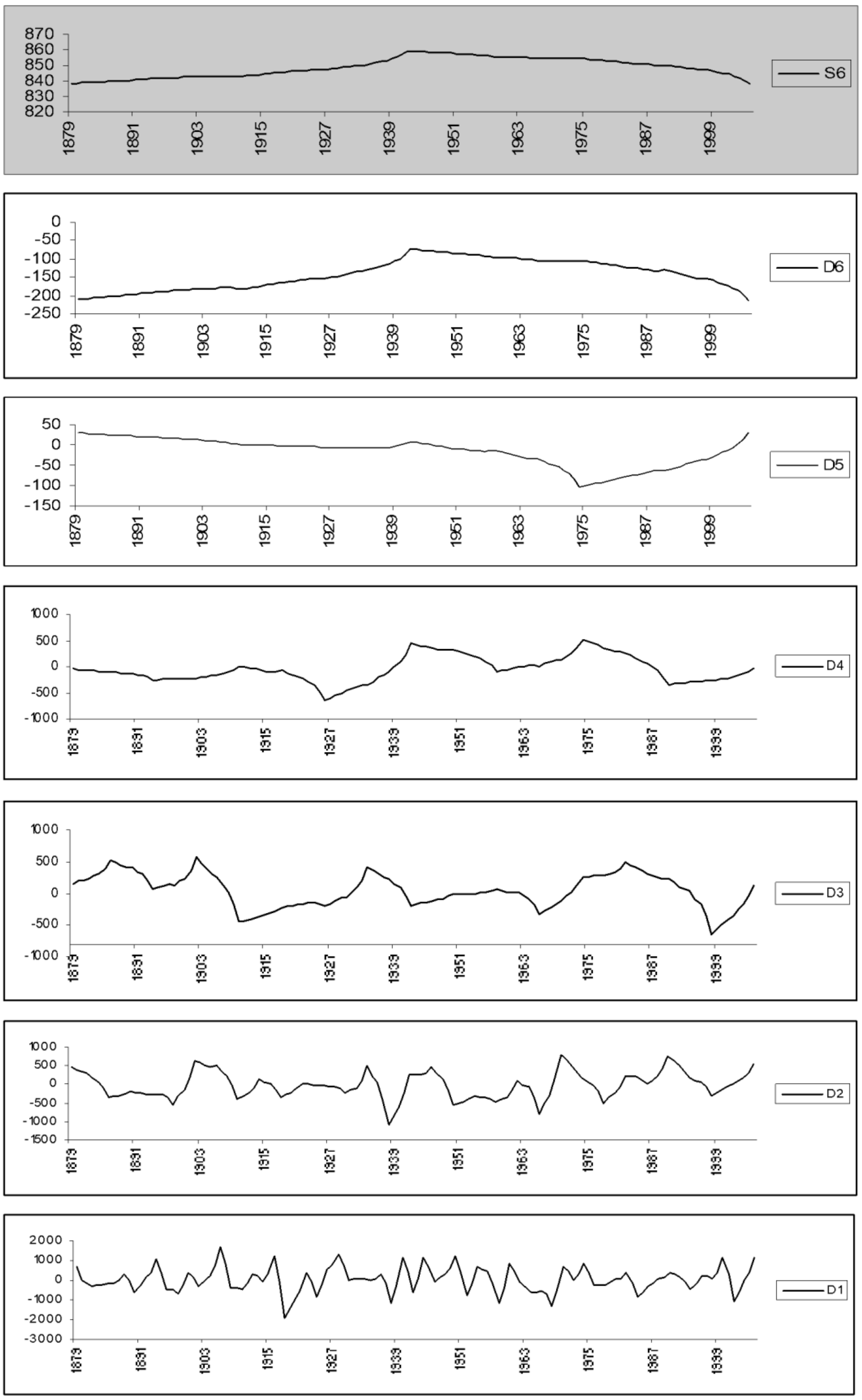


Fig. 7. MRA by D4 wavelet at level 6

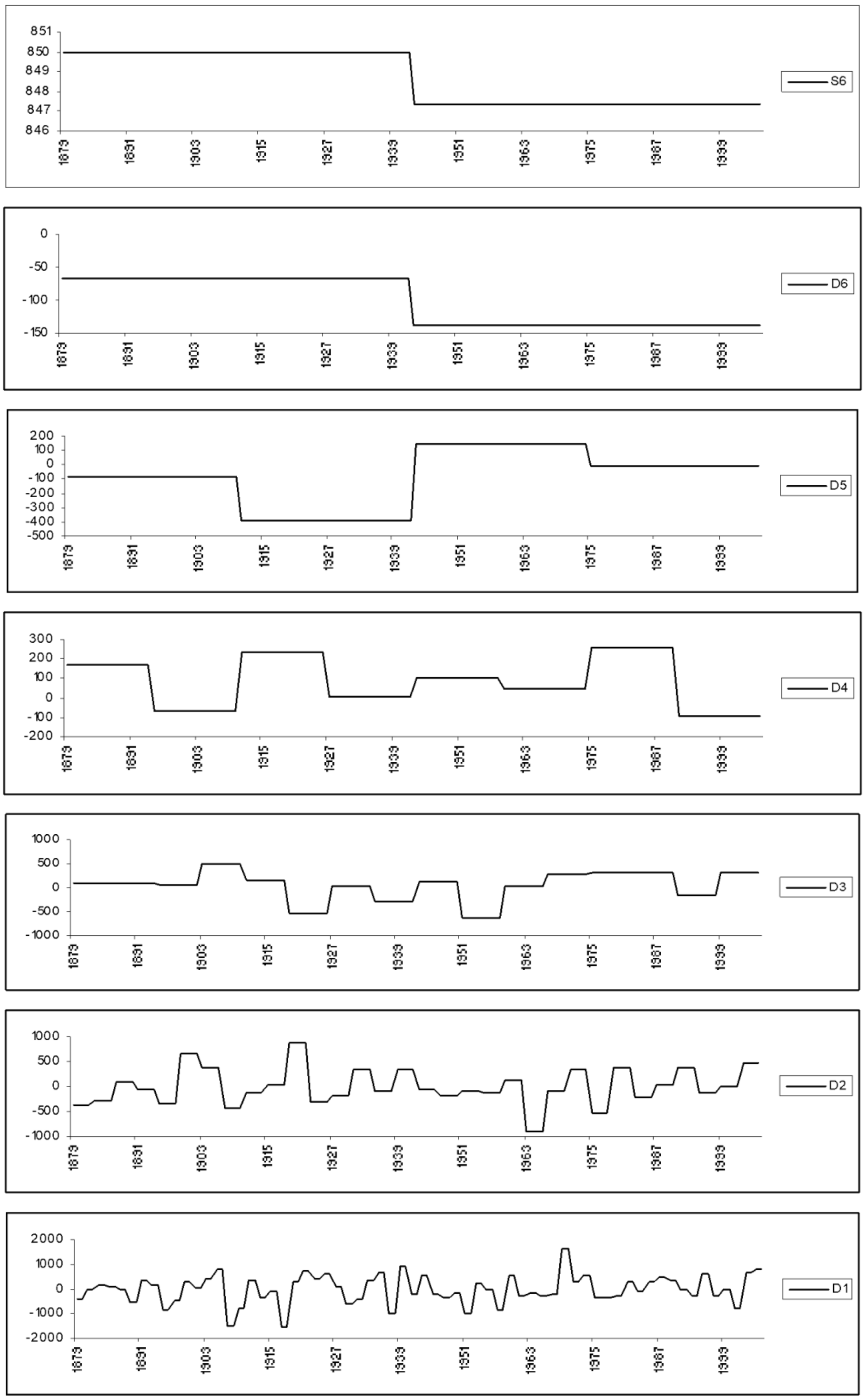


Fig. 8. MRA by Haar wavelet at level 6

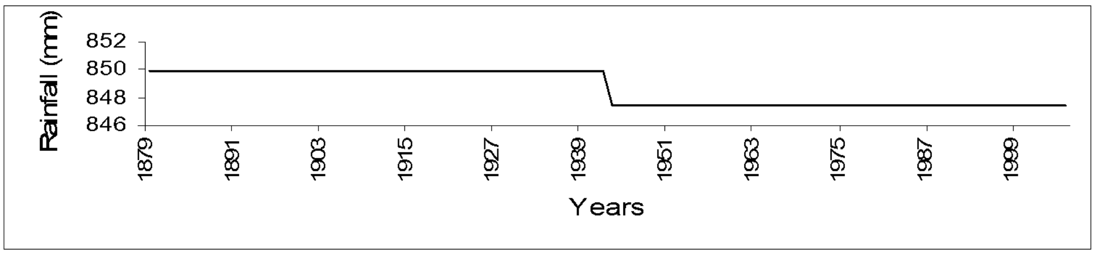


Fig. 9. Estimate of trend by Haar wavelet at level *6*

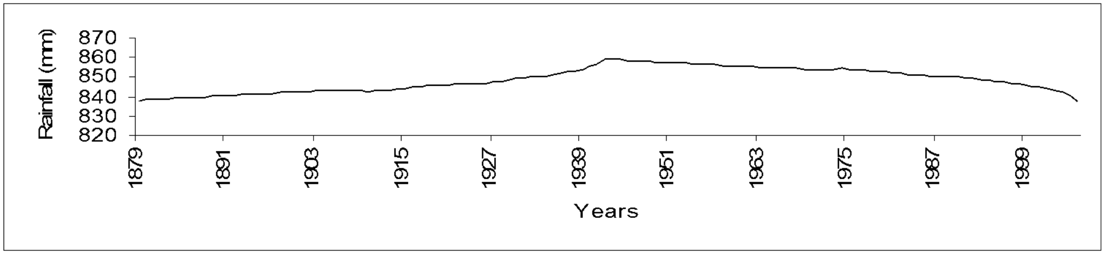


Fig. 10. Estimate of trend by Daubechies (D4) wavelet at level *6*

The discrete wavelet transform (DWT) and multiresolution analysis (MRA) of India's monsoon rainfall time-series data reveal differential behaviours at different time epochs at different scales. Two wavelets namely; Daubechies (D4) and Haar wavelets are used for estimation of trend in the rainfall data. It is found that the monsoon rainfall in India is showing a declining trend over the years, which can have very serious repercussions from "Global Warming" point of view. This important feature, however, could not be captured by ARIMA methodology.

Recently, the algorithm of wavelet based models including stochastic models and machine learning techniques have been proposed and relevant R packages have been developed for the ease of application in real data. Few of the R packages are:

https://CRAN.R-project.org/package=WaveLetLongMemory
https://CRAN.R-project.org/package=WaveletArima
https://CRAN.R-project.org/package=WaveletANN
https://CRAN.R-project.org/package=WaveletGARCH
https://CRAN.R-project.org/package=WaveletSVR
https://CRAN.R-project.org/package=WaveletRF

R code for application of ARIMA model

```
library(WaveletArima)
train<-ts[c(1:(length(ts)*0.9))] #train data
test<-ts[-c(1:(length(ts)*0.9))] #test data

###Wavelet ARIMA
```

```
WaveletARIMA<-WaveletFittingarma(ts=train, filter
='la8',Waveletlevels=floor(log(length(train))),
MaxARParam=5,MaxMAParam=5,NForecast=length(test))

Fitted<-WaveletARIMA$FinalPrediction
Forecast<-WaveletARIMA$Finalforecast

# Accuracy
MAPE_train<-MLmetrics::MAPE(Fitted, train)
MAPE_test<-MLmetrics::MAPE(Forecast, test)

RMSE_train<-MLmetrics::RMSE(Fitted, train)
RMSE_test<-MLmetrics::RMSE(Forecast, test)


### Wavelet ANN
library(WaveletANN)
WaveletANN<-
WaveletFittingann(ts=train,Waveletlevels=floor(log(length(train))),Filter='d4',
                              nonseaslag=5,hidden=3,NForecast=length(test))

Fitted<-WaveletANN$FinalPrediction
Forecast<-WaveletANN$Finalforecast
# Accuracy
MAPE_train<-MLmetrics::MAPE(Fitted[-c(1:5)], train[-c(1:5)])
MAPE_test<-MLmetrics::MAPE(Forecast, test)

RMSE_train<-MLmetrics::RMSE(Fitted[-c(1:5)], train[-c(1:5)])
RMSE_test<-MLmetrics::RMSE(Forecast, test)
```

References:

Almasri, A., Locking, H. and Shukur, G. (2008). Testing for climate warming in Sweden during 1850–1999, using wavelets analysis. *J. Appl. Stat.*, **35**, 431-43.

Box, G. E. P., Jenkins, G. M. and Reinsel, G. C. (2007). *Time-Series Analysis: Forecasting and Control*. 3rd edition. Pearson education, India.

Daubechies, I. (1992). *Ten Lectures on Wavelets*. SIAM, Philadelphia.

Ghosh, H., Paul, R. K. and Prajneshu, (2010). Wavelet Frequency Domain Approach for Statistical Modeling of Rainfall Time-Series Data. *Journal of Statistical Theory and Practice*, **4** (4)

Kulkarni, J. R. (2000). Wavelet analysis of the association between the southern oscillation and Indian summer monsoon. *Int. J. Climatol*., **20**, 89-104.

Nason, G. P. and von Sachs, R. (1999). Wavelet analysis in time series analysis. *Philosophical Transactions of Royal Society of London*, A **357**, 2511-2526

Ogden, T. (1997). *Essential Wavelets for Statistical Applications and Data Analysis*. Birkhauser, Boston

Paul, R. K., Prajneshu, and Ghosh, H. (2011). Wavelet methodology for estimation of trend in Indian monsoon rainfall time-series data. *Indian Journal of Agricultural Science*, **81** (3), 96-98.

Paul, R. K., Prajneshu, and Ghosh, H. (2013). Wavelet Frequency Domain Approach for Modelling and Forecasting of Indian Monsoon Rainfall Time-Series Data. Journal of the Indian Society of Agricultural Statistics, 67 (3), 319-327

Paul, R. K. **(**2015). ARIMAX-GARCH-WAVELET Model for forecasting volatile data. *Model Assisted Statistics and Application*, **10**(3), 243-252

Paul, R.K. and Birthal, P.S. (2015). Investigating rainfall trend over India using wavelet technique. *Journal of Water and Climate Change*, **7(2)**, 365-378

Anjoy, P. and Paul, R.K. (2017). Wavelet based hybrid approach for forecasting volatile potato price. *Journal of the Indian Society of Agricultural Statistics*, **71**(1), 7–14

Anjoy, P., Paul, R. K., Sinha, K., Paul, A. K. and Ray, M. (2017) A hybrid wavelet based neural networks model for predicting monthly WPI of pulses in India. *Indian Journal of Agricultural Sciences*, **87** (6): 834–839

Rathod, S., Singh, K.N., Paul, R.K., Meher, S.K., Mishra, G.C., Gurung, B., Ray, M. and Sinha, K. (2017). An Improved ARFIMA Model using Maximum Overlap Discrete Wavelet Transform (MODWT) and ANN for Forecasting Agricultural Commodity Price. *Journal of the Indian Society of Agricultural Statistics*, 71(2), 103–111

Anjoy, P. and Paul, R.K. (2019). Comparative performance of wavelet-based neural network approaches. Neural Computing and Applications, 31:3443-3453

Paul, R.K., Sarkar, S., Mitra, D., Panwar, S., Paul, A.K. and Bhar, L.M. (2020) Wavelets based estimation of trend in sub-divisional rainfall in India. *Mausam*, **71** (1), 551-560

Paul, R.K., Paul, A.K. and Bhar, L. M. (2020). Wavelet-based combination approach for modeling sub-divisional rainfall in India. *Theoretical and Applied Climatology*, **139**, (3–4), 949–963

Paul, RK, Sarkar, S and Yadav, SK. (2021). Wavelet based long memory model for modelling wheat price in India. *Indian Journal of Agricultural Sciences*, **91(2)**: 227–31

Singla, S., Paul, RK, and Shekhar, S. (2021). Modelling price volatility in onion using wavelet based hybrid models. *Indian Journal of Economics and Development*, **17(02)**: 256:265

Paul, R.K. and Garai, S. (2021). Performance comparison of wavelets-based machine learning technique for forecasting agricultural commodity prices, *Soft Computing*, **25(20),** 12857-12873

Paul, R.K. and Mitra, D. (2021). Forecasting Wheat Yield using Wavelet-Based Multiresolution Analysis. *Journal of the Indian Society of Agricultural Statistics* **75**(3): 181–186

Paul, R.K. and Garai, S. (2022). Wavelets Based Artificial Neural Network Technique for Forecasting Agricultural Prices. *Journal of the Indian Society for Probability and Statistics* **23**: 47–61

Paul, R.K., Vennila, S., Yeasin, M., Yadav, S.K., Nisar, S., Paul, A.K., Gupta, A., Malathi, S., Jyosthna, M.K., Kavitha, Z., Mathukumalli, S.R., and Prabhakar, M. (2022). Wavelet Decomposition and Machine Learning Technique for Predicting Occurrence of Spiders in Pigeon Pea. *Agronomy*, **12**, 1429

Sarkar, S., Paul, R.K., Paul, A.K. and Bhar, L.M. (2019). Wavelet based Multi-scale Auto-Regressive (MAR) Model: An Application for Prediction of Coconut Price in Kerala. *Journal of The Indian Society of Agricultural Statistics* **73** (1), 1-10

Percival, D. B. and Walden, A. T. (2000). *Wavelet methods for time series analysis*. Cambridge Univ. Press, U.K.

Rajeevan, M., Pai, D. S., Dikshit, S. K. and Kelkar, R. R. (2004): IMD’s new operational models for long – range forecast of southwest monsoon rainfall over India and their verification for 2003. *Curr. Sci.,* **86**, 422 - 31.

Sunilkumar, G. and Prajneshu (2004). Modelling and forecasting meteorological subdivisions rainfall data using wavelet thresholding approach. *Cal. Stat. Assn. Bull.*, **54**, 255-68.

Vidakovic, B. (1999). *Statistical Modeling by Wavelets.* John Wiley, New York

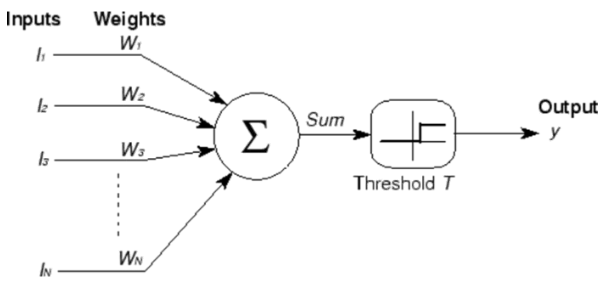
Introduction to Machine Learning

D. Arun Kumar[1], Santosha Rathod[2], Nobin Chandra Paul[2], Ponnaganti Navyasree[2], K. Ravi Kumar[2] and Prabhat Kumar[2]

[1]KSRM College of Engineering, Kadapa, A.P.
[2]ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115
Email: arunkumar.mtech09@gmail.com

Artificial Intelligence (AI) is the field of science that creates machines or devices that can mimic intelligent behaviors of human being. The term AI is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience. On the other hand, machine learning is a type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed. More formally, Machine learning (ML) is defined as a field of the computer sciences that gives computers the ability to learn without being explicitly programmed (Samuel, 1959). Arthur Samuel (1959) was a computer pioneer who wrote first self-learning program, which played checkers-learned from "experience". Machine learning (ML) is a subset of artificial intelligence (AI) that uses statistical methods to enable machines to improve with experience. This involves combining programming with probability and statistics. Machine learning is broadly classified into categories such as classification and regression. In classification, inputs are divided into two or more classes. Pattern recognition and data mining are integral parts of machine learning techniques. The regression aspect of ML is used to map data to a real-valued prediction variable. Time series modeling falls into the category of ML regression problems.

The MuCulloch and Pitts Model was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) known as linear threshold gate, the MuCulloch and Pitts Model is called as first formal model of machine learning techniques (McCulloch and Pitts, 1943).



It is divided into 2 parts. The first part, g takes an input performs an aggregation and based on the aggregated value the second part, f makes a decision.

Suppose that If someone wants to predict their own decision, whether to watch a random cricket match on TV or NOT. The inputs are all Boolean *i.e.*, {0,1} and my output variable is also Boolean {0: Will watch it, 1: Won't watch it}, the following possibilities are prevailed;

So, the inputs could be;

x_1 could be is IPL On (I like IPL more)

x_2 could be is It a Practice Match (I care less about Practice Match)

x_3 could be is MI Playing (I am a big fan of MI and so on.) .... and so on…

g(x) is just doing a sum of the inputs — a simple aggregation. And theta here is called threshold parameter, for example, if I always watch the game when the sum turns out to be 2 or more, the theta is 2 here. This is called Threshold logic.

$$g(x1, x2, x3, \ldots \ldots . xn) = g(x) = \sum_{i=1}^{n} xi$$

$$y = f\big(g(x)\big) = 1 \quad if \;\; g(x) \geq 0 \qquad = 0 \;\; if \;\; g(x) < 0$$

Frank Rosenblatt (1958) introduced a network composed of the units that were enhanced version of McCulloch-Pitts Threshold Logic Unit (TLU) model by adding an extra input that represents bias and termed it as perceptron model.

$$sum = \sum_{i=1}^{n} Xi\, Wi \; + b$$

After, McCulloch-Pitts Threshold Logic Unit (TLU) model the neural network concepts become researchable

issue and evolved as most promising and robust AI/ML techniques utilized in almost all areas.

On the other hand, the time series refers to an important statistical technique for studying the trends and characteristics of collecting data points indexed in chronological order. An ordered sequence of values of a variable at equally spaced time intervals are called as time series (TS) and analysis of such data are termed as time series analysis (TSA). The main aim of time series modeling is to carefully collect and rigorously study the past observations of a time series to develop an appropriate model which describes the inherent structure of the series.

Once a model is constructed, it can be employed to generate future values of the series, i.e., for forecasting. Time series forecasting refers to the process of predicting future observations based on past patterns. Owing to its critical importance across diverse practical domains—such as business, finance, economics, science, and engineering—forecasting of time series data has become a major area of research. One of the defining features of time series is the dependence between successive observations, and this dependence forms the basis for building predictive models.

ARIMA Model

Over time, numerous efforts have been made to enhance forecasting accuracy by developing more robust models. The effectiveness of such models often depends on the length of historical data used in the analysis. According to Box and Jenkins, a minimum of 50 observations is typically required to achieve reliable results in time series modeling. Among classical time series models, the Autoregressive Integrated Moving Average (ARIMA) model is one of the most extensively applied. Its popularity stems from its linear structure, statistical tractability, and the systematic model identification procedure offered by the well-known Box-Jenkins methodology (Box and Jenkins, 1970). For a comprehensive overview of exponential smoothing methods, readers may refer to Makridakis et al. (1998), while Pankratz (1983) provides a wide array of case studies illustrating ARIMA modeling. A detailed treatment of ARIMA and its related concepts is presented in Box et al. (1994). Since many real-world time series are non-stationary, differencing is often introduced to achieve stationarity. The integration of the differencing component into the ARMA model leads to the ARIMA(p,d,q) formulation, where d represents the order of differencing. A time series Yt is said to follow an integrated ARMA process if $\Delta Y_t = (1 - B)^d \varepsilon_t$. The ARIMA model is expressed as follows;

$$\emptyset(B)(1 - B)^d Y_t = \theta(B)\varepsilon_t \qquad (1)$$

Where, $\varepsilon_t \sim WN\ (0, \sigma^2)$ and WN is the white noise. The Box-Jenkins ARIMA model building consists of three steps viz., identification, estimation and diagnostic checking.

Artificial Neural Network (ANN) for Time series

The Artificial Neural Network (ANN) architecture specifically designed for time series analysis is referred to as the Time Delay Neural Network (TDNN). Time series phenomena can be mathematically modeled using neural networks that incorporate an implicit functional representation of time. In contrast to static neural networks such as the multilayer perceptron (MLP), which are inherently non-dynamic, TDNN introduces dynamic properties by including temporal dependencies (Haykin, 1999). One of the simplest and most effective strategies to adapt neural networks for time series forecasting involves the incorporation of time delays, also known as time lags, into the input layer of the network. These lagged inputs allow the model to capture temporal patterns and autocorrelations present in the data. The TDNN represents a class of such feed-forward neural architectures capable of handling time-dependent data structures. The general mathematical formulation for the final output Y_t of a multi-layer feed-forward TDNN is expressed as follows:

$$Y_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \, g\left(\beta_{0j} + \sum_{i=1}^{p} \beta_{ij} Y_{t-p}\right) + \varepsilon_t \qquad (2)$$

where, $\alpha_j (j = 0,1,2, \dots, q)$ and $\beta_{ij} (i = 0,1,2, \dots, p,\ j = 0,1,2, \dots, q)$ are the model parameters, also called as the connection weights, p is the number of input nodes,q is the number of hidden nodes and g is the activation function. The architecture of neural network is represented in figure 1.

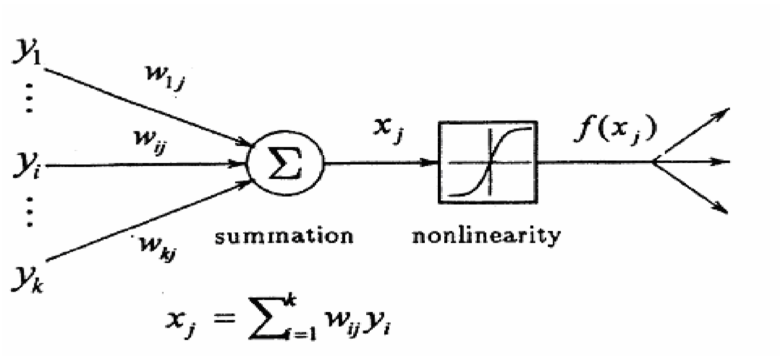


Figure1: Artificial Neural Network Structure

Support Vector Machine (SVM) for Time Series

Support Vector Machine (SVM) is a supervised machine learning technique, originally developed for solving linear classification problems. Later, in 1997, Vapnik extended the concept to handle regression problems by introducing the ε-insensitive loss function (Vapnik, 1997). This extension led to the development of Support Vector Regression (SVR), and when applied to nonlinear regression estimation problems, it is referred to as the Nonlinear Support Vector Regression (NLSVR) model.

The core idea behind NLSVR is to transform the original input time series data into a high-dimensional feature space, where a regression model is constructed. This transformation enables the model to capture nonlinear relationships that may not be apparent in the original input space. Let us consider a dataset represented $Z = \{x_i \, y_i\}_{i=1}^{N}$ where $x_i \in R^n$is the input vector, y_i is the corresponding scalar output, and NNN is the size of the dataset. The general form of the Nonlinear Support Vector Regression estimation function is given as follows:

$$f(x) = W^T \phi\,(x) + b \qquad (3)$$

where $\phi(.)$: $Rn \rightarrow R^{nh}$ is a nonlinear mapping function which map the original input space into a higher dimensional feature space vector. $W \in R^{nh}$ is weight vector, b is bias term and superscript T denotes the transpose.

Brock-Dechert-Scheinkman (BDS) test for testing nonlinearity

BDS (Brock *et al.* 1996), test utilizes the concept of spatial correlation from chaos theory. The computational procedure is given as follows

i) Let the considered time series is

$$\{x_i\} = [x_1, x_2, x_3, ..., x_N] \qquad (4)$$

The next step is to specify a value of m (embedding dimension), embed the time series into m dimensional vectors, by taking each m successive points in the series. This transforms the series of scalars into a series of vectors with overlapping entries

$$\begin{aligned} x_1^m &= (x_1, x_2, ..., x_m) \\ x_2^m &= (x_2, x_3, ..., x_{m+1}) \\ &. \\ &. \\ &. \\ x_{N-m}^m &= (x_{N-m}, x_{N-m+1}, ..., x_N) \end{aligned} \qquad (5)$$

ii) In the third step correlation integral is computed, which measures the spatial correlation among the points, by adding the number of pairs of points (*i, j*), where 1≤ *i* ≤N and 1≤ *j*≤N , in the m-dimensional space which are "close" in the sense that the points are within a radius or tolerance ε of each other.

$$C_{\varepsilon,m} = \frac{1}{N_m(N_m - 1)} \sum_{i \neq j} I_{i,j;\varepsilon} \qquad (6)$$

Where $I_{i,j;\varepsilon}$= 1 if $\left\| x_i^m - x_j^m \right\| \leq \varepsilon$

= 0 otherwise

iii) If the time series is *i.i.d.* then $C_{\varepsilon,m} \approx [C_{\varepsilon,1}]^m$

iv) The BDS test statistics is as follows

$$BDS_{\varepsilon,m} = \frac{\sqrt{N}[C_{\varepsilon,m} - (C_{\varepsilon,1})^m]}{\sqrt{V_{\varepsilon,m}}} \tag{7}$$

Where,

$$V_{\varepsilon,m} = 4[K^m + 2\sum_{j=1}^{m-1} K^{m-j} C_{\varepsilon}^{2j} + (m-1)^2 C_{\varepsilon}^{2m} - m^2 K C_{\varepsilon}^{2m-2}]$$

$$K = K_{\varepsilon} = \frac{6}{N_m(N_m-1)(N_m-2)} \sum_{i<j<N} h_{i,j,N;\varepsilon}$$

$$h_{i,j,N;\varepsilon} = \frac{[I_{i,j;\varepsilon} I_{j,N;\varepsilon} + I_{i,N;\varepsilon} I_{N,j;\varepsilon} + I_{j,i;\varepsilon} I_{i,N;\varepsilon}]}{3}$$

The choice of m and ε depends on number of data. The null hypothesis is data are independently and identically distributed (*i.i.d.*) against the alternative hypothesis the data are not *i.i.d.* this implies that the time series is non-linearly dependent. BDS test is a two-tailed test; the null hypothesis should be rejected if the BDS test statistic is greater than or less than the critical values.

K-Nearest Neighbors (KNN)

The K-Nearest Neighbors (KNN) algorithm is one of the simplest, yet highly effective, supervised machine learning methods used for both classification and regression tasks. It is a non-parametric and instance-based learning algorithm, meaning it makes no explicit assumptions about the underlying data distribution and relies directly on the training data to make predictions.

The basic idea behind KNN is intuitive:

To predict the class (or value) for a new data point, the algorithm searches for the k training samples closest to it in the feature space, where "closest" is usually defined using distance metrics like Euclidean distance, Manhattan distance, or Minkowski distance.

For classification, the algorithm assigns the class label that is most common among these k neighbors — this is called majority voting.

For regression, the algorithm typically predicts the output as the average (or sometimes the weighted average) of the output values of the k nearest neighbors.

Advantages of KNN:

- Simple to understand and easy to implement.
- Naturally handles multi-class problems.
- Works well with non-linear data structures and does not require a training phase.

Limitations of KNN:

- Computationally expensive for large datasets, as distance must be calculated for all training points at prediction time.
- Sensitive to irrelevant features and the choice of distance metric.
- Performance depends strongly on the choice of k (number of neighbors) and feature scaling.

In time regression applications, KNN can be adapted to forecast future values by comparing the current pattern to historical patterns and averaging the outcomes of the closest matches. This approach is often called KNN time series forecasting.

```
# ========================================
# K-Nearest Neighbors (KNN) in R
# Classification & Regression Example
# ========================================
# Install & Load Packages
install.packages("class")   # For KNN classification
install.packages("FNN")     # For KNN regression
library(class)
library(FNN)
# Use the built-in iris dataset
data(iris)
# Split data into training & testing sets
set.seed(123)  # For reproducibility
index <- sample(1:nrow(iris), size = 0.7 * nrow(iris))
train_data <- iris[index, ]
test_data <- iris[-index, ]
# ----------------------------
# KNN Classification
# ----------------------------
# Prepare predictors & labels
train_X_class <- train_data[, 1:4]
train_Y_class <- train_data$Species
test_X_class <- test_data[, 1:4]
test_Y_class <- test_data$Species
# Normalize features
```

```
train_X_class <- scale(train_X_class)
test_X_class <- scale(test_X_class, center = attr(train_X_class, "scaled:center"), scale =
attr(train_X_class, "scaled:scale"))
# Apply KNN classification (k = 5)
k_value <- 5
pred_Y_class <- knn(train = train_X_class, test = test_X_class, cl = train_Y_class, k =
k_value)
# Confusion Matrix & Accuracy
conf_matrix <- table(Predicted = pred_Y_class, Actual = test_Y_class)
print(conf_matrix)
accuracy <- mean(pred_Y_class == test_Y_class)
print(paste("Classification Accuracy:", round(accuracy * 100, 2), "%"))
# ----------------------------
# KNN Regression
# ----------------------------
# Example: Predict Sepal.Length from other features
train_X_reg <- train_data[, 2:4]
train_Y_reg <- train_data$Sepal.Length
test_X_reg <- test_data[, 2:4]
test_Y_reg <- test_data$Sepal.Length
# Normalize features
train_X_reg <- scale(train_X_reg)
test_X_reg <- scale(test_X_reg, center = attr(train_X_reg, "scaled:center"), scale =
attr(train_X_reg, "scaled:scale"))
# Apply KNN regression (k = 5)
knn_reg <- knn.reg(train = train_X_reg, test = test_X_reg, y = train_Y_reg, k = k_value)
# Predicted values & RMSE
pred_Y_reg <- knn_reg$pred
rmse <- sqrt(mean((pred_Y_reg - test_Y_reg)^2))
print(paste("Regression RMSE:", round(rmse, 3)))
```

R code to implement ML TS models

```
nrow(available.packages())
rm(list=ls())
library(forecast)
library(e1071)
library(tseries)
library(ggplot2)
library(tidyverse)
library(fNonlinear)
library(lmtest)
g=read.table(file="rf.txt",header=T)
head(g)
dim(g)
Box.test(g$Rainfall)
rf1=read.table(file="rf1.txt",header=T)
head(rf1)
ggplot(data = rf1, aes(x = Month, y = Rainfall) )+ geom_line(color = "#00AFBB", size = 1) +
 labs(x = "Months", y = "Rainfall") + ggtitle("TS Plot of Monthly Rainfall Data")
```

```
bdsTest(g$Rainfall, m = 3, eps = NULL, title = NULL, description = NULL)
dim(g)
a1=g$Rainfall[1:1416]
a2=g$Rainfall[1417:1428]
Box.test(a1)
acf(a1)
pacf(a1)
############# ARIMA Fitting #########
m1=auto.arima(a1)
coeftest(m1)
accuracy(m1)
Box.test(m1$residuals)
fitted1=m1$fitted
write.csv(as.data.frame(fitted1), file="ARIMA_Fitted.csv")
f1=forecast(m1, h=12)
f11=data.frame(f1)
f12=f11$Point.Forecast
mse11=abs(a2-f12)^2
mse1=mean(mse11)
rmse1=sqrt(mse1)
rmse1
write.csv(as.data.frame(f12), file="ARIMA_Forecasted.csv")
################### ANN ##########
m2=nnetar(a1,6, P=1, 10, repeats=25, xreg=NULL, lambda=NULL, model=NULL,
subset=NULL, scale.inputs=TRUE,  maxit=150)
m2
accuracy(m2)
fitted2=m2$fitted
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
Box.test(m2$residuals)
f2=forecast(m2, h=12)
f21=data.frame(f2)
f22=f21$Point.Forecast
mse21=abs(a2-f22)^2
mse2=mean(mse21)
rmse2=sqrt(mse2)
rmse2
write.csv(as.data.frame(f22), file="ANN_Forecasted.csv")
m3=nnetar(a1)
accuracy(m3)
m3
fitted3=m3$fitted
f3=forecast(m3, h=12)
f31=data.frame(f3)
f32=f31$Point.Forecast
mse31=abs(a2-f32)^2
mse3=mean(mse31)
rmse3=sqrt(mse3)
```

```
rmse3
Box.test(m3$residuals)
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
write.csv(as.data.frame(f32), file="ANN_Forecasted.csv")
#################### SVR ##########
X1=g$Rainfall[1:1416]
Y1=g$Rainfall[2:1417]
X2=g$Rainfall[1416:1427]
Y2=g$Rainfall[1417:1428]
m4=svm(X1,Y1,degree = 3,cost = 45.69, nu=0.5,tolerance = 0.00001,epsilon = 0.00001)
summary(m4)
fitted4 <- fitted(m4)   ## Fitted values
mse41=abs(Y1-fitted4)^2
mse4=mean(mse41)
rmse4=sqrt(mse4)
rmse4
Box.test(m4$residuals)
s3=predict(model,X2)
mse61=abs(Y2-s3)^2
mse6=mean(mse61)
rmse6=sqrt(mse6)
rmse6
############## ARIMA ###########
##########Significance Comparison ##########
########## For testing set ######
dm.test(m1$residuals, m2$residuals)
dm.test(m1$residuals, m3$residuals)
dm.test(m1$residuals, m4$residuals)
######## You have to do it for testing set also #####
########### Hybrid Modeling ##########
r1=m1$residuals
bdsTest(r1, m = 3, eps = NULL, title = NULL, description = NULL)
n1=nnetar(r1)
n1f=n1$fitted
c1=(m1$fitted)+n1f
c11=c1[32:1416]
a11=a1[32:1416]
mse51=abs(a11-c11)^2
mse5=mean(mse51)
rmse5=sqrt(mse5)
rmse5
############# Comparison##########
accuracy(m1)
accuracy(m2)
rmse4
rmse5
#################### Fitted Plots ##########
rm(list=ls())
```

```
library(tidyverse)
library(readxl)
library(ggplot2)
Data1<-as.data.frame(read_excel("Fitted_Plot.xlsx", col_names = TRUE,sheet = "data"))
head(Data1)
Date <- seq(as.Date("2020/1/06"), as.Date("2020/06/30"), "day")
head(Data1)
RF=Data1$RF
Actual=Data1$Actual
Model1=Data1$Model1
Model2=Data1$Model2
Model3=Data1$Model3
Data2=data.frame(Date, RF, Actual, Model1, Model2, Model3)
df <- Data2 %>%
  select(Date, Actual, Model1, Model2, Model3) %>%
  gather(key = "Models", value = "RF", -Date)
tail(df)
p1<-ggplot(df, aes(x = Date, y = RF)) +
  geom_line(aes(color = Models), size = 1) + scale_x_date(date_labels = "%d/%b-%Y")+
labs(x = "Date", y = "RF")+ ggtitle("Actual v/s Fitted plot RF")+
  theme(plot.title = element_text(size = 11))
p1+geom_vline(xintercept = as.Date("2020-06-24"), color="blue4")
```

Apart from Support Vector Regression (SVR), Artificial Neural Networks (ANN), and K-Nearest Neighbors (KNN), there are several other widely used machine learning algorithms that are covered in separate chapters of this manual. These include Decision Trees, Random Forests, Gradient Boosting Machines, Naive Bayes Classifiers, Principal Component Analysis (PCA), and various Ensemble Learning methods. Each of these techniques provides unique approaches to classification, regression, and forecasting problems in agricultural and allied sciences. For deeper understanding, readers are encouraged to consult additional references such as *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman (2009); *Pattern Recognition and Machine Learning* by Bishop (2006); *An Introduction to Statistical Learning* by James et al. (2013); and relevant R documentation and vignettes available online.

References

Box, G. E. P., & Jenkins, G. (1970). *Time series analysis, Forecasting and control*. Holden-Day.

Box, G. E. P., Jenkins, G. M., & Reinsel, G. C. (1994). *Time series analysis: Forecasting and control* (3rd ed.). Prentice Hall.

Brock, W. A., Dechert, W. D., Scheinkman, J. A., & LeBaron, B. (1996). A test for independence based on the correlation dimension. *Econometric Reviews*, *15*, 197-235.

Chitikela, G., Admala, M., Ramalingareddy, V. K., Bandumula, N., Ondrasek, G., Sundaram, R. M., & Rathod, S. (2021). Artificial-Intelligence-Based Time-Series Intervention Models to Assess the Impact of the COVID-19 Pandemic on Tomato Supply and

Prices in Hyderabad, India. *Agronomy, 11*, 1878. https://doi.org/10.3390/agronomy11091878

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation* (2nd ed.). Prentice Hall.

Jha, G. K., & Sinha, K. (2012). Time-delay neural networks for time series prediction: An application to the monthly wholesale
price of oilseeds in India. *Neural Computing and Applications, 24*(3), 563-571.

Makridakis, S., Wheelwright, S. C., & Hyndman, R. J. (1998). *Forecasting: Methods and applications* (3rd ed.). John Wiley & Sons.

Naveena, K., Rathod, S., Shukla, G., & Yogish, K. J. (2014). Forecasting of coconut production in India: A suitable time series model. *International Journal of Agricultural Engineering, 7*(1), 190-193.

Naveena, K., Singh, S., Rathod, S., & Singh, A. (2017). Hybrid ARIMA-ANN modelling for forecasting the price of Robusta coffee in India. *International Journal of Current Microbiology and Applied Sciences, 6*(7), 1721-1726.

Naveena, K., Singh, S., Rathod, S., & Singh, A. (2017). Hybrid Time Series Modelling for Forecasting the Price of Washed Coffee (Arabica Plantation Coffee) in India. *International Journal of Agriculture Sciences, 9*(10), 4004-4007.

Pankratz, A. (1983). *Forecasting with univariate Box-Jenkins models: Concepts and cases*. John Wiley & Sons.

Rathod, S., & Mishra, G. C. (2018). Statistical Models for Forecasting Mango and Banana Yield of Karnataka, India. *Journal of Agricultural Science and Technology, 20*(4), July 2018.

Rathod, S., Saha, A., Patil, R., Ondrasek, G., Gireesh, C., Anantha, M. S., Rao, D. V. K. N., Bandumula, N., Senguttuvel, P., Swarnaraj, A. K., Meera, S. N., Waris, A., Jeyakumar, P., Parmar, B., Muthuraman, P., & Sundaram, R. M. (2021). Two-Stage Spatiotemporal Time Series Modelling Approach for Rice Yield Prediction & Advanced Agroecosystem Management. *Agronomy, 11*, 2502. https://doi.org/10.3390/agronomy11122502

Saha, A., Singh, K. N., Ray, M., & Rathod, S. (2020). A hybrid spatio-temporal modelling: An application to space-time rainfall forecasting. *Theoretical and Applied Climatology, 142*, 1271-1282.

Vapnik, V., Golowich, S., & Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in Neural Information Processing Systems, 9*, 281-287. MIT Press.

Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing, 50*, 159-175.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.

Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning: With applications in R. Springer.

Artificial Neural Network

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, and Prabhat Kumar

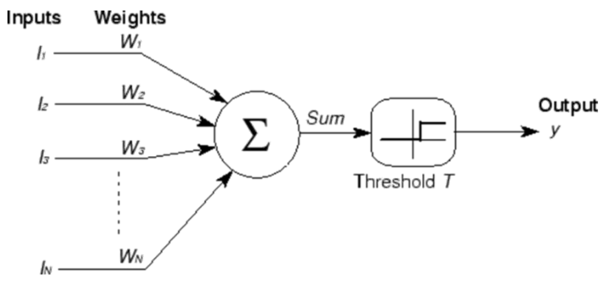
ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115

santosha.rathod@icar.org.in

Introduction

The artificial intelligence (AI) is the field of science that creates machines or devices that can mimic intelligent behaviors of human being. The term AI is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize, or learn from past experience. On other hand, machine learning is a type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed. More formally, Machine learning (ML) is defined as a field of the computer sciences that gives computers the ability to learn without being explicitly programmed (Samuel, 1959). Arthur Samuel (1959) was a computer pioneer who wrote first self-learning program, which played checkers-learned from "experience". ML is a subset of AI technique which use statistical methods to enable machines to improve with experience. This mean that combine: Programing+Probability and Statistics. Machine learning is broadly classified into two or more classes, namely, classification and regression; in classification the inputs are divided into two or more classes. The pattern recognition and data mining are part of the machine learning techniques. The Regression part of ML used to map a data to a real valued prediction variable. The time series modeling falls into the category of ML regression problem.

The MuCulloch and Pitts Model was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943 known as linear threshold gate, the MuCulloch and Pitts Model is called as first formal model of machine learning techniques.



It is divided into 2 parts. The first part, g takes an input performs an aggregation and based on the aggregated value the second part, f makes a decision.

Suppose that If someone wants to predict their own decision, whether to watch a random cricket match on TV or NOT. The inputs are all Boolean i.e., {0,1} and my output variable is also Boolean {0: Will watch it, 1: Won't watch it}, the following possibilities are prevailed;

So, the inputs could be;

x_1 could be is IPL On (I like IPL more)

x_2 could be is It a Practice Match (I care less about Practice Match)

x_3 could be is MI Playing (I am a big fan of MI and so on.) …………… and so on…

g(x) is just doing a sum of the inputs — a simple aggregation. And theta here is called threshold parameter, for example, if I always watch the game when the sum turns out to be 2 or more, the theta is 2 here. This is called Threshold logic.

$$g(x1, x2, x3, \ldots \ldots .xn) = g(x) = \sum_{i=1}^{n} xi$$

$$y = f\big(g(x)\big) = 1 \quad if \;\; g(x) \geq 0$$

$$= 0 \;\; if \; g(x) < 0$$

In late 1950s, Frank Rosenblatt introduced a network composed of the units that were enhanced version of McCulloch-Pitts Threshold Logic Unit (TLU) model by adding an extra input that represents bias and termed it as perceptron model.

$$\boldsymbol{sum = \sum_{i=1}^{n} Xi\, Wi \; + b}$$

After, McCulloch-Pitts Threshold Logic Unit (TLU) model the neural network concepts become researchable issue and evolved as most promising and robust AI/ML techniques utilized in almost all areas.

On other hand, the time series refers to an important statistical technique for studying the trends and characteristics of collecting data points indexed in chronological order. An ordered sequence of values of a variable at equally spaced time intervals are called as time series (TS) and analysis of such data are termed as time series analysis (TSA). The main aim of time series modeling is to carefully collect and rigorously study the past observations of a time series to develop an appropriate model which describes the inherent structure of the series. This model is then used to generate future values for the series, i.e. to make forecasts. Time series forecasting thus can be termed as the act of predicting the future by understanding the past. Due to the indispensable importance of time series forecasting in numerous practical fields such as business, economics, finance, science and engineering, etc. Important properties of time series data are the successive observations under considerations are dependent. Much efforts have been made by researchers over many years to develop the efficient forecasting models to improve the prediction accuracy of the models involving time series data. The accuracy of forecasting models is depending on number of observations used in time series analysis, it is generally believed that at least 50 observations are necessary to perform TSA as stated by Box and Jenkins who were pioneers in time series modeling.

One of the most important and widely used classical time series model is the Autoregressive Integrated Moving Average (ARIMA) model. The popularity of the ARIMA model is due to its linear statistical properties as well as the popular Box-Jenkins methodology (Box and Jenkins 1970) for model building procedure. A good account on exponential smoothing methods is given in Makridakis et al. (1998). A practical treatment on ARIMA modeling along with several case studies can be found in Pankratz (1983). A reference book on ARIMA and related topics are rigorously explained in Box et al. (1994).

Artificial Neural Network for Time series:

Artificial Neural Networks (ANNs) applied to time series analysis are often referred to as Time Delay Neural Networks (TDNNs). Time series data can be effectively modeled using neural networks that incorporate an implicit representation of time. Unlike static neural networks, such as the multilayer perceptron, time delay networks introduce dynamic behavior into the model (Haykin, 1999). A straightforward method for constructing a neural network for time series forecasting is by incorporating time delays, or time lags, into the input layer. These lags serve as temporal inputs, enabling the network to capture dependencies over time. The Time Delay Neural Network is one such architecture designed for this purpose. The following is the general formulation for the final output Y_t of a multi-layer feedforward time delay neural network.

$$Y_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \, g\left(\beta_{0j} + \sum_{i=1}^{p} \beta_{ij} Y_{t-p}\right) + \varepsilon_t$$

where, $\alpha_j (j = 0,1,2, \dots, q)$ and $\beta_{ij} (i = 0,1,2, \dots, p,\ j = 0,1,2, \dots, q)$ are the model parameters, also called as the connection weights, p is the number of input nodes, q is the number of hidden nodes and g is the activation function. The architecture of neural network is represented in figure 1.

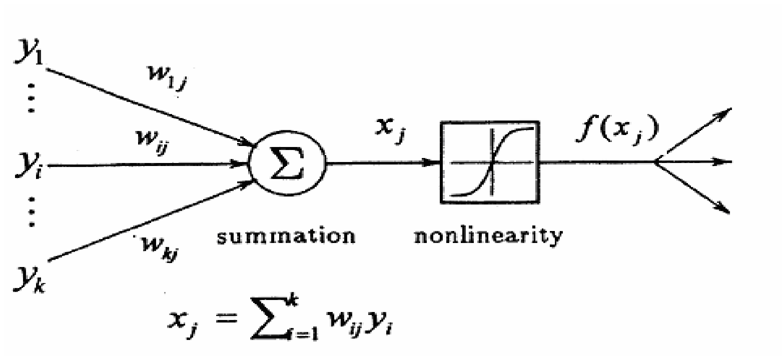


Fig.1: Artificial Neural Network Structure

The Back Propagation Algorithm:

The Multilayer Perceptron (MLP) is trained using supervised learning algorithms, with backpropagation being the most widely used method. This algorithm adjusts the network's weights and thresholds based on the training data to minimize prediction error. The weight of the W_{ij} . These connection weights can be conveniently organized into a weight matrix W, where each element corresponds to a specific connection. This matrix effectively represents the network's connectivity pattern, which defines its overall architecture. In the output layer, each unit determines its activation by following a two-step process. The first step involves computing the total weighted input X_j using the following formula:

$$X_j = \sum y_i W_{ij}$$

Where y_i is the activity level of the jth unit in the previous layer and W_{ij} is the weight of the connection between the ith and the jth unit. Next, the unit calculates the activity y_j using some function of the total weighted input. Generally, we use the sigmoid function:

$$y_i = [1 + e^{-x_j}]^{-1}$$

Once the activities of all outputs units have been determined, the network computes the error E, which is defined by the expression:

$$E = \frac{1}{2}\sum_j (y_i - d_j)^2$$

where y_i is the activity level of the jth unit in the top layer and d_j is the desired output of the jth unit.

The back propagation algorithm consists of four steps:

i. Calculate how fast the error changes as the activity of an output unit is changed. This error derivative (EA) is the difference between the actual and the desired activity.

$$EA_j = \frac{\partial E}{\partial y_i} = y_i - d_j$$

ii. Compute how fast the error changes as the total input received by an output unit is changed. This quantity (EI) is the answer from step (i) multiplied by the rate at which the output of a unit changes as its total input is changed.

$$E_j^I = \frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} \times \frac{\partial E}{\partial x_i} = EA_j y_i (1 - y_j)$$

iii. Compute how fast the error changes as a weight on the connection into output unit is changed. This quantity (EW) is the answer from, step (ii) multiplied by the activity level of the unit from which the connection emanates.

$$EW_{ij} = \frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial X_i} \times \frac{\partial X_i}{\partial W_{ij}} = E_j^I y_i$$

iv. Compute how fast the error changes as the activity of a unit in the previous layer is changed. This crucial step allows back propagation to be applied to multilayer networks. When the activity of a unit in the previous layer changes, it affects the activities of all the output units to which it's connected. So to computer the overall effect on the error, we add together all these separate effects on outputs units. But each effect is simple to calculate. It is the answer in step (iii) multiplied by the weight on the connection to that output unit.

$$EA_j = \frac{\partial E}{\partial y_i} \sum_j \frac{\partial E}{\partial x_i} \times \frac{\partial x_i}{\partial y_i} = \sum_j E_j^I \, W_{ij}$$

By using steps (ii) and (iv), we can convert the EAs of one layer of units into EAs for the previous layer. This procedure can be repeated to get the EAs for as many previous layers as desired. Once we know the EA of a unit, we can use steps (ii) and (iii) to compute the EWs on its incoming connections.

Activation functions

The activation function is also known as the transfer function. It determines the relationship between input and outputs of a node and a network. The activation function is responsible for introducing amount of nonlinearity that is valuable for most ANN applications. Roughly speaking, any differentiable function can be an activation function. Following are the commonly used activation functions;

i. The sigmoidal (logistic) function

$$f(y) = \frac{1}{1+e^{-y}}$$

ii. The hyperbolic tangent (*tanh*) function

$$f(y) = \frac{e^y - e^{-y}}{e^y + e^{-y}}$$

iii. The sine or cosine function

$$f(x) = \sin(y) \text{ or } f(x) = \cos(y)$$

iv. The linear function

$$f(x) = y$$

It is suggested to use the suitable activation function based on nature of the data.

Training sample and test sample

For building an ANN model, Training and test sample are must require. The training sample is used for ANN model development and test sample is adopted for evaluating the forecasting ability of the model. Sometimes a third one called the validation sample is also utilized to avoid the over fitting problem or to determine the stopping point of the training process. It is usually preferred to use one test set for both validation and testing purposes if the data set is small. In our view, the selection of the training and test sample may affect the performance of ANNs. The main question is to divide the data into the training and test sets. Although there is no definite answer to this problem, several factors such as the problem characteristics, the data type and the size of the available data should be considered while dividing the data set. Most of the time it is in practice that training and test sets are selected based on the rule of 90:10, 80:20 or 70:30.

Illustration:

(Results from Rathod et al 2018)

Yearly data on yield (MT/ha) of mango was collected from data base of National Horticulture Board (NHB) and http://www.indiastat.com. For forecasting yield of mango of Karnataka, data from 1980 to 2014 were considered. Data from 1980-2011 were used for model building and 2012 to 2014 were used to check the forecasting performance of the models. The time series plot of mango yield time series of Karnataka is depicted in fig.4. The ARIMA model has been built for mango yield of Karnataka, India. The original time series was found to be non-stationary, so first differencing was done to make the stationary series time series. The adequate model i.e. ARIMA (011) has been identified based on Autocorrelation and Partial Autocorrelation Function (ACF and PACF) plots. The parameters of ARIMA models are estimated using maximum likelihood methods are given in table 1. Further the model performance in training set and testing data set is given in tables 5 and 6.

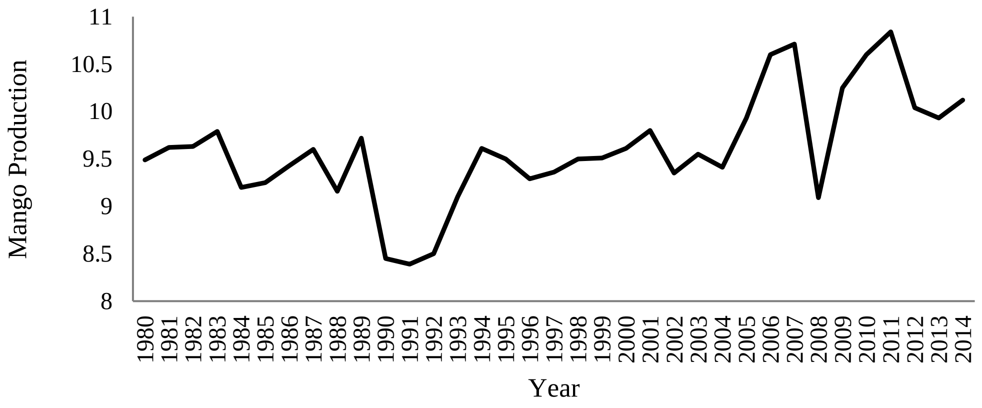


Fig. 4: Time series plot of mango yield time series

Table 1: Parameter estimation of ARIMA (0 1 1) for Mango Yield time series.

Parameter	Estimate	Standard Error	t Value	Approx. Pr > |t|	Lag	P(Resi.) at 6 Lag
Constant	0.033	0.038	0.87	0.382	0	0.240
MA 1	0.581	0.161	3.64	0.003	1	

The ANN was fitted to mango yield time series of Karnataka and the model specifications are given in table 2 and 3. Further the model performance in training set and testing data set is given in tables 5 and 6.

Table 2: ANN Model Specifications.

Time series	**Activation function**		**Time delay**	**No. of hidden nodes**	**Total No. of Parameters**
	hidden Layer	**output layer**			
Mango Yield	Sigmoidal	Linear	2	4	17

Table 4: Comparison of forecasting performance of all models in training data set.

Criteria	**ARIMA**	**ANN**
MAPE	**3.83**	**2.89**

Table 5: Comparison of forecasting performance of all models in testing data set.

Year	**Actual**	**Forecast**	
		ARIMA	**ANN**
2012	10.84	11.75	9.68
2013	10.04	11.15	10.14
2014	9.93	8.67	10.37
MAPE		**10.71**	**5.37**

The ANN model outperformed the ARIMA model in terms of MAPE in both training and testing data sets.

Conclusion:

The main finding of this study is the artificial neural network has performed better than autoregressive moving average model in both training and testing sets. However, this results cannot be treated as generalized as there are no universal approximations are existing in terms of model performance. But for nonlinear time series data machine learning techniques are feasible alternatives which performs better than linear time series models. These models can be further employed in varying autoregressive orders in different real life data sets so that practical validity of the model can be well established.

R codes:

```
nrow(available.packages())
rm(list=ls())
library(forecast)
library(e1071)
library(tseries)
library(ggplot2)
library(tidyverse)
library(fNonlinear)
library(lmtest)
g=read.table(file="rf.txt",header=T)
head(g)
dim(g)
Box.test(g$Rainfall)
rf1=read.table(file="rf1.txt",header=T)
head(rf1)
ggplot(data = rf1, aes(x = Month, y = Rainfall) )+ geom_line(color = "#00AFBB", size = 1) +
 labs(x = "Months", y = "Rainfall") + ggtitle("TS Plot of Monthly Rainfall Data")
bdsTest(g$Rainfall, m = 3, eps = NULL, title = NULL, description = NULL)
dim(g)
a1=g$Rainfall[1:1416]
a2=g$Rainfall[1417:1428]
Box.test(a1)
acf(a1)
pacf(a1)
############# ARIMA Fitting #########
m1=auto.arima(a1)
coeftest(m1)
accuracy(m1)
Box.test(m1$residuals)
fitted1=m1$fitted
write.csv(as.data.frame(fitted1), file="ARIMA_Fitted.csv")
f1=forecast(m1, h=12)
```

```
f11=data.frame(f1)
f12=f11$Point.Forecast
mse11=abs(a2-f12)^2
mse1=mean(mse11)
rmse1=sqrt(mse1)
rmse1
write.csv(as.data.frame(f12), file="ARIMA_Forecasted.csv")

################### ANN ##########
m2=nnetar(a1,6, P=1, 10, repeats=25, xreg=NULL, lambda=NULL, model=NULL,
subset=NULL, scale.inputs=TRUE,  maxit=150)
m2
accuracy(m2)
fitted2=m2$fitted
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
Box.test(m2$residuals)
f2=forecast(m2, h=12)
f21=data.frame(f2)
f22=f21$Point.Forecast

mse21=abs(a2-f22)^2
mse2=mean(mse21)
rmse2=sqrt(mse2)
rmse2
write.csv(as.data.frame(f22), file="ANN_Forecasted.csv")
m3=nnetar(a1)
accuracy(m3)
m3
fitted3=m3$fitted
f3=forecast(m3, h=12)
f31=data.frame(f3)
f32=f31$Point.Forecast
mse31=abs(a2-f32)^2
mse3=mean(mse31)
rmse3=sqrt(mse3)
rmse3
Box.test(m3$residuals)
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
write.csv(as.data.frame(f32), file="ANN_Forecasted.csv")
```

Suggested Readings

Box, G.E.P. and Jenkins, G. (1970). Time series analysis, Forecasting and control, Holden-Day, San Francisco, CA.

Brock, W.A., Dechert, W.D., Scheinkman, J.A, and lebaron, B. (1996). A test for independence based on the correlation dimension, *Econometric reviews*, 15:197-235.

Chitikela, G.; Admala, M.; Ramalingareddy, V.K.; Bandumula, N.; Ondrasek, G.; Sundaram, R.M.; Rathod, S. Artificial-Intelligence-Based Time-Series Intervention Models to Assess the Impact of the COVID-19 Pandemic on Tomato Supply and Prices in Hyderabad, India. Agronomy 2021, 11, 1878.

Jha, G. K. and Sinha, K. (2012) Time-delay neural networks for time series prediction: an application to the monthly wholesale price of oilseeds in India, *Neural Computing and Applications*, 24(3), 563-571

Rathod, S. and Mishra, G.C. (2018). Statistical Models for Forecasting Mango and Banana Yield of Karnataka, India. Journal of Agricultural Science and Technology. 20(4) July 2018.

Rathod, S.; Saha, A.; Patil, R.; Ondrasek, G.; Gireesh, C.; Anantha, M.S.; Rao, D.V.K.N.; Bandumula, N.; Senguttuvel, P.; Swarnaraj, A.K.; Meera, S.N.; Waris, A.; Jeyakumar, P.; Parmar, B.; Muthuraman, P.; Sundaram, R.M. Two-Stage Spatiotemporal Time Series Modelling Approach for Rice Yield Prediction & Advanced Agroecosystem Management. *Agronomy* 2021, *11*, 2502. https://doi.org/10.3390/agronomy11122502

Vapnik, V., Golowich, S., and Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing, In Mozer, M., Jordan, M and Petsche, T. (Eds) Advances in Neural Information Processing Systems, 9:281-287, Cambridge, MA, MIT Press.

Zhang, G.P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50, 159-175.

Support Vector Machine

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar
ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115
Email: santosha.rathod@icar.gov.in

Introduction

Support vector machine (SVM) was originally developed for classification problems by Cortes and Vapnik (1995) for binary classification. A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one "target value" (i.e. the class labels) and several "attributes" (i.e. the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Support vector machine (SVM) is supervised machine learning technique, which was originally developed for linear classification problems. Later in the year 1997, the support vector machine for regression problems were developed by Vapnik by introducing ε-insensitive loss function and it has been extended to the nonlinear regression estimation problems and modeling of such problems is called as Nonlinear Support Vector Regression (NLSVR) model. The basic principle involved in NLSVR is to transform the original input time series into a high dimensional feature space and then build the regression model in a new feature space. The support vector regression, particularly the nonlinear support vector regression has been widely used in time series prediction in many areas viz., agriculture, industry, stock market price prediction *etc.*, (Hong et al. 2006, Cong et al. 2016, Kumar and Prajneshu, 2015).

Support Vector Regression:

Consider a vector of data set $Z = \{x_i\, y_i\}_{i=1}^{N}$ where $x_i \in R^n$ which contains both vector of input and $x_i \in R$ is the scalar output and N is the size of data set. The general expression of NLSVR estimation function is expressed as follows

$$f(x) = W^T \phi\,(x) + b \qquad (1)$$

where $\phi(.)$: $Rn \rightarrow R^{nh}$ is a nonlinear mapping function from original input space into a higher dimensional feature space, which can be infinite dimensional, $w \in R^{nh}$ is weight vector, b is bias term and superscript T denotes the transpose. The coefficients W and b are estimated from data by minimizing the following regularized risk function:

$$R(\theta) = \frac{1}{2}\|w\|^2 + C\left[\frac{1}{N}\sum_{i=1}^{N} L_\varepsilon\left(y_i, f(x_i)\right)\right] \tag{2}$$

The equation (2) contains two components, one is regularized term i.e. $\frac{1}{2}\|w\|^2$ and another term is $\frac{1}{N}\sum_{i=1}^{N} L_\varepsilon\left(y_i, f(x_i)\right)$ called as empirical error term, which is estimated by using Vapnik ε-insensitive loss function which is function given by

$$L_\varepsilon\left(y_i, f(x_i)\right) = f(x) = \begin{cases} |y_i, f(x_i) - \varepsilon|; & |y_i - f(x_i)| \geq \varepsilon, \\ 0 & |y_i - f(x_i)| < \varepsilon, \end{cases} \tag{3}$$

where y_i is actual value and $f(x_i)$ is estimated value. In Equation (14), C is denoted as regularized constant which determines the trade-off between empirical error and regularized parameter. Both C and ε are user-determined hyper-parameters. The final form of Nonlinear SVR function is:

$$f(x) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)K\left(x_i, x_j\right) + b, i = 1,2, \dots, N \tag{4}$$

where α_i and α_i^* are called Lagrange multipliers.

Selection of optimal hyper-parameters is a key step in NLSVR modelling. The performance of NLSVR model is strongly depends on the kernel function (Table 1) and set of hyper-parameters. The value ε is called as tube size equivalent to approximation accuracy in training data (Fig.1). Both C and ε are user determined hyper-parameters. The training points within the ε-tube have no loss and do not provide any information for decision. Only those data points located on or outside the ε-tube are penalized and will serve as the support vectors. This property of sparseness algorithm results only from the ε-insensitive loss function and greatly simplifies computation of Nonlinear SVR. Two positive slack variables ξ_i and ξ_i^* (in interval) are introduced for representing the distance from actual values to corresponding boundary values of the ε-tube. These equal zeros when data points fall within the tube. These slack variables are used for determining the number of support vectors.

The most commonly used kernel function is radial basis function (RBF) which is given as follows.

$$k(x_i, x_j) = exp\{-\gamma\|x - x_i\|^2) \tag{5}$$

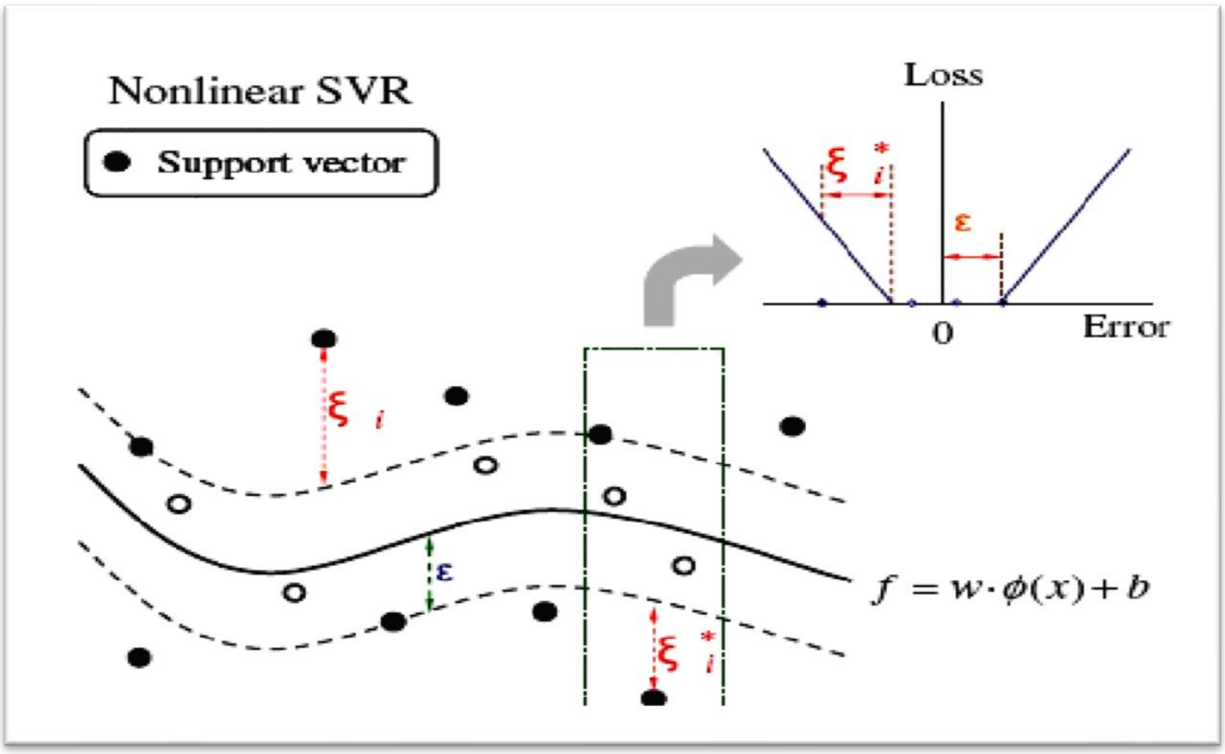


Fig.1: A schematic representation of Vapnikε-insensitive loss function and accuracy tube under Nonlinear SVR model setup

The RBF kernel function in NLSVR requires optimization of two hyper-parameters, i.e. the regularization parameter *C*, which balances the complexity and approximation accuracy of the model and the kernel bandwidth parameter γ, which defines the variance of RBF kernel function (Vapnik 2000).

Table 1: Commonly used Kernel functions in Support Vector Machine problems

Kernel type	Expression
Linear SVM	$K(x, x_i) = x_i^T x$
Polynomial of degree d	$K(x, x_i) = (x_i^T x + k)^d$
Radial Basis Function (RBF)	$K(x, x_i) = \exp\left\{-\frac{\lVert x - x_i \rVert^2}{2\sigma^2}\right\}$ Equivalently
	$K(x, x_i) = \exp\{-\gamma \lVert x - x_i \rVert^2\}$
Multi-Layer Perceptron (MLP)	$K(x, x_i) = \tanh\,(k_1 x_i^T x + k_2$

The support vector machine train the data set based on certain learning rules. Actually, the challenges in learning from data have led to a revolution. In the statistical learning framework, learning means estimating a function

$$y = f(x) \tag{7}$$

Where$x \epsilon R^n$and $y \in \{-1, +1\}$. The estimate must be constructed given only N examples of the mapping performed by the unknown function $(x_1 y_1, x_2 y_2, \ldots, x_N, y_N)$(called the training set). The ultimate goal of learning rule is to minimize the error function or risk function

$$R(\theta\ ) \ = \int L(y, f(x; \theta\ ))dF(x, y) \tag{8}$$

where$\int L(y, f(x\ ))$ is the loss function, a measure of difference between the estimate $f(x\ )$ and the actual value y given by the unknown function at a point x. By defining our goal as minimizing the risk function, we state that our objective is to minimize the expected average loss for a given problem. For this definition to be of value, we need to define learning problems with associated loss functions. In minimizing the risk function, we have to choose the function that provides minimum deviation (in the sense of our loss function) from the true function across the whole function space (for every point$x$). In reality, however, the joint distribution function $F(x, y)$ is unknown, and we do not have value of y for each point x in the function space, but only the training set pairs $\{x_i, y_i\}_{i=1}^N$. We can insert approximate function in risk function by considering the *empirical risk function*:

$$Remp(\theta\ ) = \sum_{i=1}^N L(y_i, f(x_i, \theta\ ) \tag{9}$$

The ERM principle, which minimizes the empirical risk but sometimes it gives larger confidence interval. This induction principle is called Empirical Risk Minimization principle; the popular neural network back propagation algorithm works on this principle. To overcome these difficulties, the structural risk minimization principles has been used to minimize the error function. The principle of Structural Risk Minimization (SRM) is intended to minimize the risk functional with respect to both empirical risk and dimension of the set of functions. Objective of SRM principle is to minimize both the empirical risk and the confidence interval (the two terms in the bound). Thus the SRM principle defines a trade-off between the accuracy and complexity of the approximation by minimizing over both terms.

Illustration:

Annual data on the total oilseed production (in million tonnes) in India for the period 1950–51 to 2015–16 were obtained from the agricultural statistics published by the Reserve Bank of India (RBI), Government of India (RBI Statistics, 2016). The data set covering the years 1950–51 to 2010–11 was utilized for model development, while the observations from 2011–12 to 2015–16 were reserved for model validation purposes. A summary of the descriptive statistics, along with the corresponding time series plot for the data under study, is presented in Table 2 and Figure 2, respectively.

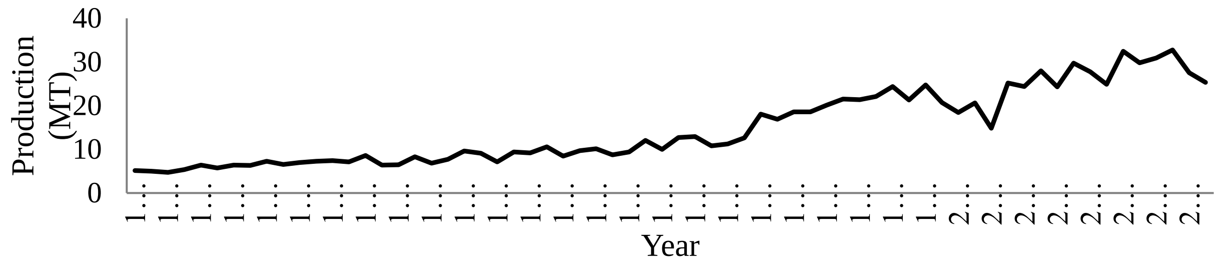


Fig.2: Time series plot of Oilseed production of India

Table 2: Summary statistics of Oilseed Production time series

Statistic	**Oilseed Production**	**Statistic**	**Oilseed Production**
Observation	66	Maximum	32.75
Mean	14.86	Standard Deviation	8.47
Median	11.05	Skewness	0.6
Mode	6.4	Kurtosis	-1.04
Minimum	4.73	Coefficient of Variation (%)	56.98

The Nonlinear Support Vector Regression (NLSVR) model for oilseed production time series was developed using the parameter settings outlined in Table 2. Cross-validation was performed on the time series data, yielding a minimum cross-validation error of 0.035. The model's performance on both the training and testing datasets is presented in Tables 5 and 6, respectively.

Table 3: Model specification of SVR for Oilseed Production time series

Kernel function	No. of SVs	C	γ	ε	K fold cross validation (K)	Cross Validation Error
RBF	7	8.19	3.06	0.15	10	0.035

Univariate ARIMA Model Fitting

The ARIMA model has been built for oilseed production of India. The original time series was found to be non-stationary, so first differencing was done to make the stationary series time series (Figure 3).

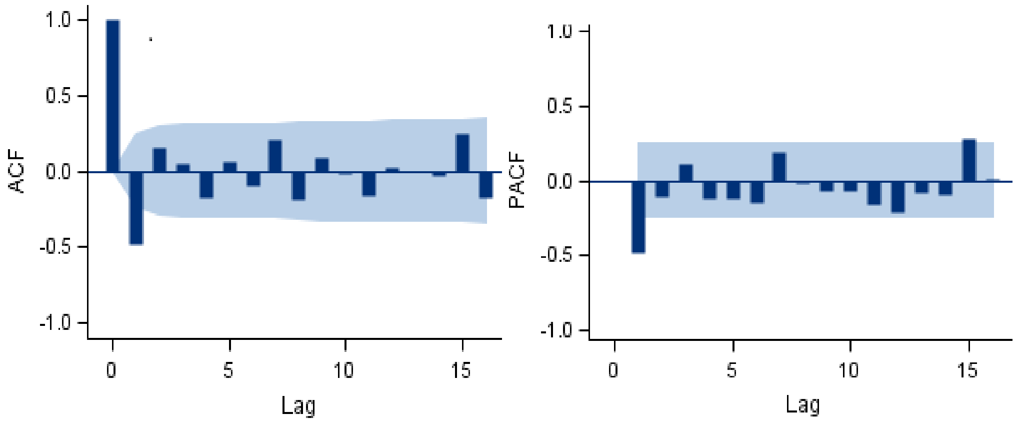


Fig. 3. ACF and PACF time series Oilseed production of India

The appropriate model, ARIMA(1,1,0), was selected based on the analysis of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots (Figure 3). A residual autocorrelation check for the ARIMA model applied to the mango production time series revealed that the residuals are non-autocorrelated, with a chi-square probability value of 0.45. The model's performance on the training and testing datasets is summarized in Tables 5 and 6, respectively.

Table 4: Parameter estimation of ARIMA (1, 1, 0) by Maximum Likelihood Estimation method for Oilseed Production time series

Parameter	Estimate	Standard Error	t Value	Approx. Pr > |t|	Lag
MU	0.43	0.19	1.64	0.1012	0
AR1,1	-0.56	0.18	-4.39	<0.0001	1

Table 5: Model performance of Oilseed Production time series for training data set

Criteria	ARIMA	NLSVR
MSE	5.33	1.39
RMSE	2.32	1.18
MAPE	11.64	6.88

Table 6: Model performance of Oilseed Production time series for testing data set

Year	Actual	Forecast	
		ARIMA	NLSVR
2011	29.80	28.84	30.28
2012	30.94	31.54	31.42
2013	32.75	30.72	33.64
2014	27.51	31.88	30.15
2015	25.30	31.90	26.77
Criteria	MSE	13.31	2.06
	RMSE	3.64	1.43
	MAPE	10.58	4.24

Based on the lowest values of Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) across all models for both the training dataset (Table 3) and the testing (validation) dataset (Table 4), it can be concluded that the Nonlinear Support Vector Regression (NLSVR) technique outperformed the ARIMA model. Despite the high coefficient of variation observed in the dataset (Table 1), the artificial intelligence-based approach, specifically NLSVR, demonstrated superior performance. This may be attributed to the ability of nonlinear machine learning techniques to effectively capture heterogeneous patterns in the data, offering an advantage over the univariate ARIMA model.

Conclusion:

ARIMA models are not always suitable for time series data that exhibit nonlinear structures. In such cases, nonlinear artificial learning techniques like Support Vector Machines (SVM) can provide a more effective means to enhance forecasting performance. Based on the findings of this study, it can be inferred that the use of Nonlinear Support Vector Regression (NLSVR) techniques for modeling and forecasting time series data significantly improves forecasting accuracy. The NLSVR model demonstrated superior performance in forecasting oilseed production in India when compared to other models. This approach may be further extended by incorporating additional machine learning techniques that account for varying autoregressive and moving average orders.

R codes

```
nrow(available.packages())
rm(list=ls())
library(forecast)
library(e1071)
library(tseries)
library(ggplot2)
library(tidyverse)
library(fNonlinear)
library(lmtest)
g=read.table(file="rf.txt",header=T)
head(g)
dim(g)
Box.test(g$Rainfall)
rf1=read.table(file="rf1.txt",header=T)
head(rf1)
ggplot(data = rf1, aes(x = Month, y = Rainfall) )+ geom_line(color = "#00AFBB", size = 1) +
 labs(x = "Months", y = "Rainfall") + ggtitle("TS Plot of Monthly Rainfall Data")
bdsTest(g$Rainfall, m = 3, eps = NULL, title = NULL, description = NULL)
dim(g)
a1=g$Rainfall[1:1416]
a2=g$Rainfall[1417:1428]

X1=g$Rainfall[1:1416]
Y1=g$Rainfall[2:1417]
X2=g$Rainfall[1416:1427]
Y2=g$Rainfall[1417:1428]
m4=svm(X1,Y1,degree = 3,cost = 45.69, nu=0.5,tolerance = 0.00001,epsilon = 0.00001)
summary(m4)
fitted4 <- fitted(m4)   ## Fitted values
mse41=abs(Y1-fitted4)^2
mse4=mean(mse41)
rmse4=sqrt(mse4)
rmse4
```

```
Box.test(m4$residuals)
s3=predict(model,X2)
mse61=abs(Y2-s3)^2
mse6=mean(mse61)
rmse6=sqrt(mse6)
```

Suggested Readings

Cong, Y., Wang J. and Li X. (2016). Traffic Flow Forecasting by a Least Squares Support Vector Machine with a Fruit Fly Optimization Algorithm, *Procedia Engineering,*137: 59-68.

Cortes, C. and Vapnik, V. (1995). Support-vector network. Machine Learning, 20, 1-25.

Hong, W.C. and Pai, P.F. (2006). Predicting engine reliability by support vector machines. *International Journal of Advanced Manufacturing Technology*, 28: 154-161.

Kumar, T.L.M. and Prajneshu. (2015). Development of Hybrid Models for Forecasting Time-Series Data Using Nonlinear SVR Enhanced by PSO. *Journal of Statistical Theory and Practice*, 9(4): 699-711.

Naveena, K., Rathod, S., Shukla, G. and Yogish, K.J. 2014. Forecasting of coconut production in India: A suitable time series model, International Journal of Agricultural Engineering, 7(1):190-193.

Naveena, K., Singh, S., Rathod, S., and Singh, A. 2017. Hybrid ARIMA-ANN Modelling for Forecasting the Price of Robusta Coffee in India. International Journal of Current Microbiology and Applied Sciences, 6(7): 1721-1726.

Naveena, K., Singh, S., Rathod, S., and Singh, A. 2017. Hybrid Time Series Modelling for Forecasting the Price of Washed Coffee (Arabica Plantation Coffee) in India. International Journal of Agriculture Sciences, 9(10): 4004-4007.

Rathod, S. and Mishra, G.C. (2018). Statistical Models for Forecasting Mango and Banana Yield of Karnataka, India. Journal of Agricultural Science and Technology. 20(4) July 2018.

Vapnik, V. (2000). The Nature of Statistical Learning Theory. 2nd Edition, Springer-Verlag, New York.

Vapnik, V., Golowich, S., and Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing, In Mozer, M., Jordan, M and Petsche, T. (Eds) Advances in Neural Information Processing Systems, 9:281-287, Cambridge, MA, MIT Press.

CART (Classification and Regression Tree) and Decision Tree

Ramasubramanian V. and Abin George
ICAR-National Academy of Agricultural Research Management, Hyderabad
Email: ram.vaidhyanathan@gmail.com

1. Introduction

A decision tree is a supervised machine learning model that uses a tree-like structure to make decisions. It is composed of nodes, which represent decision points based on input features, and branches, which represent the outcomes of those decisions. The final leaf nodes represent the predicted outcomes either a class label in classification or a numeric value in regression. Decision trees operate by recursively partitioning the input data based on the values of features. The goal of each split is to reduce impurity i.e., to create subsets that are as homogeneous as possible with respect to the target variable.

Tree-based classification and regression techniques have gained significant popularity in recent years. These decision-tree methods are statistical tools used to explore data and make predictions or classifications of future observations through a set of clearly defined decision rules. Often referred to as rule induction methods, they are valued for their transparent and interpretable structure. The Classification and Regression Tree (CART) methodology has become particularly popular across various disciplines—such as agriculture, medicine, forestry, and natural resource management—as an effective alternative to traditional methods like discriminant analysis, multiple linear regression, and logistic regression.

In CART, observations are recursively partitioned into two subgroups based on predictor variables that show strong association with the response variable. This process yields intuitive and easily interpretable decision rules. CART models can be applied either as classification trees when the response variable is categorical, or as regression trees when it is continuous. One of the major strengths of tree-based approaches is their flexibility; they do not rely on strict assumptions such as normality or linearity. These methods are non-parametric, robust to outliers, capable of handling both continuous and categorical variables, and can efficiently process datasets with large numbers of cases and variables—although they are computationally more intensive.

Unlike traditional techniques such as ordinary least squares (OLS) regression or discriminant analysis, tree-based models do not require the user to specify the underlying functional form or distributional assumptions. Moreover, in contrast to other non-parametric approaches like kernel methods or k-nearest neighbors, tree-based predictors tend to yield relatively simple and interpretable functions of the input variables, making them especially practical. The origins of tree-based methods trace back to the 1960s, with the introduction of AID (Automatic Interaction Detector) by Morgan and Sonquist (1963), initially developed for regression tree analysis. AID works through a stepwise splitting process, beginning with a single group of observations and evaluating each predictor variable by sorting the data and examining all possible n−1 binary splits. The best split is selected by minimizing the within-group sum of squares about the group mean of the dependent variable. Categorical predictors, which lack a natural order, are handled differently: for k categories, $2^{(k-1)-1}$ possible splits are considered, and their effectiveness is also evaluated using the within-cluster sum of squares criterion.This methodology was later extended through the development of THAID (Theta AID) by Morgan and Messenger (1973) to generate classification trees. A key feature emphasized by Morgan and Sonquist is that AID naturally captures interaction effects among predictors. Unlike traditional ANOVA models, where interactions are explicitly specified using cross-product terms, tree-based models capture interactions structurally—manifested as divergent branches from the same node based on different variables. This makes decision tree algorithms like AID highly automatic and well-suited to modeling complex real-world data, where interactions are often inherent and prevalent.

Classification trees operate similarly to regression trees but are used when the dependent variable is categorical. Kass (1980) introduced the CHAID (Chi-squared AID) algorithm as a modification of AID for use with categorized variables. CHAID follows a sequential merge-and-split process based on chi-square statistics. For each predictor:

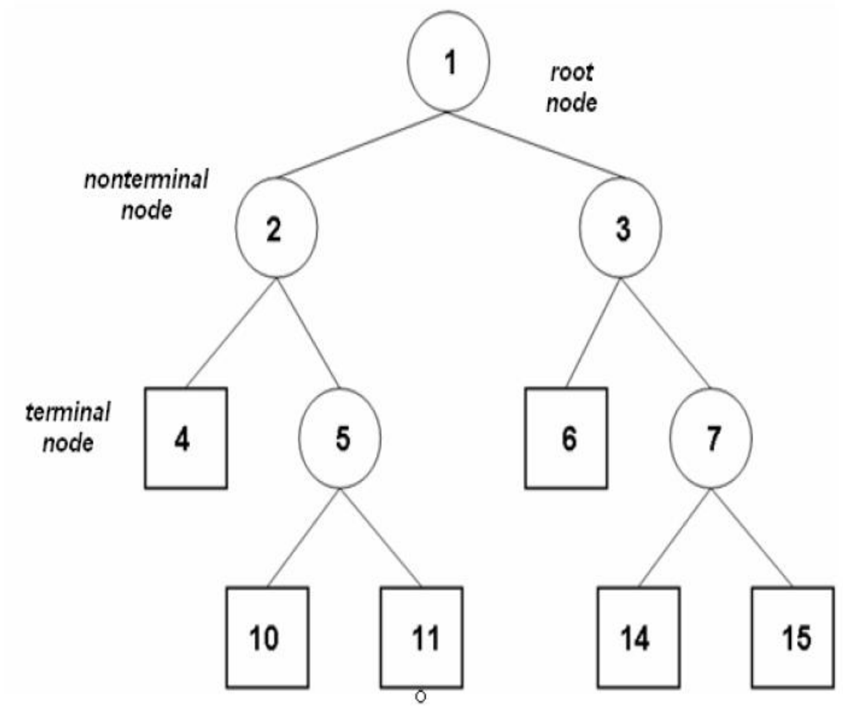
1. A cross-tabulation is created between the predictor categories and the outcome classes.
2. Pairs of categories with the least significant differences (smallest chi-square) are merged.
3. This merging continues until no further insignificant differences remain.
4. The predictor with the largest overall chi-square value is selected for splitting.

Although CHAID is a computationally efficient heuristic, it does not guarantee the best predictive split at each step, unlike exhaustive search methods. CHAID is limited to categorical predictors and cannot accommodate mixed data types. In parallel, within computer science, Quinlan (1986, 1993) developed a family of algorithms such as ID3 and its successors based on information theory. These methods build decision trees by selecting splits that maximize information gain and have been widely adopted in data mining and machine learning applications. Minz and Jain (2003) have employed Rough Set (RS) theory-based decision tree model for classification on the premise that in real life while dealing with sets, due to limited resolution of our perception mechanism, we can distinguish only classes of elements rather than individuals. Elements within classes are indistinguishable. RS offers a simplified search for dominant attributes.

Breiman *et al.* (1984) developed CART (Classification and Regression Trees) which is a sophisticated program for fitting trees to data. Breiman, later in 1994, developed the bagging predictors which is a method of generating multiple versions of a predictor and using them to get an aggregated predictor. A good account of the CART methodology can be found in many recent books, say, Izenman (2008). An application of classification trees in the field of agriculture can be found in Sadhu *et al*. (2014).

2. Broad Outline of CART methodology

The conventional CART methodology is outlined briefly. Following is a schematic representation of a conventional CART tree structure:



In a classification tree, the unique starting point is known as the root node, which contains the entire learning dataset L and appears at the top of the tree structure. Each node in the tree represents a subset of variables and may either be a non-terminal (parent) or a terminal (leaf) node. A parent node undergoes a binary split, dividing it into two child nodes—left and right—based on a decision rule applied to a single predictor variable. If an observation satisfies the condition, it is directed to one child node; otherwise, it proceeds to the other. A node that is not further split becomes a terminal node and is assigned a class label. Every observation from the learning set ultimately falls into one of these terminal nodes, and an unseen observation is classified according to the label of the terminal node it reaches. To build such tree-structured models, the CART (Classification and Regression Trees) algorithm employs recursive binary partitioning, which involves determining the optimal splits of dataset L and its successive subsets. This process includes identifying the variable for the split, formulating the split rule, deciding when to terminate further splitting, and assigning class labels to terminal nodes. While the procedures for assigning labels and generating splits are relatively straightforward, determining the optimal tree size is more complex. Typically, a fully grown tree is first constructed, and then pruning is applied to obtain a tree of optimal size. In exhaustive search procedures, the algorithm evaluates all possible binary splits of each subset at every stage, selecting the one that maximizes node purity. This is assessed using an impurity function, which measures the heterogeneity of class labels within a node. Common impurity metrics include the Gini diversity index and entropy. The reduction in impurity resulting from a split is calculated by subtracting the weighted average impurity of the two child nodes from the impurity of the parent node. Weights are assigned based on the proportion of samples in each child node. The split that yields the greatest impurity reduction (or equivalently, the greatest increase in purity) is selected. Tree construction begins at the root node by evaluating the best split across all predictor variables using the impurity reduction criterion. The process is recursively repeated for each child node, considering only the observations contained within them. This layer-by-layer construction is referred to as recursive partitioning. When each parent node splits into exactly two child nodes, the result is a binary tree. If the tree is expanded until no further splits are possible, it is called a saturated tree. Initially, a large, fully expanded tree is grown, often splitting nodes even when minimal impurity reduction is achieved. To avoid overfitting, a sequence of smaller subtrees is then generated via pruning, which removes certain splits to simplify the tree. The challenge lies in determining the right tree complexity. An overly complex tree may overfit the training data and generalize poorly to new data, whereas a tree with too few terminal nodes may fail to capture essential patterns, reducing predictive accuracy. As trees become more complex, initial increases in classification accuracy are often followed by a deterioration in performance due to overfitting. Therefore, evaluating a tree's misclassification rate for future observations is essential. One method for this is the resubstitution estimate, where the tree is used to classify the same data it was trained on, and the proportion of misclassified instances is calculated. However, a more reliable estimate is obtained using an independent test set, which consists of observations from the same population as the learning set, with known true class labels. The test set error rate is calculated as the proportion of misclassified test cases. Generally, one-third of the available data is reserved for the test set, while the remaining two-thirds are used as the learning set. Alternatively, smaller proportions (e.g., one-tenth) can be used in combination with k-fold cross-validation, such as 10-fold cross-validation, to estimate generalization performance more robustly. A widely used approach to generating an optimal tree is minimum cost-complexity pruning, which involves generating a nested sequence of subtrees by systematically removing the weakest links—a process known as weakest-link pruning. In this method, all nodes descending from a selected non-terminal node are pruned, converting that node into a terminal one. The node chosen for pruning is the one whose removal results in the smallest per-node decrease in the resubstitution misclassification rate. If multiple nodes yield the same reduction, the one associated with the largest number of nodes removed is preferred.This process results in a set of candidate subtrees, from which the optimal tree is selected based on its estimated misclassification rate for future observations—either using a validation set or through cross-validation. This final selection ensures a balance between tree complexity and predictive accuracy.

3. CART tree growing procedure

Let Y,X) be a multivariate random variable where X represents a vector of K explanatory variables, which may include both categorical and continuous types, and Y denotes the response variable. The response variable YYY can either be categorical, taking values from a set of classes C(=1,...,j,...,J) ,or continuous, taking values on the real line. When YYY is categorical, the model constructed is a classification tree, whereas for a continuous YYY, the model is a regression tree.

Splitting Strategy

In determining how to divide subsets of L to create two daughter nodes from a parent node, the general rule is to increase the "purity" of each daughter node with respect to the response variable. This means minimizing the number of misclassified cases in each node. For a complete description of splitting rules, it is important to distinguish between continuous and categorical variables.

For a continuous variable, the number of possible splits at a given node is one less than the number of its distinctly observed values. Suppose a categorical variable is defined by MMM distinct categories, $l_1,l_2,...,l_M$. The set of possible splits at that node is the set of all subsets of $l_1,l_2,...,l_M$. Denote by τ_L and τ_R the left and right daughter nodes, respectively, emerging from a parent node τ. In general, there will be 2^{M-1}-1 distinct splits for an MMM-level categorical variable.

Several types of splits can be considered at each step. For a numerical predictor variable x_k, a subset of L can be divided such that one subset contains $x_k \leq s_k$, and the other contains $x_k > s_k$,, where sks_ksk is an observed value of x_k . For a categorical predictor variable x_k with class labels from a finite set D_k , a subset of L can be divided such that one subset contains $x_k \in S_k$, and the other contains $x_k \notin S_k$, where S_k is a nonempty proper subset of D_k .

At each node, the tree-growing algorithm determines the variable on which it is "best" to split. To do this, all possible splits across all variables at that node are evaluated. Each split is enumerated, assessed, and the one that maximizes a chosen criterion is selected.

Node Impurity Function

During recursive partitioning, all allowable splits of a subset of LLL are examined, and the split that results in the highest increase in node purity is selected. This is achieved using an "impurity function," which quantifies the distribution of response variable classes in a node. The function is designed to be maximal when all classes are equally represented (i.e., the node is most impure), and minimal when the node is pure (i.e., all samples belong to the same class).

To identify the best split for each variable, a "goodness of split" criterion is used. Among all splits, the one that results in the greatest reduction in impurity is chosen. Two commonly used impurity functions are the Gini diversity index and the entropy-based information gain.

Let, $\Pi_1, \Pi_2, \ldots, \Pi_K$ be the $K \geq 2$ classes. For node τ, the node impurity function is defined as $i(\tau) = \left(p(1|\tau), \ldots, p(K|\tau)\right)$ where $p(k|t)$ is an estimate of $P(X \in \Pi_k|\tau)$, , the conditional probability that an observation **X** is in Π_k given that it falls into node τ. Under this set up, the Entropy function is given by, $i(\tau) = -\sum_{k=1}^{K} p(k|\tau) log p(k|\tau)$. When there are only two classes, the entropy function reduces to $i(\tau) = -plogp - (1-p)\log(1-p)$, where, $p = p(1|\tau)$. The other impurity function, i.e. the Gini diversity index is defined as, $i(\tau) = \sum_{k=k'} p(k|\tau)p(k'|\tau) = 1 - \sum_k \{p(k|\tau)\}^2$. In the two class case, the Gini index reduces to $i(\tau) = 2p(1-p)$.

Choosing the best split for a variable:

To evaluate the effectiveness of a potential split, the impurity function value is first calculated using the cases in the learning sample corresponding to the parent node. From this, the weighted average of the impurity values of the resulting daughter nodes is subtracted. The weights are proportional to the number of cases in the learning sample assigned to each daughter node. The result of this calculation represents the decrease in overall impurity that would result from the split. In the tree-growing procedure, all permissible ways of splitting a subset of L are considered. Among them, the split that yields the greatest reduction in node impurity or, equivalently, the greatest increase in node purity—is selected. This method ensures that the tree evolves in a manner that increasingly separates the data based on class homogeneity. The splitting procedure is elaborated further in the following sections.

Suppose, at node τ, a split *s* is applied so that a proportion p_L of the observations drops down to the left daughter node τ_L and the remaining proportion p_R drops down to the right daughter node τ_R. For example, suppose there is a dataset in which the response variable Y has two possible values, 0 and 1. Suppose that one of the possible splits of the explanatory variables X_j is $X_j \leq c$ vs. $X_j > c$, where c is some value of X_j. Then the 2×2 table can be prepared as

Split	Class of Y		Row total
	1	0	

$X_j \leq c$	n_{11}	n_{12}	n_{1+}
$X_j > c$	n_{21}	n_{22}	n_{2+}
Column total	n_{+1}	n_{+1}	n_{++}

Consider, first, the parent node τ. If p_L is estimated by n_{+1}/n_{++} and p_R by n_{+2}/n_{++}, and Gini's index is used as the impurity measure, then the estimated impurity function is,

$$i(\tau) = 2\left(\frac{n_{+1}}{n_{++}}\right)\left(1 - \frac{n_{+1}}{n_{++}}\right) = 2\left(\frac{n_{+1}}{n_{++}}\right)\left(\frac{n_{+2}}{n_{++}}\right)$$

Now consider the daughter nodes, τ_L and τ_R. For $X_j \leq c$, p_L is estimated by n_{11}/ n_{1+} and p_R by n_{12}/ n_{1+}, and for $X_j > c$, p_L is estimated by n_{21}/ n_{2+} and p_R by n_{22}/ n_{2+}. Then the following two quantities are computed,

$$i(\tau_L) = 2\left(\frac{n_{11}}{n_{1+}}\right)\left(1 - \frac{n_{11}}{n_{1+}}\right) = 2\left(\frac{n_{11}}{n_{1+}}\right)\left(\frac{n_{22}}{n_{2+}}\right)$$

The goodness of a split *s* at node τ is given by the reduction in impurity gained by splitting the parent node τ into its daughter nodes, τ_R and τ_L, $\Delta i(\tau) = i(\tau) - \{p_L i(\tau_L) + p_R i(\tau_R)\}$. The best split for the single variable X_j is the one that has the largest value of $\Delta \mathrm{i}(\mathrm{S}, \mathrm{T})$ over all $\ni S_j$, the set of all possible distinct splits for X_j.

Recursive partitioning: To construct a decision tree, the process begins at the root node, which consists of the full learning dataset L. Using a predefined goodness-of-split criterion, the algorithm evaluates each predictor variable and identifies the optimal split at the root node as the one that yields the maximum reduction in impurity across all possible single-variable splits. Once the best split is selected and implemented at the root node, the algorithm proceeds to split each resulting daughter node in the same manner. For each daughter node, calculations are performed using only the subset of data corresponding to that node, rather than the entire dataset. This procedure of recursively partitioning the data into increasingly homogeneous subsets continues layer by layer, and is referred to as recursive partitioning.

When every parent node gives rise to exactly two daughter nodes, the resulting structure is known as a binary tree. If the binary tree continues to grow until no further splits are possible—meaning each terminal node contains data that cannot be further partitioned—the tree is said to be saturated. In high-dimensional classification problems, allowing the tree to grow without constraints can result in a structure that becomes overwhelmingly large and difficult to interpret. To prevent such unmanageable growth, it is often useful to impose restrictions on the tree-building process. One such restriction involves defining a node as terminal if it contains fewer than a specified minimum number of observations, denoted n_{min} . If the number of cases in a node τ satisfies $n(\tau) \leq n_{min}$, then the node is not split further. This threshold acts as a control on tree growth, where larger values of n_{min} result in more aggressive limitations.

Another early stopping approach involves halting the splitting process when the largest value of the goodness-of-split criterion at a given node falls below a pre-specified threshold. However, this method is not always effective. There are instances where a split may yield only a minor decrease in impurity at a certain stage, but can lead to significant reductions in impurity in subsequent splits of its descendant nodes. As a result, relying solely on such stopping rules can prevent the algorithm from discovering more meaningful partitions.

A more effective alternative is to allow the tree to grow to a considerable size initially, even if some splits produce only modest reductions in impurity. Once a large tree has been grown, a pruning process can then be applied to simplify the structure. During pruning, previously made splits are removed, resulting in a sequence of smaller and more manageable subtrees. This post-processing step produces trees with fewer nodes, allowing for a balance between model complexity and predictive accuracy. By starting with a large tree and then pruning it down, the model has a better chance of exploring informative partitions in the data before settling on a simpler and more generalizable structure. This grow-then-prune approach is commonly preferred, particularly in high-dimensional settings, as it avoids premature termination of potentially valuable splits and facilitates the construction of more effective classification models.This aspect of “pruning” will be discussed in the later sections. Thereafter, assignment of a class with a terminal node is done by associating a class with each of the terminal node by the rule of majority. Suppose at terminal node τ there are $n(\tau)$ observations, of which $n_k(\tau)$ are from class Π_K, k=1, 2, …, K. Then, the class which corresponds to the largest of the $\{n_k(\tau)\}$ is assigned to τ. This is called the plurality rule i.e. the node τ is assigned to class Π_i if $p(i|\tau) = max_i p(k|\tau)$.

Estimating the misclassification rate and pruning procedure: The crucial aspect of constructing a reliable tree-structured classification model lies in determining the appropriate complexity of the tree. If nodes are continuously split until no two distinct values of X from the learning sample share the same node, the model may overfit the training data, resulting in poor classification performance on future observations. Conversely, a tree with too few terminal nodes may underutilize the available information in the learning sample, thereby compromising its classification accuracy. Typically, during the tree-growing process, predictive accuracy improves as the partition becomes more refined with additional nodes. However, beyond a certain point, increasing complexity leads to deterioration in the model's ability to generalize, as evidenced by a rise in the misclassification rate for unseen data.To compare the predictive performance of different tree-structured models, it is essential to estimate each tree's misclassification rate on future observations, commonly referred to as the generalization error. Another important metric is the resubstitution estimate of the misclassification rate. This is calculated by applying the tree to classify the same cases from the learning sample used in its construction, and then computing the proportion of misclassified cases.

The resubstitution estimate of the misclassification rate R(τ) of an observation at node τ is calculated as follows: $r(\tau) = 1 - max_k p(k|\tau)$ which, for the two class case, reduces to $r(\tau) = 1 - \max(p, 1-p) = \min\,(p, 1-p)$.

However, it does not work well to use the resubstitution estimate of the misclassification rate. Because, if no two members of the learning sample have the same value of **X**, then a tree having a resubstitution misclassification rate of zero can be obtained by continuing to make splits until each case in the learning sample is by itself in a terminal node. This may be due to the condition that the class associated with a terminal node will be that of the learning sample case corresponding to the node, and when the learning sample is then classified using the tree, each case in the learning sample will drop down to the terminal node that it created in the tree-growing process, and will have its class match the predicted class for the node. Thus the resubstitution estimate can be a very poor estimate of the tree's misclassification rate for the future observations, since it can decrease as more nodes are created, even if the selection of splits is just responding to "noise" in the data, and not to the real structure. This phenomenon is similar to R^2 increasing as more terms are added to a multiple regression model, with the possibility of R^2 nearing one if enough terms are added, even though more complex regression models can be much worse predictors than simpler ones involving fewer variables and terms.

Let T be the classification tree and let $\widetilde{T} = \{T_1, T_2, \ldots, \tau_L\}$ denote the set of all terminal nodes of T. The misclassification rate for T can now be estimated by $R(T) = \sum_{T=\tilde{T}} R(T)P(\tau) =$

$\sum_{l=1}^{L} R(T_l)p(\tau_l)$ for T, where P(τ) is the probability that an observation falls into node τ. If P(τ_l) is estimated by the proportion p(τ_l) of all observations that fall into node τ_l, then, the resubstitution estimate of R(T) is $R^{re}(T) = \sum_{l=1}^{L} r(\tau_1)p(\tau_1) = \sum_{l=1}^{L} R^{re}\,(\tau_1)$ where, $R^{re}(T_l) = r(T_l)p(T_l)$.

A more reliable estimate of a decision tree's misclassification rate can be obtained through the use of an independent test set, which consists of observations drawn from the same underlying population or distribution as the training (learning) set. Similar to the learning set, each observation in the test set possesses known values of the predictor variables along with the true class labels. The misclassification rate estimated from the test set is defined as the proportion of test set observations incorrectly classified when their predicted classes are determined using the tree developed from the learning set. Typically, it is recommended that approximately one-third of the total observations be reserved as a test set, while the remaining two-thirds are used for training. However, in some instances, a smaller proportion, such as one-tenth, may also be considered adequate for the test set.

Regardless of the approach adopted to estimate the misclassification rate, the central challenge remains: how to construct the most accurate classification tree, or more specifically, how to generate a set of candidate trees from which the best-performing one can be selected based on its estimated misclassification rate. As noted earlier, implementing a stopping rule to determine the optimal tree size tends to be ineffective. Instead, it is generally advisable to first grow a fully expanded tree and then apply a pruning procedure, wherein certain nodes are systematically removed to obtain simpler subtrees. This pruning process yields a finite sequence of nested subtrees, where each subsequent tree is a proper subtree of the previous one. The classification accuracy of each subtree in this sequence is evaluated using reliable estimates of misclassification rate, derived either from a test sample or through cross-validation techniques. Ultimately, the subtree with the best performance is selected as the final classification model.

Pruning procedure: An effective method for generating a meaningful sequence of trees of varying sizes is the application of minimum cost-complexity pruning. This technique involves creating a nested series of subtrees from the original, fully grown tree through a process known as weakest-link cutting. In this procedure, all descendant nodes stemming from a particular nonterminal node are removed—effectively converting that node into a terminal one. The node selected for pruning is the one whose removal yields the smallest decrease in the resubstitution misclassification rate per pruned node. When multiple pruning options result in identical per-node decreases in the resubstitution error, the preference is given to the option that eliminates the largest number of nodes. This ensures a more substantial simplification of the tree while maintaining comparable predictive performance. In cases of minimal pruning, only two daughter terminal nodes are pruned from a single parent node, turning it into a terminal node. However, in other situations, a more extensive set of descendant nodes may be pruned simultaneously from a deeper internal node within the tree structure.

Instead of using the resubstitution measure $R^{re}(\tau)$ as the estimate of R(T), it is modified for tree pruning. Let $\alpha \geq 0$ be a complexity parameter. For any node $\tau \in T$, the cost-complexity measure $R_\alpha(\tau)$ is given by $R_\alpha(T) = R^{re}(T) + \alpha$. A cost-complexity pruning measure for a tree T is defined as $R_\alpha(T) = \sum_{l=1}^{L} R_\alpha(T_1) = R^{re}\,(T) + \alpha|\widetilde{T}|$, where, $|\tilde{T}| = L$ is the number of terminal nodes in the subtree T, which is a measure of tree complexity, and α is the contribution to the measure for each terminal node. One can think of$\alpha|\tilde{T}|$ as a penalty term for tree size, so that $R_\alpha(T)$ penalizes $R^{re}(T)$ for generating too large a tree. For each α, the subtree T(α) of T_{max} that minimizes $R_\alpha(T)$, is selected. To minimize this measure, for small values of α, trees having a large number of nodes, and a low resubstitution estimate of misclassification rate, will be preferred. Thus, the value of α determines the size of the tree. When α is very small, the penalty term will be small, and so the size of the minimizing subtree T(α), which will essentially be determined by $R^{re}(T(\alpha))$, will be large. For large enough values of α, a one node tree will minimize the measure. For example, suppose α is set to zero, i.e. α=0 and the tree T_{max} is grown so large that each terminal node contains only a single observation; then, each terminal node takes on the class of its solitary observation, every observation is classified correctly, and $R^{re}(T_{max})=0$. So, T_{max} minimizes $R_0(T)$. As the value of α is increased, the minimizing subtree T(α) will have fewer and fewer terminal nodes. When α is very large, it results in a tree having only the root node.

Since the resubstitution estimate of misclassification rate is generally overoptimistic and becomes unrealistically low as more nodes are added to a tree, it is expected that there is some value of α that properly penalizes the overfitting of a tree which is too complex, so that the tree which minimizes $R_\alpha(T)$, for the proper value of α, will be a tree of about the right complexity (to minimize the misclassification rate of the future observations). Even though the proper value of α is unknown, utilization of the weakest-link cutting procedure explained earlier guarantees that for each value of $\alpha(\geq 0)$, a subtree of the original tree that minimizes $R_\alpha(T)$ will be a member of the finite nested sequence of subtrees produced.

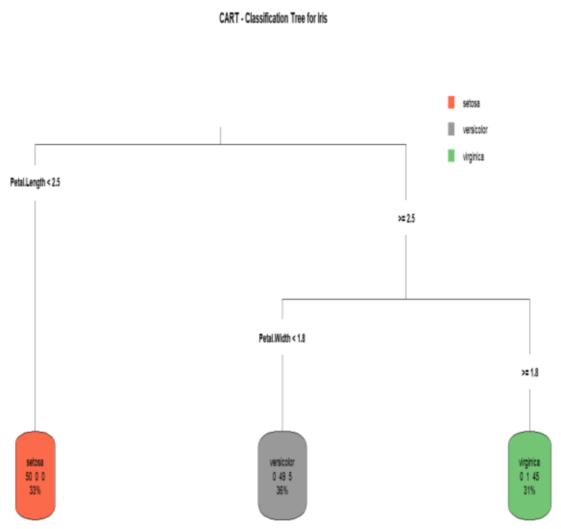
It is worth noting that although α is defined on the interval $[0,\infty)$, the number of subtrees of T is finite. Suppose that, for $\alpha=\alpha_1$, the minimizing subtree is $T_1=T(\alpha_1)$. As the value of α is increased, T_1 continues to be the minimizing subtree until a certain point, say, $\alpha=\alpha_2$, is reached, and a new subtree, $T_2=T(\alpha_2)$, becomes the minimizing subtree. As α is increased further, the subtree T_2 continues to be the minimizing subtree until a value of α is reached, $\alpha=\alpha_3$, say, when a new subtree $T_3=T(\alpha_3)$ becomes the minimizing subtree. This argument is repeated a finite number of times to produce a sequence of minimizing subtrees $T_1,T_2,T_3, \ldots$. The aforesaid discussion states that a finite increasing sequence of complexity parameters,
$0 = \alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots < \alpha_M$ corresponds to a finite sequence of nested subtrees, say, M in number, of the fully grown tree, $T_{max} = T_0 > T_1 > T_2 > \cdots > T_M$.

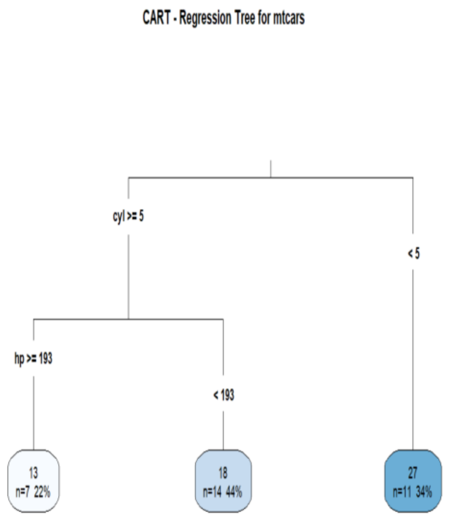
Selecting the right sized tree among the candidate sub-trees: The sequence of subtrees produced by the pruning procedure serves as the set of candidate subtrees for the model, and to obtain the classification tree, all that remains to be done is to select the one which will hopefully have the smallest misclassification rate for future observations. The selection is based on estimated misclassification rates, obtained using a test set or by cross validation. Selection based on test set is discussed subsequently.

If an independent test set is available, it is used to estimate the error rates of the various trees in the nested sequence of subtrees, and the tree with minimum estimated misclassification rate can be selected to be used as the tree-structured classification model. For this purpose, the observations in the learning dataset ($\mathcal{L}$) are randomly assigned to two disjoint datasets, a training dataset ($\mathcal{D}$) and a test set ($\mathcal{T}$), where $\mathcal{D} \cap \mathcal{T} = \Phi$. Suppose there are $n_{\mathcal{T}}$ observations in the test set and that they are drawn independently from the same underlying distributions as the observations in $\mathcal{D}$. Then the tree T_{max} is grown from the learning set only, and it is pruned from bottom up to give the sequence of subtrees $T_0 > T_1 > T_2 > \cdots > T_M$, and a class is assigned to each terminal node. Once a sequence of subtrees has been produced, each of the $n_{\mathcal{T}}$ test-set observations are dropped down the tree T_k. Each observation in $\mathcal{T}$ is then classified into one of the different classes. Because the true class of each observation in $\mathcal{T}$ is known, $R(T_k)$ is estimated by $R^{ts}(T_k)$, which is (4) with $\alpha=0$; i.e., $R^{ts}(T_k) = R^{re}(T_k)$, the resubstitution estimate computed using the independent test set. When the costs of misclassification are identical for each class, $R^{ts}(T_k)$ is the proportion of all test set observations that are misclassified by T_k. These estimates are then used to select the best pruned subtree T_* by the rule, $R^{ts}(T_*) = min_k R^{ts}(T_k)$ and $R^{ts}(T_*)$ is its estimated misclassification rate. A popular alternative is to recognize that since all of the error rates are not accurately known, but only estimated, it could be that a simpler tree with only a slightly higher estimated error rate is really just as good as or better than the tree having the smallest estimated error rate.

4. R code for CART based Class prediction (Classification Tree)

```
# Install and load required packages
install.packages(c("rpart", "rpart.plot", "caret", "e1071"))
library(rpart)
library(rpart.plot)
library(caret)
library(e1071)
# Classification Tree on iris data
# Build CART model
set.seed(123)
iris_tree <- rpart(Species ~ ., data = iris, method = "class")
# Plot tree
rpart.plot(iris_tree, type = 3, extra = 101, fallen.leaves = TRUE, main = "CART - Classification
Tree for Iris")
# Predict on training data
iris_pred <- predict(iris_tree, iris, type = "class")
# Confusion matrix
conf_matrix <- confusionMatrix(iris_pred, iris$Species)
print(conf_matrix)
```





The CART classification tree for the Iris dataset effectively separates the three iris species—setosa, versicolor, and virginica—using two key variables: Petal.Length and Petal.Width. The first split occurs at Petal.Length < 2.5, which perfectly isolates all 50 setosa flowers, demonstrating that this species has distinctly short petals. For flowers with Petal.Length ≥ 2.5, a second split at Petal.Width < 1.8 distinguishes between versicolor and virginica. Flowers with narrower petals (Petal.Width < 1.8) are mostly versicolor (49 correct, 5 misclassified), while those with wider petals (≥ 1.8) are mostly virginica (45 correct, 1 misclassified). This simple, interpretable tree highlights the strong predictive power of petal dimensions in distinguishing iris species and offers an easily explainable classification model with high accuracy.

5. R code for CART based Prediction (Regression Tree)

```
# Install and load required packages
install.packages(c("rpart", "rpart.plot", "caret", "e1071"))
library(rpart)
library(rpart.plot)
library(caret)
library(e1071)
# Regression Tree on mtcars
# Build CART model
set.seed(123)
```

```
car_tree <- rpart(mpg ~ ., data = mtcars, method = "anova")
# Plot tree
rpart.plot(car_tree, type = 3, extra = 101, fallen.leaves = TRUE, main = "CART - Regression Tree for mtcars")
# Predict on training data
car_pred <- predict(car_tree, mtcars)
# Calculate RMSE
rmse <- sqrt(mean((car_pred - mtcars$mpg)^2))
cat("\nRMSE for Regression Tree on mtcars dataset:", round(rmse, 3), "\n")
```

The CART regression tree for the mtcars dataset predicts the car's mileage (mpg) using two key variables: the number of cylinders (cyl) and horsepower (hp). The tree first splits the data based on whether a car has 5 or more cylinders. Cars with fewer than 5 cylinders (typically 4-cylinder vehicles) form a group with the highest average mileage of 27 mpg, indicating better fuel efficiency. Among cars with 5 or more cylinders, those with horsepower under 193 have a moderate average mileage of 18 mpg, while those with 193 or more horsepower have the lowest mileage of 13 mpg, reflecting the inefficiency of powerful engines in larger vehicles. The model clearly demonstrates that fewer cylinders and lower horsepower lead to better fuel economy in cars.

References:

Breiman, L., Freidman, J.H., Olshen, R.A. and Stone, C.J. (1984). *Classification and regression trees*. Wadsworth, Belmont CA.

Izenman, A.J. (2008). *Modern multivariate statistical techniques: Regression, classification and manifold learning*. Springer, New York.

Kass, G. V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data, *Applied Statistics*, **29(2)**, 119-127.

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the Analysis of Survey Data, and a Proposal. *Journal of the American Statistical Association*, **58(302)**, 415-434.

Minz, S. and Jain, R. (2003). Rough set based decision tree model for classification, In: Kambayashi, Y., Mohania, M., Wöß, W. (eds) Data Warehousing and Knowledge

Discovery. DaWaK 2003. Lecture Notes in Computer Science, vol 2737. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-45228-7_18.

Morgan, J.N. and Messenger, R.C. (1973). THAID: a sequential search program for the analysis of nominal scale dependent variables. Institute for Social Research, University of Michigan, Ann Arbor, MI.

Quinlan, J. R. (1986). Induction of Decision Trees, *Machine Learning*, **1**, 81-106. https://link.springer.com/article/10.1007/BF00116251.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers.

Sadhu, S.K., Ramasubramanian, V., Rai, A. and Kumar, A. (2014). Decision tree based models for classification in agricultural ergonomics, *Statistics and Applications*, **12(1&2)**, 21-33.

Extreme Learning Machine (ELM)

Kapil Choudhary
College of Agriculture, Sumerpur (Pali)- 306902, Agriculture University, Jodhpur
Email: kapiliasri@gmail.com

Introduction

Time series forecasting involves the prediction of future values based on historical observations and plays a critical role in various domains such as finance, energy, agriculture, weather prediction, and supply chain management. With increasing uncertainty, globalization, and data availability, the need for accurate and efficient forecasting models has become more significant than ever. Effective time series forecasting assists decision-makers in planning, resource allocation, risk mitigation, and responding dynamically to real-time trends and anomalies. However, time series forecasting is inherently complex due to its nonlinear, dynamic, and often noisy nature, especially when affected by external factors such as market fluctuations, seasonal effects, or environmental variability. Traditional statistical models like ARIMA, exponential smoothing, or Box–Jenkins methodologies often struggle to capture such nonlinear patterns effectively, particularly in the presence of large, high-frequency, or multivariate datasets.

Recently, artificial neural networks (ANNs) have shown great promise in modeling and forecasting time series data owing to their capacity for nonlinear mapping and adaptability. Among them, extreme learning machine (ELM), a relatively novel learning algorithm for single-hidden-layer feedforward neural networks (SLFN), has gained increasing attention due to its fast learning speed and excellent generalization performance. Unlike conventional backpropagation-based ANNs, ELM randomly assigns the input weights and biases and analytically computes the output weights using the Moore–Penrose generalized inverse, eliminating issues like local minima, overfitting, and long training times. Numerous studies have demonstrated that ELM outperforms conventional gradient-based learning algorithms in terms of speed and accuracy in time series applications ranging from energy load prediction to stock market analysis. Its structural simplicity and non-iterative training approach make it highly suitable for real-time forecasting tasks and large-scale data environments. However, due to the randomness in the assignment of input weights and hidden biases, ELM's forecasting performance can vary across runs. To mitigate this issue, ensemble or integrated ELM frameworks have been proposed, where the final forecast is obtained by aggregating the outputs of multiple ELM models, leading to improved robustness and accuracy.

Moreover, preprocessing steps such as normalization are typically applied to scale the data within a defined range, which not only facilitates faster training but also prevents saturation of activation functions. A subsequent denormalization step is essential to restore the predicted values to their original scale.

2. Extreme learning machine (ELM)

ELM is a powerful learning technique for single-hidden-layer feedforward neural networks (SLFNs) known for its fast learning speed and strong generalization capability. ELM is an efficient neural network-based model which is similar in architecture to a feedforward neural network (FFNN). However, unlike FFNN, the learning algorithm of this methodology does not require the tuning of every parameter of its network (weights and biases). In both conventional FFNN and ELM, the hidden nodes are initialized randomly, but the gradient descent and backpropagation algorithm of FFNN keep updating their parameters till the loss function is minimized. However, in the ELM, the random values of hidden layer units stay constant throughout the whole training process. The novelty in the learning algorithm of ELM is that the parameters or weights that connect the hidden layer to the output layer are determined by Moore–Penrose generalized inverse technique, which ultimately makes the algorithm time-efficient (Qu *et al.*, 2016). The time series data, which may include historical records of a variable of interest (e.g., stock prices, energy consumption, crop yield, etc.), is first preprocessed to extract meaningful input–output pairs. Typically, a sliding window approach is employed to transform the univariate or multivariate time series into a supervised learning format. That is, previous lagged values of the series (and possibly exogenous variables) are used as inputs, while the target variable at the next time step is considered the output. After feature construction, the dataset is split into three subsets: training, testing, and forecasting (or validation). Prior to training, both training and testing datasets are normalized to a defined range (e.g., [0, 1] or [−1, 1]) to ensure stable and efficient learning and to prevent neuron saturation. The ELM model is then trained using the training set, wherein the input weights and biases are randomly assigned, and the output weights are computed analytically using the Moore Penrose generalized inverse.

Once the ELM is trained, it is used to forecast future values by directly applying the learned model on the forecasting dataset. After predictions are generated, an unnormalization step is carried out to convert the outputs back to their original scale for evaluation and interpretation.

The advantage of ELM lies in its simplicity, high computational efficiency, and ability to handle nonlinear and nonstationary patterns often present in time series data.

The whole procedure is described as follows:

Suppose a time series converted into supervised learning format as a collection of S samples, $\{\boldsymbol{y}_i, t_i\}; i = 1, \dots, S$ where $\boldsymbol{y_i} = y_{i1}, y_{i2}, \dots, y_{ip}$ represents input patterns and $t_{\boldsymbol{i}}$ is the output for i^{th} sample. If the model is to be made using p input nodes and q hidden nodes, then the final output through a single output node can be represented by

$$t_i = \sum_{m=1}^{q} \partial_m \psi(\boldsymbol{w}_m . \boldsymbol{y}_i + b_m) \tag{1}$$

where, $\boldsymbol{w}_m = [w_{m1}, w_{m2}, \dots, w_{mp}]'$ is the synaptic weight between j^{th} input neuron ; $j = 1, \dots, p$ and m^{th} hidden neuron ; $m = 1, \dots, q$, ∂_m is the output weight between m^{th} hidden neuron and the output neuron, b_m is the bias and $\psi(.)$ is the activation function of the hidden nodes.

As discussed earlier, the Extreme Learning Machine (ELM) differs from traditional single-hidden-layer feedforward neural networks (SLFNs) in its training mechanism. In ELM, the input weights and hidden layer biases are randomly generated and remain fixed throughout the training process. This eliminates the need for iterative tuning or gradient-based optimization, which is often required in conventional neural networks. The only parameters that need to be determined are the output weights, which connect the hidden layer to the output layer. The evaluation of these output weights is equivalent to solving a linear system of the form denoted in matrix notation as $\mathbf{H}\boldsymbol{\partial} = \mathbf{T}$ where

$$\mathbf{H} = \begin{bmatrix} \psi(\boldsymbol{w}_1 . \boldsymbol{y}_1 + b_1) & \cdot & \cdot & \cdot & \psi(\boldsymbol{w}_m . \boldsymbol{y}_1 + b_m) \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \psi(\boldsymbol{w}_1 . \boldsymbol{y}_S + b_1) & \cdot & \cdot & \cdot & \psi(\boldsymbol{w}_m . \boldsymbol{y}_S + b_m) \end{bmatrix}_{\boldsymbol{S \times m}} \tag{2}$$

and called the hidden layer output matrix of the ELM, $\boldsymbol{\partial} = [\partial_1, \partial_2, \dots, \partial_m]'$ and $\mathbf{T} = [t_1, t_2, \dots, t_S]'$. The parameters in $\boldsymbol{\partial}$ is estimated through least squares fitting by solving

$$\underset{\boldsymbol{\partial}}{min}\frac{1}{S}\|\mathbf{T}-\mathbf{H}\boldsymbol{\partial}\|^2$$

This will give the minimum norm least square solution of output weight $\hat{\boldsymbol{\partial}} = \mathbf{H}^{\dagger}\mathbf{T}$, where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of the hidden layer output matrix **H**.

3. General Algorithm for Extreme Learning Machine (ELM)

The general algorithm for training an Extreme Learning Machine (ELM) can be described in the following steps:

Step 1: Randomly assign input weights $\boldsymbol{w}_m = [w_{m1}, w_{m2}, \dots, w_{mp}]'$ and biases b_m for each hidden node.

Step 2: Compute the hidden layer output matrix **H** applying the activation function $\psi()$ to the linear combination of inputs and biases. For each input sample xjx_j, the output of the hidden layer is computed as

$$h_J = [\psi(\boldsymbol{w}_1.\boldsymbol{y}_1 + b_1), \psi(\boldsymbol{w}_2.\boldsymbol{y}_2 + b_2), \dots . \psi(\boldsymbol{w}_m.\boldsymbol{y}_1 + b_m)]$$

Step 3: Calculate the output weights $\hat{\boldsymbol{\partial}} = \mathbf{H}^{\dagger}\mathbf{T}$, by solving the linear system where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse of the hidden layer output matrix **H**.

R package for practical implications

```
 elm
library (nnfor)
fit =elm(AirPassengers)
print(fit)
plot(fit)
frc = forecast(fit,h=36)
plot(frc)
```

Suggested Readings

- Wong, W. K., and Guo, Z. X. (2010). A hybrid intelligent model for medium-term sales forecasting in fashion retail supply chains using extreme learning machine and harmony search algorithm. *International Journal of Production Economics*, *128*(2), 614-624.
- Huang, G. B., Zhu, Q. Y., and Siew, C. K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*(1-3), 489-501.
- Chen, X., Dong, Z. Y., Meng, K., Xu, Y., Wong, K. P., and Ngan, H. W. (2012). Electricity price forecasting with extreme learning machine and bootstrapping. *IEEE transactions on power systems*, *27*(4), 2055-2062.
- Rong, H. J., Ong, Y. S., Tan, A. H., and Zhu, Z. (2008). A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, *72*(1-3), 359-366.

Random Forest Regression

Ramasubramanian V. and Abin George
ICAR-National Academy of Agricultural Research Management, Hyderabad
Email: ram.vaidhyanathan@gmail.com

1. Introduction

The word forest in 'Random Forest' refers a group of decision trees and the word random refers to the way the results of the trained decision trees are combined. Before starting to discuss random forest regression (RFR), the preliminaries of building decision trees are briefly described and the logic for resorting to RFR is explained.

2. Preliminaries and terminologies

Some preliminaries and terminologies related to decision trees are given here. For some supplementary information, the readers are also referred to read the lecture notes on "CART (Classification And Regression Tree) and Decision Tree" which lecture is also a part of the training programme in which this lecture is also there.

Decision trees: Decision trees are intuitive and interpretable models used for both classification and regression tasks. In these models, predictions are made through a series of hierarchical decisions based on comparisons of predictor variables with threshold values. Each decision leads to a branch, ultimately arriving at a leaf node that represents the predicted outcome. Visually, decision trees can be represented as flowcharts, making them easy to understand and explain. Geometrically, they work by partitioning the predictor (feature) space into a set of distinct, non-overlapping regions. Within each region, a prediction is made, typically by taking the average (in regression) or the majority class (in classification) of the training data points contained in that region. The core idea is to recursively divide the feature space into simpler sub-regions using splitting rules. These rules are derived from the data and are chosen to optimize a certain criterion (e.g., reducing variance or impurity). Because these splits can be naturally visualized as a tree structure, such models are aptly named decision trees.

Root node: At the top of the tree lies the root node, which represents the entire dataset. From this node, the model makes its first decision or split, selecting the predictor variable and threshold value that most effectively reduce the variation in the target variable.

Internal node: Internal node, also known as a decision node, continues the process of partitioning the data based on specific values of the predictor variables. These nodes represent points where the dataset is divided further using decision rules, such as "Is variable X less than or equal to a certain value?". The goal at each internal node is to create child nodes that are more homogeneous with respect to the response variable than their parent node.

Leaf node: The tree continues to grow until it reaches leaf nodes or terminal nodes, which are the end points of the tree. In a regression tree, each leaf node contains a predicted value, typically the mean of the target variable for the observations in that node. This value is used as the model's prediction when a new observation falls into the corresponding region of the feature space.

Splitting: The process of dividing the data is called splitting. Each split is made using a rule that aims to reduce the impurity or variation of the target variable within the resulting nodes. For regression trees, the most common measure of impurity is variance or mean squared error (MSE). The model evaluates potential splits and selects the one that results in the greatest reduction in variance, meaning that it produces child nodes with more similar target values than the parent node.

Pruning: Growing a tree without constraints can lead to overfitting, where the model captures noise in the training data rather than the underlying pattern. To address this, trees are often pruned. Pre-pruning involves setting constraints such as the maximum depth of the tree or the minimum number of observations required to split a node. Post-pruning, on the other hand, involves building a full tree and then trimming branches based on performance on validation data.

Depth of the tree: It refers to the number of levels or splits from the root node to the deepest leaf node. While deeper trees can model complex relationships, they are more prone to overfitting. Therefore, striking a balance between model complexity and generalizability is essential.

3. Genesis of need for improved models like Random Forest regression

Limitations of decision trees: One major problem with decision trees is that they can only split data along straight lines based on one feature at a time (called axis-aligned splits). So, if the relationship in the data is complex, the tree needs to grow very deep with many splits to try and capture that complexity. Deep trees tend to memorize the training data too well, a problem known as overfitting. This means the model performs very well on the training set but poorly on new, unseen data. Due to this high variance and tendency to overfit, decision trees often perform worse compared to other models like Random Forests or Gradient Boosting Machines.

Bagging (short form for Bootstrap Aggregating): To solve the overfitting problem and reduce the variance of decision trees, a technique called Bagging (Bootstrap Aggregating) is used. The idea is simple but powerful. First, many new training datasets are created by randomly sampling the original dataset with replacement - this process is called bootstrapping. Then, train a separate decision tree is trained on each of these bootstrapped datasets. Since each sample is slightly different, each tree learns slightly different patterns. Once all trees are trained, predictions are made by combining their outputs. For regression tasks, this means averaging the predictions from all the trees. For classification, a majority vote will decide the final class label. This process of combining many models helps to reduce the risk of overfitting from any one tree. Even though individual trees may be overfit, their errors tend to cancel each other out when averaged, resulting in a more stable and accurate overall prediction. Bagging has two major benefits: high expressiveness and low variance. Each individual decision tree is allowed to be fully grown, so it can model complex patterns. At the same time, because we average many different trees, the result becomes much less sensitive to the noise or randomness in the training data. This leads to a model that is both powerful and reliable. Thus, Bagging is an effective way to improve decision trees, and it forms the basis for more advanced models like Random Forests.

In bagging, many decision trees are created by training each one on a different random sample (called a bootstrap sample) of the original dataset. However, even with different data, if a very strong predictor (say, a specific variable like "income") dominates, then most trees will end up splitting on that variable early in the tree. This means that although the trees are trained on different data, they behave similarly leading to 'high correlation' between trees. As a result, averaging their predictions may not reduce the variance as much as it is desired.

To fix ideas, consider the formula for the variance of the average prediction from B trees. If the trees are not completely independent and have a correlation ρ, the variance of their average is given by $Var(mean) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$. As one increases the number of trees (B), the second term (variance due to randomness) shrinks, but the first term (variance due to correlation) remains. This suggests that the overall variance reduction from bagging is limited by the correlation between trees. Random Forests address this limitation by adding an extra layer of randomness.

4. Random Forests Regression

Random Forests are an improvement over Bagging, especially in situations where the trees in the bagging ensemble are highly correlated. Like bagging, they use bootstrap samples to train each tree. But in addition, at each split in the tree, a random subset of features is selected. The best split is then chosen only from this subset, not from all features. This forces the trees to consider different variables and paths during training, which makes them less correlated. This 'de-correlation' increases the diversity of the trees, and thus boosts the effectiveness of averaging.

A good account on Random Forests can be found in, among others, Cutler *et al*. (2011), Protopapas and Rader (2025) etc. It has many real time applications, to cite one, Akselrud (2024) has employed Random Forests for pre-season predictions of total catches with uncertainty for California market squid (*Doryteuthis opalescens*), the most valuable fishery in California.

Tuning Random Forests: Random Forests also introduce hyperparameters that can be adjusted for better performance. These include

1. The number of predictors randomly chosen at each split: This controls how diverse the trees will be.
2. The number of trees in the forest: More trees usually lead to more stability, but come at a computational cost.
3. The minimum size of leaf nodes: This controls how deep each tree grows and helps avoid redundancy or overfitting in practice.

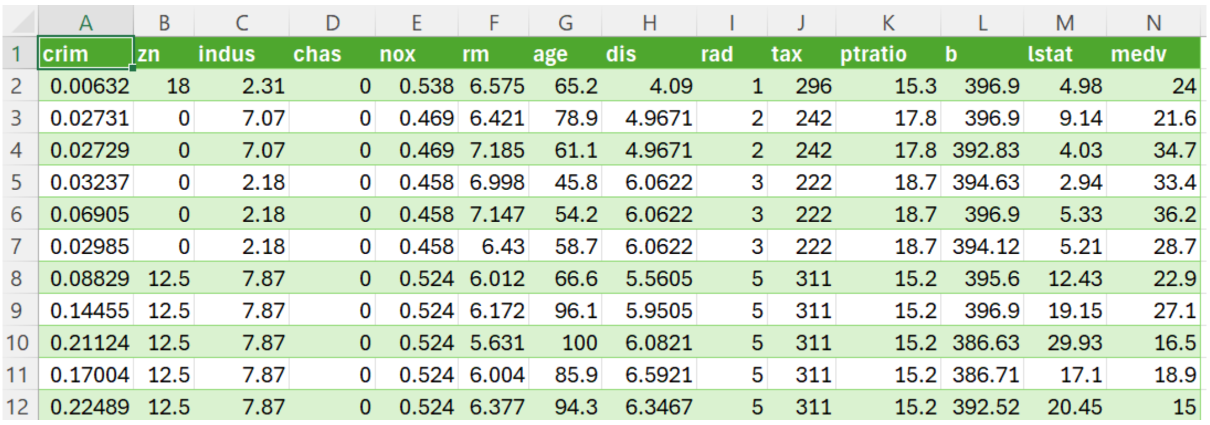
While there are standard default values (e.g., square root of the number of predictors), it is best to tune these parameters using cross-validation. Fortunately, Random Forests also offer a helpful built-in validation method: Out-of-Bag (OOB) error. Since each tree is trained on only part of the data, the unused portion (about one-third of the data) can be used to test the accuracy of the tree under consideration. This allows for efficient model evaluation without separate cross-validation sets.

Variable Importance in Random Forests: Another powerful feature of Random Forests is that they can estimate variable importance. For each tree, the model first predicts using the out-of-bag samples and records the accuracy. Then, it randomly shuffles the values of one feature in the data and measures how much the accuracy drops. If the accuracy decreases significantly, it means that feature was important for prediction. This process is repeated for each feature, and the average drop in accuracy tells us which variables are most useful in the model.

5. Data, R code for Random Forest Regression and interpretation of results

The following discussion is partially adopted from the work by Ehrlinger (2015).

# Data: Boston Housing Data <There are 506 records, hence only Preview given below>



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
2	0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
3	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
4	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
5	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
6	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
7	0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
8	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
9	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
10	0.21124	12.5	7.87	0	0.524	5.631	100	6.0821	5	311	15.2	386.63	29.93	16.5
11	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
12	0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15

The details about the variables in this Boston Housing data are:
Crim - Crime rate by town.
Zn - Proportion of residential land zoned for lots over 25,000 sq.ft.
indus -Proportion of non-retail business acres per town.
chas -Charles River (tract bounds river).
nox -Nitrogen oxides concentration (10 ppm).
rm -Number of rooms per dwelling.
age -Proportion of units built prior to 1940.
dis -Distances to Boston employment center.
rad -Accessibility to highways.
tax -Property-tax rate per $10,000.
ptratio -Pupil-teacher ratio by town.
black -Proportion of blacks by town.
lstat -Lower status of the population (percent).
medv -Median value of homes ($1000s).

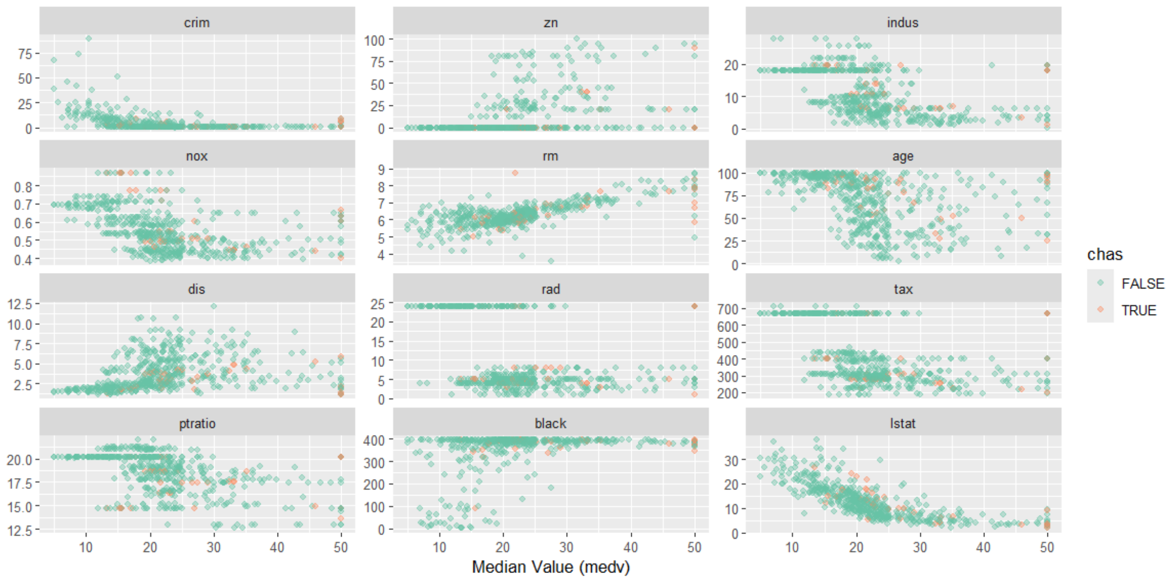
#R code for Random Forest Regression

# 1. Install and Load Required Packages

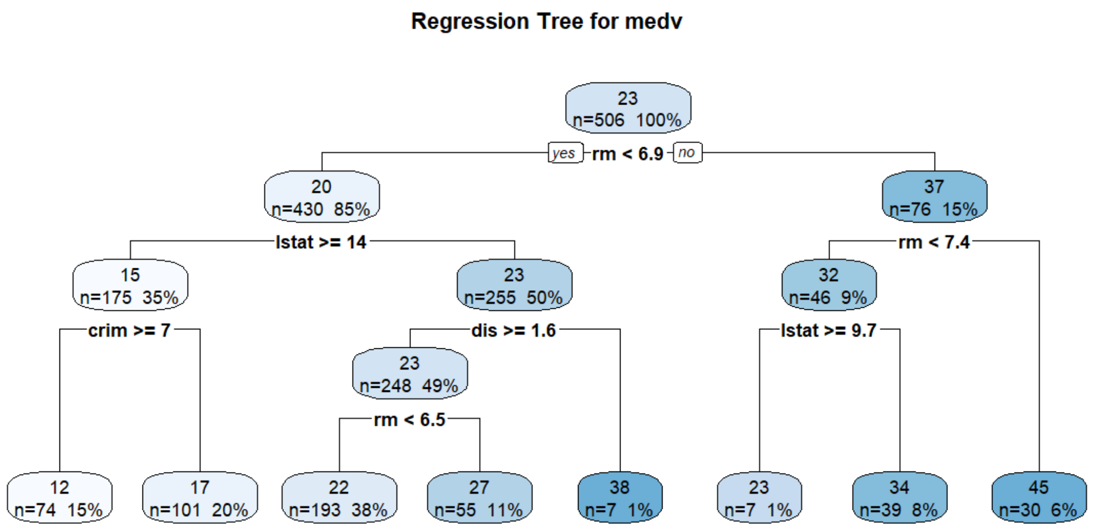
```
library(MASS)
library(randomForestSRC)
library(reshape2)
library(ggplot2)
library(rpart)
library(rpart.plot)
library(caret)

# 2. Load and Prepare the Boston Housing Data
data(Boston, package = "MASS")
Boston$chas <- as.logical(Boston$chas)

# 3. Melt Data for Exploratory Visualization
dta <- melt(Boston, id.vars = c("medv", "chas"))
ggplot(dta, aes(x = medv, y = value, color = chas)) +
  geom_point(alpha = 0.4) +
  labs(y = "", x = "Median Value (medv)") +
  scale_color_brewer(palette = "Set2") +
  facet_wrap(~variable, scales = "free_y", ncol = 3)

# 4. Random Forest Regression Model
set.seed(123)
rfsrc_Boston <- rfsrc(medv ~ ., data = Boston, ntree = 500, importance = TRUE)
print(rfsrc_Boston)

# 5. Variable Importance Plot
vimp <- sort(rfsrc_Boston$importance, decreasing = TRUE)
vimp_df <- data.frame(Variable = names(vimp), Importance = vimp)
ggplot(vimp_df, aes(x = reorder(Variable, Importance), y = Importance)) +
  geom_bar(stat = "identity", fill = "darkorange") +
  coord_flip() +
  labs(title = "Variable Importance", x = "Variables", y = "Importance") +
  theme_minimal()

# 6. Partial Dependence Plots
plot.variable(rfsrc_Boston, xvar.names = c("lstat", "rm"),
              partial = TRUE, show.plots = TRUE)

# 7. Tree-like CART Model for Comparison
tree_model <- rpart(medv ~ ., data = Boston, method = "anova")
rpart.plot(tree_model, type = 2, extra = 101, fallen.leaves = TRUE,
           main = "Regression Tree for medv")

# 8. Predict and Evaluate the Model
predictions <- predict(rfsrc_Boston)$predicted
results <- data.frame(Actual = Boston$medv, Predicted = predictions)
rmse_val <- RMSE(predictions, Boston$medv)
r2_val <- R2(predictions, Boston$medv)
```

```
cat("\nModel Performance:\n")
cat("RMSE:", round(rmse_val, 2), "\n")
cat("R-squared:", round(r2_val, 2), "\n")

# 9. Discretize medv for Confusion Matrix
Boston$medv_cat <- cut(Boston$medv,
                       breaks = quantile(Boston$medv, probs = seq(0, 1, 0.25)),
                       labels = c("Low", "MidLow", "MidHigh", "High"),
                       include.lowest = TRUE)

results$Predicted_cat <- cut(predictions,
                             breaks = quantile(Boston$medv, probs = seq(0, 1, 0.25)),
                             labels = c("Low", "MidLow", "MidHigh", "High"),
                             include.lowest = TRUE)

confusion <- confusionMatrix(results$Predicted_cat, Boston$medv_cat)
print(confusion)
```

# Interpretation



The Random Forest builds many decision trees to make better predictions. Each tree is built by choosing a random subset of the predictor variables at every decision point (called a "split"). The tree keeps splitting the data based on which variable best separates the values until it meets a stopping rule (like when the group is too small or the predictions are similar enough). In regression problems (like predicting house prices – here Median value of homes), the trees try to minimize the average squared error between the predicted and actual values. In the end, the Random Forest makes a prediction by averaging the results from all trees. For this Boston Housing example, the Random Forest was trained to predict medv (the median value of houses) using the other 13 variables (like crime rate, number of rooms, etc.). The forest had 1000 trees (the default) and considered five variables at each split.



The regression tree for the Boston housing dataset provides a clear and interpretable breakdown of how different variables influence the median home value (medv). The tree begins by splitting on the variable rm (average number of rooms per dwelling), with a threshold of 6.9 rooms. This initial split distinguishes between houses with fewer rooms (lower prices) and those with more rooms (higher prices), highlighting rm as a critical determinant of home value. For homes with fewer than 6.9 rooms, further splits are made on socioeconomic indicators such as lstat (percentage of lower status population) and crim (crime rate), which help identify areas with depressed home values due to higher poverty and crime. On the other side, homes with rm above 6.9 split on more refined thresholds of rm and lstat, ultimately identifying groups of homes with very high median values, especially those with low poverty and high room counts. The predicted values at each terminal node represent average home prices for that subgroup, and the structure of the tree offers valuable insights into how combinations of housing, demographic, and economic variables drive home values in the Boston area. This tree, though simpler than a random forest, helps us understand the key drivers and interactions in a visual, easy-to-interpret format.

References

- Akselrud, C.I.A. (2024). Random forest regression models in ecology: Accounting for messy biological data and producing predictions with uncertainty, *Fisheries Research*, 280, 107161, https://doi.org/10.1016/j.fishres.2024.107161

- Cutler, A., Cutler, D.R., Stevens, J.R. (2012). Random Forests. In: Zhang, C., Ma, Y. (eds) *Ensemble Machine Learning*. Springer, New York, NY, pp 157-175. https://doi.org/10.1007/978-1-4419-9326-7_5.

- Ehrlinger (2015). ggRandomForests: Random Forests for Regression, pages 1-30, https://arxiv.org/pdf/1501.07196

- Protopapas, P. and Rader, K. (2025). Regression Trees, Bagging and Random Forest, https://harvard-iacs.github.io/2018-CS109A/lectures/lecture-16/presentation/lecture16_bagging_random_forest.pdf, accessed on 30 June, 2025.

Xgboost Algorithm

Kapil Choudhary
College of Agriculture, Sumerpur (Pali)- 306902, Agriculture University, Jodhpur
Email: kapiliasri@gmail.com

1. Introduction

Time series data is a collection of observations recorded sequentially over time, typically at consistent intervals such as hourly, daily, monthly, or annually. This form of data is central to many real-world applications, including but not limited to financial market analysis, weather prediction, traffic forecasting, energy consumption monitoring, and agricultural output assessment. The temporal order and dependency of observations are crucial characteristics of time series data, which distinguish it from other types of datasets. Time series forecasting involves using historical data to predict future values. Accurate forecasting is vital in strategic decision-making processes across sectors such as finance (forecasting stock prices), energy (projecting electricity demand), agriculture (predicting crop prices), and healthcare (forecasting disease outbreaks). It aids in planning, resource allocation, inventory management, and risk assessment. To achieve reliable forecasts, it is essential to understand and model the underlying patterns of time series data, such as trends (long-term increases or decreases), seasonality (systematic calendar-related movements), cyclic behavior (longer-term fluctuations without fixed periodicity), and irregular variations (noise). Traditional time series models such as ARIMA, Exponential Smoothing, and SARIMA are statistical in nature and have been widely used for decades. While these models are effective in capturing linear patterns and seasonality, they often struggle with nonlinear relationships, complex feature interactions, and high-dimensional data. Moreover, they require assumptions such as stationarity, which may not hold in real-world scenarios. As a result, their applicability becomes limited when dealing with large, noisy, and intricate datasets.

To overcome these limitations, machine learning models have been increasingly adopted for time series forecasting. Among them, XGBoost (Extreme Gradient Boosting) has emerged as a powerful tool. XGBoost is a highly efficient and scalable implementation of the gradient boosting framework that builds an ensemble of decision trees sequentially to improve model performance. Originally designed for classification and regression on tabular data, XGBoost can be adapted for time series tasks through appropriate preprocessing techniques. Although

XGBoost does not inherently account for the temporal structure of data, it can effectively model time series when transformed into a supervised learning format. This involves the creation of lag features, rolling statistics, and calendar-based indicators. Once the time-based features are engineered, XGBoost can learn from the transformed data and capture complex, non-linear dependencies. It includes features like regularization to control overfitting, efficient handling of missing data, and support for parallel computation, which make it particularly suitable for large-scale forecasting problems. In essence, XGBoost provides a flexible and robust alternative to traditional time series models, especially when used in hybrid frameworks or combined with domain-specific feature engineering. Its adaptability and predictive power have made it a popular choice in both academic research and industry applications for forecasting tasks.

2. Gradient Boosting and the XGBoost Framework

To understand the strength of XGBoost, it is essential to first grasp the underlying principles of gradient boosting. Gradient boosting is an ensemble learning technique that constructs a strong predictive model by combining the outputs of several weaker models, typically decision trees. The main idea is to build the model in a sequential manner, where each new tree corrects the errors made by the previous ensemble. The model improves over iterations by minimizing a specified loss function using a gradient descent approach.

XGBoost enhances this process by incorporating several advanced features that make it faster, more accurate, and more scalable. It introduces regularization terms (L1 and L2) in the objective function to prevent overfitting, which is particularly beneficial when dealing with noisy or high-dimensional datasets. XGBoost also employs an efficient algorithm known as "approximate tree learning," which allows for faster computation and memory optimization. Additionally, it supports parallel and distributed computing, making it highly efficient for large datasets.

Another key innovation of XGBoost is its sparsity-aware algorithm, which handles missing values effectively during model training. This is crucial in real-world scenarios where time series data often contains gaps or missing records. Furthermore, XGBoost offers built-in cross-validation functionality, early stopping criteria, and tree pruning mechanisms, all of which contribute to improved model robustness and generalization performance. In the context of time series forecasting, these features allow XGBoost to adapt to the complexity and dynamics of temporal data, even though it is not inherently time-aware. By using lagged inputs and engineered time features, the model can learn from historical patterns and generalize them to future observations. This makes XGBoost not just a viable alternative, but often a superior choice compared to classical time series models, particularly when non-linearity, high variance, and multiple influencing variables are involved.

Ultimately, XGBoost serves as a bridge between traditional statistical methods and modern machine learning approaches. Its flexible architecture allows integration with other techniques such as signal decomposition, neural networks, or statistical preprocessing methods, enabling the development of hybrid models that leverage the strengths of multiple forecasting paradigms. Following are the mathematical formulation of the XGBoost model for time series forecasting:.

1. Feature Construction

Let the univariate time series be represented as $Y_t, t = 1,2,3 \dots T$, where T is the total number of observations. The objective is to learn a function f that maps past observations to a future value: $\hat{Y}_t = f(X_t)$, where X_t is a feature vector derived from past observations.

To frame the time series forecasting as a supervised learning problem, we construct feature vectors using lagged values: $X_t = [Y_{t-1}, Y_{t-2}, Y_{t-3} \dots, Y_{t-P}]$

This results in a training dataset: $(X_t, Y_t)\ for\ t = p + 1, \dots, T$

2. XG boost Model

XGBoost models the prediction as an ensemble of K regression trees:

$$\hat{Y}_t = \sum_{k=1}^{K} f_k(X_t)$$

Where $f_k \in \boldsymbol{\mathcal{F}}$ Here, $\boldsymbol{\mathcal{F}}$ is the space of regression trees.

3. Objective Function

The regularized objective function is defined as: $L = \sum_{t=p+1}^{T} l(\ Y_t, \hat{Y}_t) + \sum_{k=1}^{K} \varphi\ (f_k)$

Where l is typically the squared loss and the regularization term φ

4. Addative Training

XGBoost builds the model in an additive manner. At step m: $\hat{Y}_t(m) = \hat{Y}_t(m-1) + f_m(X_t)$

5. Forecasting

Once trained, the model forecasts the value at horizon h as:

$$\hat{Y}_{T+h} = f(X_{T+h})$$

6. Evaluation

Standard error metrics used for evaluating the model include:

$$\text{RMSE}= \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y(t) - \hat{y}(t))^2};$$

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y(t)-\hat{y}(t)}{y(t)}\right|$$

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y(t) - \hat{y}(t)|;$$

where $y(t)$ and $\hat{y}(t)$ stand for the actual values and predicted values

XG boost python code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import TimeSeriesSplit
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')

# Set style for plots
plt.style.use('seaborn-v0_8')
sns.set_palette("husl")
def install_packages():
    """Install required packages if not available"""
    import subprocess
    import sys
        packages = ['pandas', 'numpy', 'matplotlib', 'seaborn', 'scikit-learn', 'xgboost']

    for package in packages:
        try:
            __import__(package)
        except ImportError:
            print(f"Installing {package}...")
            subprocess.check_call([sys.executable, "-m", "pip", "install", package])

# Uncomment the line below if you need to install packages
# install_packages()

def read_data_from_clipboard():
    """Read data from clipboard"""
    try:
        # Try to read from clipboard
        df = pd.read_clipboard(sep='\t')
        print("Data successfully read from clipboard!")
        print(f"Data shape: {df.shape}")
        print("\nFirst few rows:")
        print(df.head())
        return df
    except Exception as e:
        print(f"Error reading from clipboard: {e}")
        print("Please make sure you have copied tab-separated data to clipboard")
        return None
def create_lag_features(data, lags):
    """Create lagged features for time series"""
    df = pd.DataFrame()
    for lag in lags:
        df[f'lag_{lag}'] = data.shift(lag)
    return df

def create_time_features(n):
    """Create time-based features"""
    time_index = np.arange(1, n + 1)
    df = pd.DataFrame({
        'trend': time_index,
        'trend_sq': time_index ** 2,
        'sin_annual': np.sin(2 * np.pi * time_index / 12),  # Assuming monthly data
        'cos_annual': np.cos(2 * np.pi * time_index / 12),
        'sin_quarterly': np.sin(2 * np.pi * time_index / 4),
        'cos_quarterly': np.cos(2 * np.pi * time_index / 4),
        'sin_weekly': np.sin(2 * np.pi * time_index / 7),   # Weekly pattern
        'cos_weekly': np.cos(2 * np.pi * time_index / 7)
    })
    return df

def calculate_metrics(y_true, y_pred):
    """Calculate evaluation metrics"""
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100
    return rmse, mae, mape

def plot_results(ts_data, test_predictions, forecasts, split_point):
    """Plot actual vs predicted values and forecasts"""
    plt.figure(figsize=(15, 8))

    # Plot last 50 points or all if less
    n_plot = min(50, len(ts_data))
    start_idx = len(ts_data) - n_plot

    # Actual data
    plt.plot(range(start_idx, len(ts_data)),
             ts_data[start_idx:],
             label='Actual', color='blue', linewidth=2)
        # Test predictions
    if len(test_predictions) > 0:
        test_start = max(split_point, start_idx)
        test_end = min(split_point + len(test_predictions), len(ts_data))
        test_range = range(test_start, test_end)
        test_pred_slice = test_predictions[:len(test_range)]

        plt.plot(test_range, test_pred_slice,
                 label='Test Predictions', color='red', linewidth=2, alpha=0.8)

    # Forecasts
    forecast_range = range(len(ts_data), len(ts_data) + len(forecasts))
    plt.plot(forecast_range, forecasts,
             label='Forecast', color='green', linewidth=2, marker='o')
    plt.title('XGBoost Time Series Forecasting Results', fontsize=16, fontweight='bold')
    plt.xlabel('Time Index', fontsize=12)
    plt.ylabel('Value', fontsize=12)
    plt.legend(fontsize=12)
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

  def plot_feature_importance(model, feature_names, top_n=15):
    """Plot feature importance"""
    importance_dict = model.get_booster().get_score(importance_type='weight')

    # Convert to DataFrame and sort
    importance_df = pd.DataFrame(list(importance_dict.items()),
                                 columns=['feature', 'importance'])
    importance_df = importance_df.sort_values('importance', ascending=True).tail(top_n)

    plt.figure(figsize=(10, 8))
    plt.barh(range(len(importance_df)), importance_df['importance'])
    plt.yticks(range(len(importance_df)), importance_df['feature'])
    plt.xlabel('Feature Importance', fontsize=12)
    plt.title(f'Top {top_n} Feature Importance', fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    return importance_df

def forecast_multi_step(model, last_values, time_features_future, n_ahead, max_lags):
    """Generate multi-step ahead forecasts"""
    forecasts = []
    current_lags = list(last_values[-max_lags:])

    for i in range(n_ahead):
        # Create feature vector
        lag_features = current_lags[::-1]  # Reverse for most recent first
        time_features = time_features_future.iloc[i].values

        # Combine features
        features = np.array(lag_features + list(time_features)).reshape(1, -1)

        # Make prediction
        pred = model.predict(features)[0]
        forecasts.append(pred)

        # Update lags for next iteration
        current_lags = current_lags[1:] + [pred]

    return np.array(forecasts)

def main():
    """Main function to run XGBoost time series forecasting"""

    print("=== XGBoost Univariate Time Series Forecasting ===\n")

    # Read data from clipboard
    df = read_data_from_clipboard()
    if df is None:
        return

    # Get the first numeric column
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    if len(numeric_columns) == 0:
        print("No numeric columns found in the data!")
        return

    ts_column = numeric_columns[0]
    ts_data = df[ts_column].values
```

```
print(f"\nUsing column: {ts_column}")
print(f"Time series length: {len(ts_data)}")
print(f"Data range: {ts_data.min():.4f} to {ts_data.max():.4f}")

# Parameters
max_lags = 12
forecast_horizon = 6
test_size = 0.2

# Create features
print("\nCreating features...")

# Lag features
lag_features = create_lag_features(ts_data, range(1, max_lags + 1))

# Time features
time_features = create_time_features(len(ts_data))

# Combine features
all_features = pd.concat([lag_features, time_features], axis=1)

# Create target variable (next period value)
target = pd.Series(ts_data).shift(-1)

# Remove rows with NaN values
valid_idx = ~(all_features.isnull().any(axis=1) | target.isnull())
X = all_features[valid_idx]
y = target[valid_idx]

print(f"Training samples after removing NAs: {len(X)}")

# Split data
split_point = int(len(X) * (1 - test_size))
X_train, X_test = X[:split_point], X[split_point:]
y_train, y_test = y[:split_point], y[split_point:]

print(f"Training samples: {len(X_train)}")
print(f"Test samples: {len(X_test)}")

# XGBoost parameters
params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'eta': 0.1,
    'max_depth': 6,
    'min_child_weight': 1,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'gamma': 0,
    'alpha': 0,
    'lambda': 1,
    'random_state': 42,
    'verbosity': 0
}

# Create DMatrix objects
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Train with early stopping
print("\nTraining XGBoost model...")

model = xgb.train(
    params=params,
    dtrain=dtrain,
    num_boost_round=1000,
    evals=[(dtrain, 'train'), (dtest, 'test')],
    early_stopping_rounds=50,
    verbose_eval=False
)

print(f"Optimal rounds: {model.best_iteration}")

# Make predictions on test set
test_predictions = model.predict(dtest)

# Calculate metrics
rmse_test, mae_test, mape_test = calculate_metrics(y_test, test_predictions)

print(f"\n=== Test Set Performance ===")
print(f"RMSE: {rmse_test:.4f}")
print(f"MAE: {mae_test:.4f}")
print(f"MAPE: {mape_test:.2f}%")

# Feature importance
print(f"\n=== Feature Importance ===")
importance_df = plot_feature_importance(model, X.columns)
print("\nTop 10 Features:")
print(importance_df.tail(10))

# Generate forecasts
print(f"\n=== Generating Forecasts ===")

# Create future time features
future_time_features = create_time_features(len(ts_data) + forecast_horizon)
future_time_features = future_time_features.iloc[len(ts_data):]
```

```
    # Generate forecasts
    forecasts = forecast_multi_step(
        model=model,
        last_values=ts_data,
        time_features_future=future_time_features,
        n_ahead=forecast_horizon,
        max_lags=max_lags
    )

    print(f"Forecasts for next {forecast_horizon} periods:")
    for i, forecast in enumerate(forecasts, 1):
        print(f"Period {i}: {forecast:.4f}")

    # Visualization
    print(f"\n=== Creating Visualizations ===")
    plot_results(ts_data, test_predictions, forecasts, split_point)

    # Summary
    print(f"\n=== MODEL SUMMARY ===")
    print(f"Model type: XGBoost")
    print(f"Training samples: {len(X_train)}")
    print(f"Test samples: {len(X_test)}")
    print(f"Number of features: {X.shape[1]}")
    print(f"Optimal boosting rounds: {model.best_iteration}")
    print(f"Test RMSE: {rmse_test:.4f}")
    print(f"Forecast horizon: {forecast_horizon}")

    # Return results
    results = {
        'model': model,
        'forecasts': forecasts,
        'test_rmse': rmse_test,
        'test_mae': mae_test,
        'test_mape': mape_test,
        'feature_importance': importance_df,
        'X_train': X_train,
        'X_test': X_test,
        'y_train': y_train,
        'y_test': y_test,
        'test_predictions': test_predictions
    }

    print(f"\nForecasting complete! Results stored in 'results' dictionary.")
    return results
# Run the main function
if __name__ == "__main__":
    results = main()
```

References

Semmelmann, L., Henni, S., & Weinhardt, C. (2022). Load forecasting for energy communities: a novel LSTM-XGBoost hybrid model based on smart meter data. *Energy Informatics*, *5*(Suppl 1), 24.

Xiong, G., Zhang, J., Fu, X., Chen, J., & Mohamed, A. W. (2024). Seasonal short-term photovoltaic power prediction based on GSK–BiGRU–XGboost considering correlation of meteorological factors. *Journal of Big Data*, 11(1), 164.

Deep Learning for Abiotic Stress Management in Agriculture: A Focus On RNN, GRU, CNN, LSTM And Transformers

G. Avinash

Chairman & CEO, Avyagraha Research and Analytics LLP, Ballari, Karnataka – 583132
Email: avinash143stat@gmail.com

Abstract

This chapter explores the application of deep learning (DL) models namely Recurrent Neural Networks (RNN), Gated Recurrent Units (GRU), Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Transformers in the context of abiotic stress management in agriculture. Abiotic stress factors such as drought, temperature extremes, and erratic rainfall pose significant threats to crop productivity and food security. Traditional statistical and machine learning methods often fall short in capturing complex temporal patterns and dependencies inherent in environmental and crop-related data. Deep learning models, with their capacity to learn non-linear, high-dimensional, and time-dependent features, offer robust alternatives for forecasting, classification, and decision support. The chapter introduces each architecture with relevant mathematical foundations and showcases their comparative strengths in handling time series data, such as rainfall and crop price forecasting. This comprehensive overview emphasizes the practical potential of DL models in enhancing resilience and precision in agricultural systems under changing climatic conditions.

Introduction & Methodology

Time series data in agriculture such as rainfall variability, temperature fluctuations, soil moisture dynamics, and crop phenology are critical for understanding and managing abiotic stresses like drought, heatwaves, salinity, and waterlogging. Traditional time series forecasting techniques such as Holt–Winters, Kalman Filters, ARIMA and SARIMA (Box et al., 1995) are commonly employed for such tasks. However, their reliance on assumptions like linearity, stationarity, and predefined lag structures limits their effectiveness in modeling the nonlinear, non-stationary, and complex interactions inherent in agrometeorological data. Statistical enhancements like ARCH/GARCH, TAR, and Smooth Transition Models offer some improvements but often involve cumbersome tuning and limited interpretability (Engle, 1982; Bollerslev, 1986; Tong & Lim, 2009).

In response to these challenges, machine learning models such as Support Vector Regression (SVR), Random Forests, and Gradient Boosting Machines (GBM) have been applied to rainfall prediction, yield forecasting, and early stress detection. These models are more flexible in capturing nonlinear patterns but still lack an innate capability to learn temporal dependencies from sequential data.

This is where Deep Learning (DL) transforms the game. By enabling models to learn patterns directly from raw sequences without manual feature engineering DL opens up new possibilities for accurate and scalable prediction in agriculture. Let's take a deep dive into Deep Learning, starting with Recurrent Neural Networks (RNNs) and their powerful variants!

Note: (Abbreviations and Hyperparameters details are listed at the end of the chapter in the supplementary section)

Understanding Sequential Networks

1. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a class of neural networks designed to model **sequential data**, where the current prediction depends not only on the current input but also on **past inputs**. This temporal dependency makes RNNs uniquely suitable for tasks like **weather forecasting**, Time Series Forecasting, **NDVI time series modeling**, or **crop stress prediction**, where the sequence of past conditions (Price Series, temperature, rainfall, etc.) directly influences future outcomes.

RNNs achieve this by maintaining a **hidden state** that gets updated at each time step as new data arrives. This allows the model to form a form of **internal memory**, capturing patterns over time.

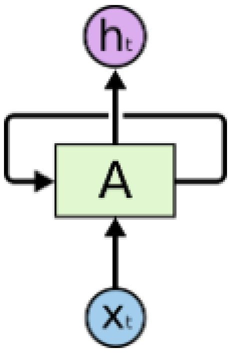


Figure 1: Recurrent Neural Network have loops (Ref: Olah, C. (2015))

In the above diagram, a chunk of neural network, A, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.

These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

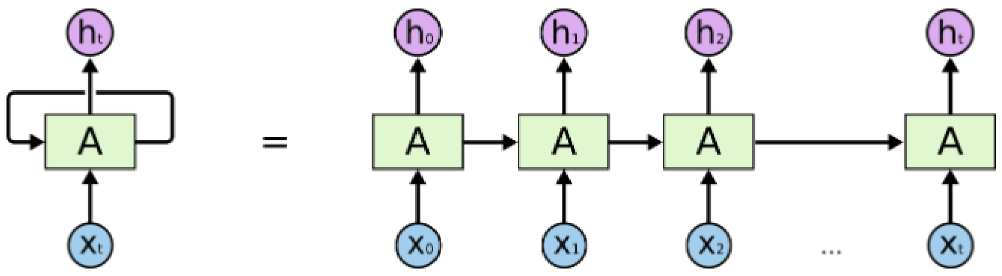


Figure 2: An unrolled recurrent neural network (**Ref**: Olah, C. (2015))

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

And they certainly are used! In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning… The list goes on.

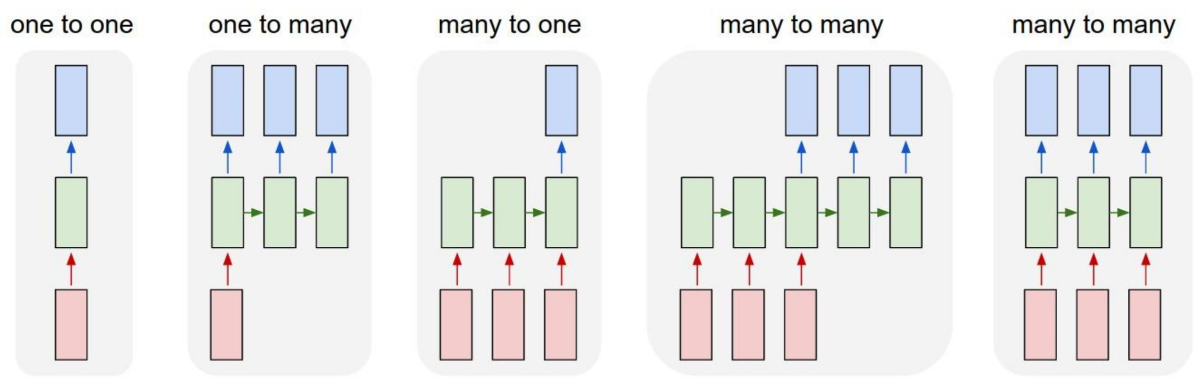


Figure3: Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. (Ref: Karpathy 2015)

1.1.Core Mechanism

Training RNNs involves backpropagation through time (BPTT), adjusting weights to refine prediction accuracy. Despite exhibiting various architectures, such as, one-to-one, one-to-many, many-to-many and many-to-one. RNNs encounter challenges like exploding and vanishing gradients, impacting stability and learning efficiency. Sequential data processing also constrains scalability, potentially leading to slow training times, rendering them less suitable for certain sequential tasks.

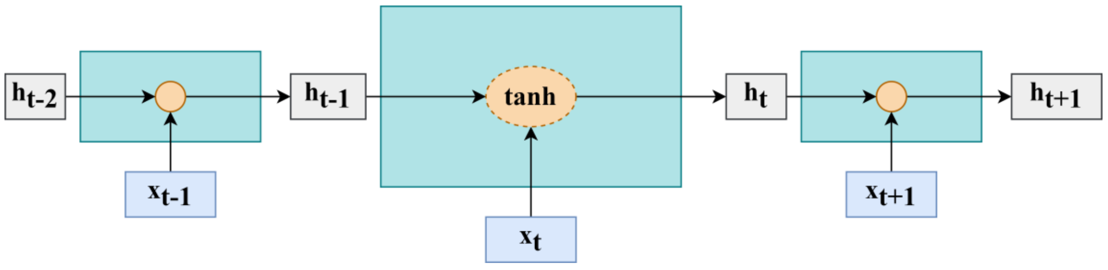


Figure 4: RNN architecture (Avinash *et al.*, 2024)

Hidden State Update: The workflow involves sequential input processing and hidden state update. At each time step t, the hidden state h_t is computed using the input x_t and the previous hidden state h_{t-1}:

$$h_t = tanh(W_{ih}.x_t + W_{hh}.h_{t-1} + b_h) \quad \text{... [1]}$$

Output Prediction: The output $\hat{y}_t$ is predicted based on the current hidden state:

$$\hat{y}_t = W_{hy}.h_t + b_y \quad \text{... [2]}$$

Loss Function (MSE): During training, the loss L_t:

$$L_t = \frac{1}{t}\sum_{t=1}^{T}(\hat{y}_t - y_t)^2 \quad \text{... [3]}$$

Gradient Computation: These are computed with respect to the model parameters for weight updates:

$$\frac{\partial L_t}{\partial W_{hy}}, \frac{\partial L_t}{\partial h_t}, \frac{\partial L_t}{\partial W_{ih}}, \frac{\partial L_t}{\partial W_{hh}} \quad \text{... [4]}$$

Weight Update (Gradient Descent): These gradients are used in the weight update through gradient descent as depicted in Eq. (5):

$$\theta_{new} = \theta_{old} - \eta.\frac{\partial L_t}{\partial \theta_{old}} \quad \text{... [5]}$$

Where:

- η: learning rate
- $\theta \in \{W_{ih},\ W_{hy}, W_{hh}, b_h, b_y\}$

Here, W_{ih}, W_{hh}, W_{hy} are the weight matrices, b_h, b_y are the biases, the activation function used is typically $\boldsymbol{tanh}$, which helps bound the hidden state values. However, due to issues like long-term dependency degradation, newer architectures such as Long Short-Term Memory

(LSTM) and Gated Recurrent Unit (GRU) have been developed to enhance performance on sequential learning tasks.

2. Long Short-Term Memory (LSTM)

In 1997, Hochreiter (Hochreiter, 1997) recognized that traditional RNNs were unable to retain important historical information for extended periods of time. To address this issue, they developed the LSTM model, which introduced gate mechanisms to the RNN framework.

LSTMs utilize three gate structures, namely the forget, input and output gates, which are implemented as sigmoid layers. These gates receive inputs from both the previous network output (h_{t-1}) and the current input (x_t) and are designed to decide whether to retain or delete the information processed by the previous cell state (C_{t-1}).

The forget gate plays a critical role in the LSTM architecture, as it determines whether previously processed information is necessary for further analysis. The output of this processing gate is represented as f_t. By incorporating these gate mechanisms, LSTMs can process and predict useful information with extended time intervals and delays in TS.

$$f_t = \sigma\left(W_f . [h_{t-1}, x_t] + b_f\right) \quad \text{... [6]}$$

The forget gate in the LSTM model plays a crucial role in determining the relevance of previously processed information. Its output, represented as f_t , determines whether information should be retained or discarded. A value of 0 indicates that the information should be completely discarded, while a value of 1 indicates that it should be retained entirely.

The forget gate utilizes a weight matrix $\left(W_f\right)$, a sigmoid function σ and a bias term $\left(b_f\right)$ to determine the importance of the previous cell state (C_{t-1}) in the current computation.

In addition to the forget gate, the LSTM model also employs an input gate, which determines which values require updating. This gate combines the previous network output (h_{t-1}) and current input (x_t) using a weight matrix (W_i), sigmoid function (σ) and bias term (b_i) to produce a new candidate value that can be added to the current cell state (C_t).

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i) \quad \text{... [7]}$$

The input gate of the LSTM model receives a bias term (b_i) in addition to the input weighted by the weight matrix (W_i). This input is processed using the sigmoid function (σ) to determine which information needs to be updated before being transferred to the cell state (C_t).

The output of the input gate (i_t), takes on values between 0 and 1, inclusive, indicating the degree to which information needs to be updated in the current computation. The use of a sigmoid function allows for this output to be interpreted as a probability and ensures that the input gate's influence on the current computation remains bounded.

$$\widehat{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c) \quad \text{... [8]}$$

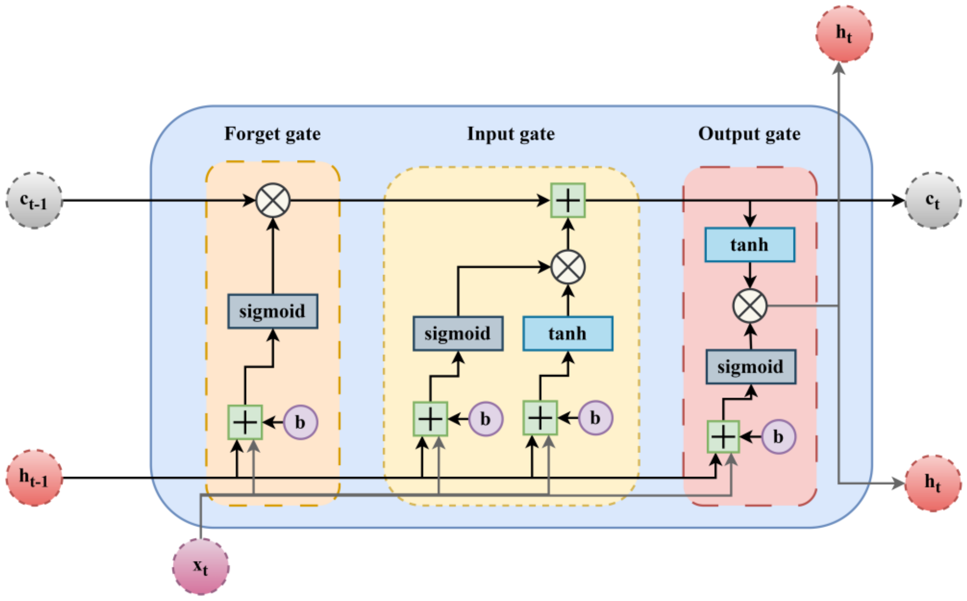


Figure 5: LSTM architecture (Avinash *et al.*, 2024)

Equation (8) represents the updated cell state value, denoted as $\widehat{C}_t$, which is computed by taking into account the current input and hidden node (h_{t-1}), which are weighted with W_c and added bias (b_c), respectively and then passed through the hyperbolic tangent $(tanh)$ function that yields a value ranging from –1 to +1. The forget gate's non-zero output (f_t) indicates that it contains useful information from the previous cell state (C_{t-1}), which is multiplied with (C_{t-1}) to forget old and irrelevant features. The output from the input gate (i_t) is then multiplied with the new candidate status $(\widehat{C}_t)$ to incorporate additional information and refine the updated (C_t).

$$C_t = f_t * C_{t-1} + i_t * \widehat{C}_t \quad \text{... [9]}$$

The current network module is determined based on the current cell state and the output gate decides which parts of the cell state should be used as output. The output gate utilizes a sigmoidal function to determine which information should be kept or discarded. The previous cryptic output and the current input are multiplied with the weight matrix (W_o) and added with the bias (b_o) to generate the input to the sigmoid function. The candidate value of the current output (h_t) is computed by taking the hyperbolic tangent $(tanh)$ of the current state (C_t), which is then multiplied with the output gate value (o_t) to produce the final output value.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \qquad \text{... [10]}$$

$$h_t = o_t * \tanh(C_t) \qquad \text{... [11]}$$

The fully connected layer of the LSTM model employs the Rectified Linear Unit (ReLU) as its activation function. To fine-tune the performance of the model, the mean square error (MSE) is utilized as the loss function.

3. Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU), introduced by Chung et al. (2014), is a streamlined and computationally efficient alternative to the Long Short-Term Memory (LSTM) network. Unlike LSTM, GRU utilizes a single hidden state by merging the cell state and hidden state, and it replaces the traditional three gates (input, forget, output) with only two: the update gate and the reset gate. This reduction in complexity leads to fewer trainable parameters, making GRUs faster and more cost-effective, particularly beneficial for time series forecasting applications.

GRU layers leverage the fact that recent events are more informative for predicting the future than distant past events. By remembering recent past information more efficiently than older information, GRU confirms the present task. The reset gate in GRU layers, which is composed of the reset and hidden states, determines the extent of previous information to forget, while the update gate remembers the useful information for predicting the present. The update gate describes how much of the GRU unit or cell will be updated.

$$Z_t(Update\ Gate) = \sigma(W_z.[h_{t-1}, x_t] + b_z) \qquad \text{... [11]}$$

$$R_t(Reset\ Gate) = \sigma(W_R.[h_{t-1}, x_t] + b_R) \qquad \text{... [12]}$$

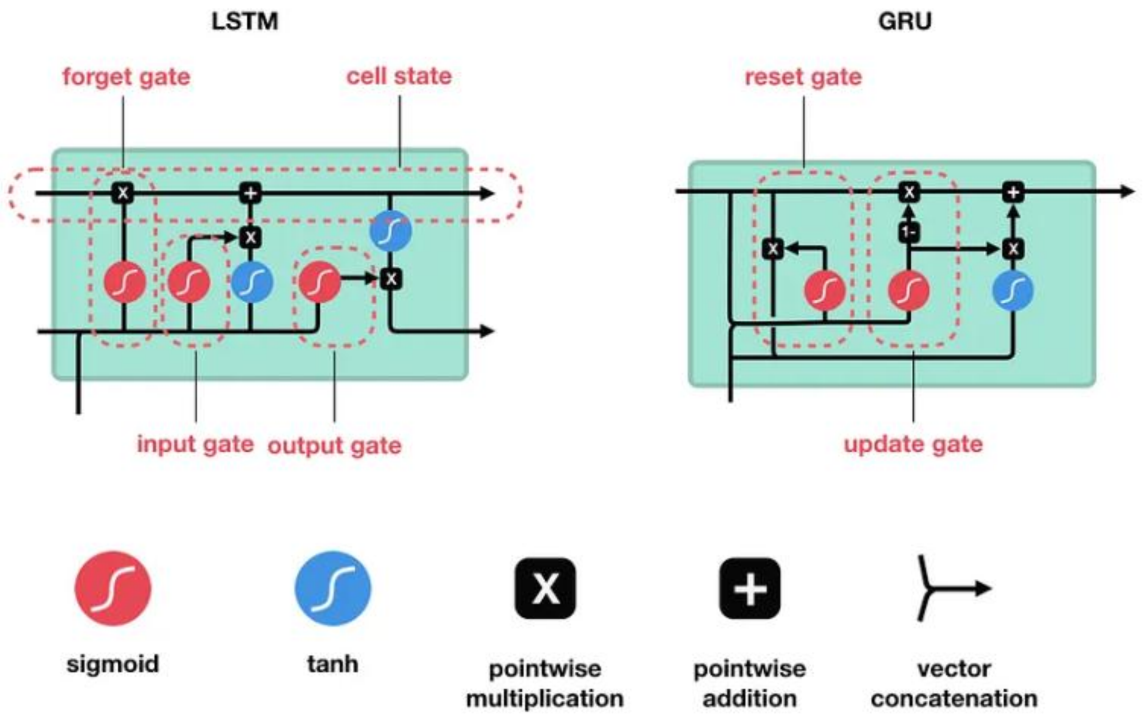


Figure 6: GRU architecture

The candidate activation in the GRU layers is computed using the hyperbolic tangent ($tanh$) function of the reset gate, as represented in equation (13). This function is used to regulate the amount of information that needs to be added to the cell state based on the reset gate's decision of which previous information to forget. The output of the $tanh$ function ranges between -1 and 1, which allows the GRU to adaptively control the flow of information and maintain the relevant information while discarding the irrelevant one.

$$\widetilde{h_t} = tanh(W.[R_t \, . \; h_{t-1}, x_t)] + b) \qquad \text{... [13]}$$

hidden state of the GRU layer is determined by the input and the previous hidden state.

$$h_t = \left(1 - Z_t\right) * h_{t-1} + Z_t * \widetilde{h_t}\right) \qquad \text{... [14]}$$

The update and reset gates, controlled by sigmoid activation function, are used to manipulate the recurrent connections and the inputs. The new member gate is obtained through the hyperbolic tangent function applied to the reset gate. Weight matrices, represented by W_Z, W_R and W, along with the bias terms b_Z and b_R are used to control the input values in the update and reset gates (Fig.6). The final hidden state is calculated by combining the previous hidden state and the new member gate with the update gate.

Due to their ability to balance memory efficiency with computational simplicity, GRUs have become a standard architecture for modeling agricultural time series such as rainfall forecasting, NDVI trend analysis, and drought prediction *etc*.

4. Transformer

The Transformer model is an encoder/decoder-based architecture that is utilized for machine translation tasks in which it translates one sequence of language to another proposed by (Vaswani *et al.*, 2017). Now it is widely used in time series, vision, genomics, and climate modeling. Through the use of self-attention mechanisms, the Transformer model is able to effectively evaluate the significance of different elements within the input data. This capability enhances the accuracy of the model's predictions, making it highly efficient for various tasks. Moreover, Transformer is equipped with an attention mechanism that facilitates faster learning compared to other DL architectures.

4.1.The Architecture of Transformer:

The Transformer model is a type of neural sequence transformation model that utilizes an encoder-decoder structure. The encoding process involves transforming an input sequence represented symbolically as $(x_1, \dots, x_n)$ into a continuous representation $z = (z_1, \dots, z_n)$. The decoder then generates the output sequence $(y_1, \dots, y_m)$ symbol by symbol using z as input. The model is autoregressive, meaning it uses newly generated symbols based on previously created ones at each stage. The encoder and decoder layers are stacked and interconnected for self-awareness, with the encoder depicted in the left half and the decoder in the right half of Figure 7.

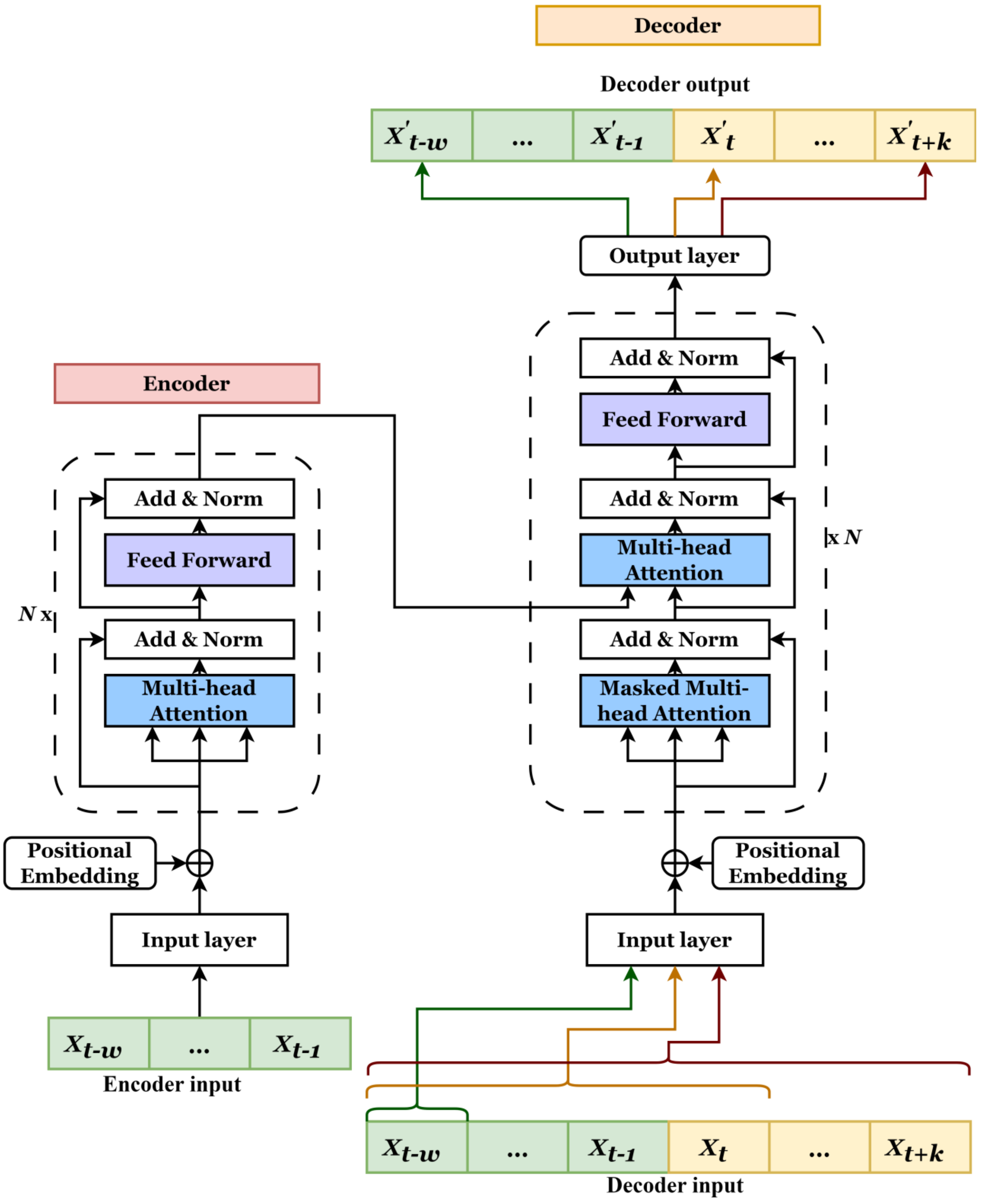


Figure 7: Transformer architecture (Nayak *et al*., 2024b)

4.1.1. Input Embedding and Positional Encoding: The purpose of the input embedding is to transform each item in the input sequence into a high dimensional vector space. This transformation allows the model to capture more complex features of each of them.

$$Embedding(X_i) = X_i E \qquad \text{... [15]}$$

where,

X_i: Represents the i^{th} item in the input sequence

E: The embedding matrix, typically learned during training

For TS, each X_i could represent a point in time or a set of features at that time point and the embedding layer captures temporal features in a higher dimensional space.

Since the Transformer model lacks any inherent sense of sequence order (due to the absence of recurrent structures), positional encodings are added to give the model information about the position of each item in the sequence.

$$Positional\ Encoding\ (pos, 2i) = sin\left(\frac{pos}{10000^{2i/d}}\right) \qquad \text{... [16]}$$

$$Positional\ Encoding\ (pos, 2i+1) = cos\left(\frac{pos}{10000^{2i/d}}\right) \qquad \text{... [17]}$$

where,

pos: The position of the item in the sequence

i: The dimension index of the positional encoding

d: The dimensionality of the embeddings

The positional encoding for each dimension is calculated using sinusoidal functions, with the sine function applied to even indices and the cosine function to odd indices.

These equations generate a unique positional encoding for each position in the sequence. The use of sinusoidal functions helps the model to easily learn to attend by relative positions. The positional encodings are added to the embedding vectors, ensuring that each position in the sequence is distinguishable and the sequential nature of the data is preserved.

$$X' = Embedding(X) + Positional\ Encoding \qquad \text{... [18]}$$

This combination of embeddings and positional encodings forms the initial representation of the input data that is fed into the subsequent layers of the Transformer model.

4.1.2. *The Encoder Block:*

The encoder module is made up of $N = 6$ encoder layers that are stacked on top of each other. Each encoder layer comprises of two sublayers - a multi-head self-attention layer and a feedforward layer, which are connected by a residual link and a normalization layer. The residual connection is a common technique used in deep neural network training to improve learning and stability. Additionally, layer normalization is frequently used in neural networks for analysing sequential data and to aid in training convergence. The feedforward layer consists of two linear layers with ReLU activation functions. The output of one encoder block becomes the input to the next encoder block. To form the input for the initial encoder block, the word embeddings and position-encoding vectors are added together (Ahmed *et al.*, 2023). In order to facilitate the residual connections, both the embedding layers and the sub-layers of the model produce outputs with a dimension of $d_{model} = 512$.

4.1.3. *The Decoder Block:*

The decoder component is comprised of $N = 6$ identical layers of stacked decoders, each of which contains the same layers and operations as its corresponding encoder block. However, unlike the encoder, the decoder receives two inputs: one from the previous decoder and one from the latest encoder. The decoder consists of three sublayers: (1) multi-headed self-attention, (2) encoder/decoder attention layer and (3) feedforward layer, along with residual connections and layer normalization operations. The final output of the encoder is used to create a set of key-value vectors in the attention layer of the encoder/decoder. The query vector is generated using the output of the preceding multi-head self-attention layer before the encoder/decoder layer.

4.2.*Self – Attention:*

The primary distinction between the attention mechanism and conventional RNN or LSTM models lies in the attention mechanism's ability to focus directly on specific segments of a sequence rather than treating them uniformly based on their order. This unique characteristic enables the model to capture information from early positions in the sequence, enhancing its comprehension of the overall context.

The Transformer architecture utilizes dot products to establish connections between various input segments, with location information added to those segments. A single sequence of n words or data points $\{x_i\}_{i=1}^{n}$, where $x \in R^d$ is represented. The index i corresponds to the position of the vector x_i, indicating the position of the word in the original sentence or phrase. The self-attention mechanism involves computing a weighted dot product between these input vectors x_i. The self-attention process has two steps. Firstly, normalized dot products are calculated between each input vector in the given input sequence. The normalization is done using the softmax operator, which scales the set of numbers to ensure that they add up to 1. The resulting normalized correlations are then calculated between a single input segment x_i and all other segments $j = 1, \dots, n$

$$w_{ij} = softmax(x_i^T x_j) = \frac{e^{x_i^T x_j}}{\sum_j e^{x_i^T x_j}} \qquad \dots [19]$$

where, $\sum_{j=1}^{n} w_{ij} = 1$ and $1 \leq i, j \leq n$.

In the second step, for a specific input segment x_i a new representation z_i is obtained by computing a weighted sum of all input segments $\{x_i\}_{j=1}^{n}$.

$$z_i = \sum_{j=1}^{n} w_{ij}\, x_j, \quad \forall \quad 1 \leq i \leq n \qquad \dots [20]$$

It is important to note that the weights w_{ij} for any input segment x_i sum up to 1. Consequently, the resulting representation vector z_i is similar to the input vector x_j that has the highest attention weight w_{ij}. The maximum attention weight is determined by the maximum correlation value obtained by the normalized inner product between x_i and x_j. It is worth mentioning that the position of z_i in the sequence is the same as that of x_i. The process continues for the subsequent output vector $z_i + 1$ and a new set of weights corresponding to$x_i + 1$ is computed and utilized.

4.2.1. Linearly Weighting Input Using Query, Key and Value:

To perform self-attention in Transformers, the model first creates three vectors (query q, key k and value v) from the input sequence $\{x_i\}_{i=1}^{n}$. These vectors are obtained by linearly combining the input features and they have different sizes ($\boldsymbol{q} \in \mathbb{R}^{s_1}$, $\boldsymbol{k} \in \mathbb{R}^{s_1}$ and $\boldsymbol{v} \in \mathbb{R}^{s}$).

After applying linear transformations to the input sequence $\{x_i\}_{i=1}^{n}$, the Transformer generates three vectors (query, key and value), each with a size of $s_1 = s = d$. The input sequence has n elements, which means that the Transformer produces n query vectors, n key vectors and n value vectors (Ahmed *et al.*, 2023).

To obtain the query q_i, key k_i and value v_i vectors from the input x_i, the Transformer applies linear transformations to the input x_i using three sets of learnable weights:

$$q_i = W_q x_i,\ k_i = W_k x_i \text{ and } v_i = W_v x_i \qquad \text{... [21]}$$

where W_q and $W_k \in \mathbb{R}^{s_1 \times d}$, $W_v \in \mathbb{R}^{s \times d}$, depict learnable weight matrices. The output vectors $\{z_i\}_{i=1}^{n}$ are given by,

$$z_i = \sum_j softmax\left(q_i^T k_j\right) v_j \qquad \text{... [22]}$$

In the self-attention mechanism of Transformers, the relevance of a value vector v_i to a query vector q_i is determined by its correlation with a key vector k_j at a different location j. The strength of this correlation is reflected in the dot product of q_i and k_j, which tends to increase with the sizes of the corresponding vectors. However, since the softmax function used to calculate attention weights is sensitive to large values, the dot product is scaled down by the square root of the dimension $d_q = 64$. of the query and key vectors to avoid instability in the attention weights.

$$z_i = \sum_j softmax\left(\frac{q_i^T k_j}{\sqrt{d_q}}\right) v_j \qquad \text{... [23]}$$

When represented in matrix form, the self-attention operation in Transformers can be expressed as follows:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \qquad \text{... [24]}$$

where Q and $K \in \mathbb{R}^{s_1 \times n}$ and $V \in \mathbb{R}^{s \times n}$, $Z \in \mathbb{R}^{s \times n}$ and T depicts the transpose mechanism.

4.3.Multi-Head Self-Attention

The input data X in Transformers may contain correlations that can be leveraged at multiple layers for effective learning. To achieve this, multiple self-attention heads can be used in parallel on the same input, each employing unique weight matrices W_q, W_k and W_v, to extract distinct correlation values between the input data. In Transformers, each self-attention head is constructed using a unique set of weight matrices for query, key and value vectors. By using multiple heads, the model can evaluate its self-perception on the input sequence in parallel, with each head computing its own attention scores independently (Zeng *et al.*, 2021). The concept of employing multiple self-attention heads in Transformers is analogous to the use of multiple kernels in Convolutional Neural Networks (CNNs). In CNNs, each kernel is responsible for learning a distinct representation or property at each level of the network. Similarly, in Transformers, each head is designed to extract unique correlation information from the input data, contributing to a more comprehensive and effective learning process.

*4.4.**Masking in Self-Attention***

In the training phase of Transformers, a multi-headed self-attention layer in the decoder masks portions of the target input to prevent the model from using future data points during the self-attention process. This ensures that the decoder only processes previously predicted data and does not try to anticipate future inputs. During training, the decoder does not receive the model's projected output; instead, it uses the actual targets to drive learning. In the testing phase, the expected words in the sequence are fed back to the decoder after passing through the word embedding layer and position-coding vector.

4.5.Residual Connections

The use of residual connections in neural networks enables the gradient to flow directly from the input to the output through bypasses. In Transformers, these connections are employed to stabilize the training process, reduce the risk of vanishing gradients in deep neural networks, enhance the model's generalization and facilitate efficient learning. Residual connections are incorporated into both the encoder and decoder layers of the Transformer model. The output of each sub-layer (self-attention or feed-forward) is added to its input (residual connection) before being passed to the next layer

$$Output = Sublayer(x) + x \qquad \text{... [25]}$$

4.6.Layer normalization

Layer normalization is a neural network technique that standardizes the distribution of intermediary layers to facilitate gradient smoothing during training, resulting in quicker convergence and better generalization to new data. This method is commonly utilized to improve the performance of deep neural networks.

$$LayerNorm(x) = \gamma.\frac{x-\mu}{\sigma} + \beta \qquad \text{... [26]}$$

where,

μ and σ are the mean and standard deviation of the features

γ and β are learnable parameters

4.7.Feed-forward Network

The feed-forward network in the Transformer model, which contain around two-thirds of its parameters, have received less attention. These layers utilize a basic feedforward neural network to convert the attention vector into a format that can be processed by subsequent encoding or decoding layers. In contrast to RNNs, the feedforward network processes each attention vector independently and these vectors are not dependent on each other (Geva *et al.*, 2020). This parallelization allows for all words to be sent simultaneously to the encoder block and encoded at the same time, resulting in improved efficiency.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \qquad \text{... [27]}$$

where,

W_1 and W_2 are weight matrices and b_1 and b_2 are bias vectors

4.8.Linear layer

To augment the dimensionality of vectors, a linear layer is employed, which performs matrix multiplication of the input vectors with a weight matrix, resulting in an output with a higher number of dimensions. When dealing with machine translation, this layer is used to increase the dimensionality of the encoded vectors to match the number of words in the translated output in the target language.

$$Y = DecoderOutput * W + b \qquad \text{... [28]}$$

where,

$DecoderOutput$: The output from the final decoder block

W: Weight matrix of the linear layer

b: Bias vector

4.9.Softmax layer

After passing through the linear layer, the output undergoes a softmax activation function, which transforms the input into a probability distribution that is easy to interpret. The softmax layer outputs a probability distribution over the vocabulary, from which the most probable token is selected during inference. The purpose of the softmax function is to normalize the results of the linear layer, making them all positive and adding up to 1 (Tang and Matteson).

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \qquad \text{... [29]}$$

where,

z_i: The i^{th} element of the output vector from the linear layer

e^{z_i}: The exponential function applied to z_i

$\sum_j e^{z_j}$: The sum of the exponentials of all elements of all elements in the output vector

4.10. The Output

Transformers are a class of neural networks that are primarily used for natural language processing tasks, such as machine translation. They consist of two primary components: an encoder and a decoder, which are made up of multiple interconnected layers of nodes. The encoder processes the input data, while the decoder generates the output data. To train the model, an optimization technique such as Adam and a loss function like mean squared error (MSE) are typically used. By employing self-attention mechanisms, the Transformer model can effectively model sequential data, allowing it to capture complex dependencies of varying lengths in TS data (Wu *et al.*, 2020). The Transformer model's versatility is enhanced by its ability to be applied to a broad range of nonlinear dynamical systems and its flexibility to handle both univariate and multivariate TS data with minimal changes to the model. Furthermore, due to its parallel processing capability, the Transformer model can efficiently handle input data and achieve faster and more effective training than other sequence models such as RNNs, LSTMs and GRUs.

Question: Does Time Series Forecasting Use Both Encoder and Decoder in Transformers?

Use Cases	**Encoder**	**Decoder**	**Explanation**
Univariate forecasting (e.g., rainfall prediction)	Yes	Often skipped or simplified	Only encoder used to learn from the input sequence. Decoder is often omitted or reduced to a projection head.
Multivariate forecasting with long horizon (e.g., NDVI + rainfall → forecast 7 days)	Yes	Yes	Full encoder–decoder setup helps model complex input-output mappings
Sequence-to-sequence tasks (e.g., demand translation, data imputation, anomaly recovery)	Yes	Yes	True seq2seq structure needed
Classification of time series (e.g., crop stress or drought label)	Yes	No decoder	Only encoder needed to learn features for classification

Insights: In agricultural time series forecasting (e.g., predicting temperature, NDVI, evapotranspiration for next 7 days):

- If it’s multi-variate, long-horizon, or multi-output, you’ll benefit from an encoder–decoder Transformer.
- If it’s single-step forecast, univariate or simple regression, a well-designed encoder-only Transformer is sufficient (and faster).

5. One dimensional Convolutional Neural Network (1d-CNN)

1d-CNN is a formidable tool in TS analysis, showcasing remarkable success in recognizing temporal patterns. The fundamental architecture involves convolutional layers, sub-sampling layers and fully-connected layers, stacked together for efficient feature extraction and classification in TS data.

In the convolutional layer, nodes receive inputs from adjacent nodes in the preceding layer, mimicking the temporal dependencies found in sequential data. Employing shared local weights, this layer reduces memory usage and improves the network's capability for capturing the intricate time-dependent patterns:

$$Z_t = \sigma(W.X_t + b) \qquad \text{... [30]}$$

Sub-sampling layers which incorporate non-linear down-sampling, aiding in reducing the dimensionality of temporal data while enhancing the network's efficiency in learning sequential features:

$$Y_t = pooling(Z_t) \qquad \text{... [31]}$$

In the final fully-connected layer, analogous to conventional neural networks, a comprehensive matrix calculation occurs. This facilitates reasoning and generates the model's output based on the extracted temporal features, crucial for forecasting financial prices:

$$\hat{Y}_t = W_{output}.A_t + b_{output} \qquad \text{... [32]}$$

During training, the CNN optimizes model parameters to minimize the error between predicted and actual output values in a TS, utilizing gradient-based optimization using backpropagation to efficiently capture temporal dependencies:

$$L_t = \frac{1}{2}\sum_i \left(y_{t,i} - \hat{y}_{t,i}\right)^2 \qquad \text{... [33]}$$

In essence, the integration of convolutional, sub-sampling and fully connected layers equips CNNs to effectively discern temporal patterns and features in TS data, establishing them as a robust tool for various applications in TS analysis (Fig. 3.15).

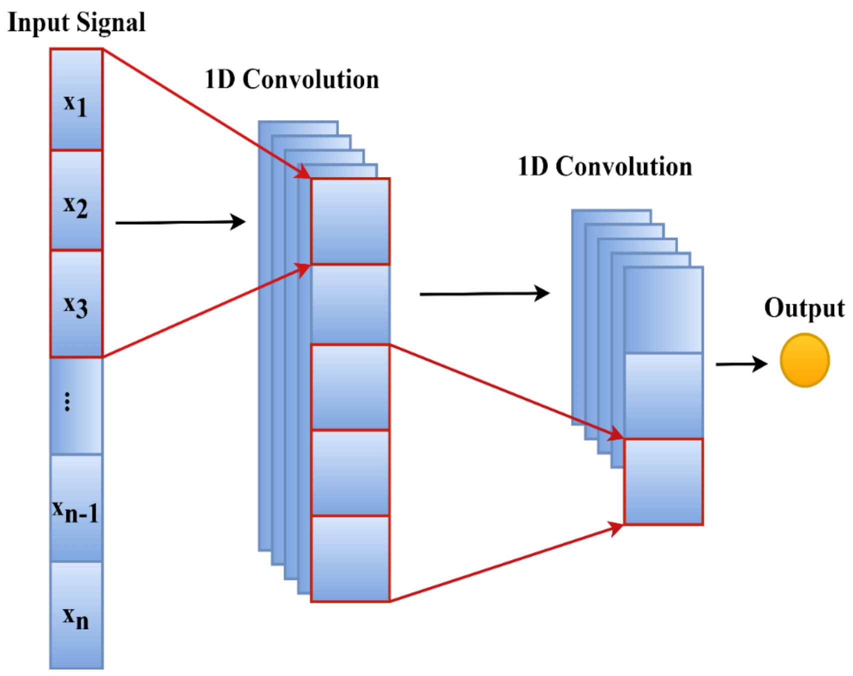


Figure 8: Convolutional Neural Network (1D-CNN) architecture (Nayak *et al*., 2024c)

<u>CNN model Illustration for Image classification</u>

Example: Mango Fruit Image Classification (Healthy vs Spongy)

Assume input image: 64×64×3 (RGB leaf image)

Class 0: Healthy fruit

Class 1: Spongy Fruit

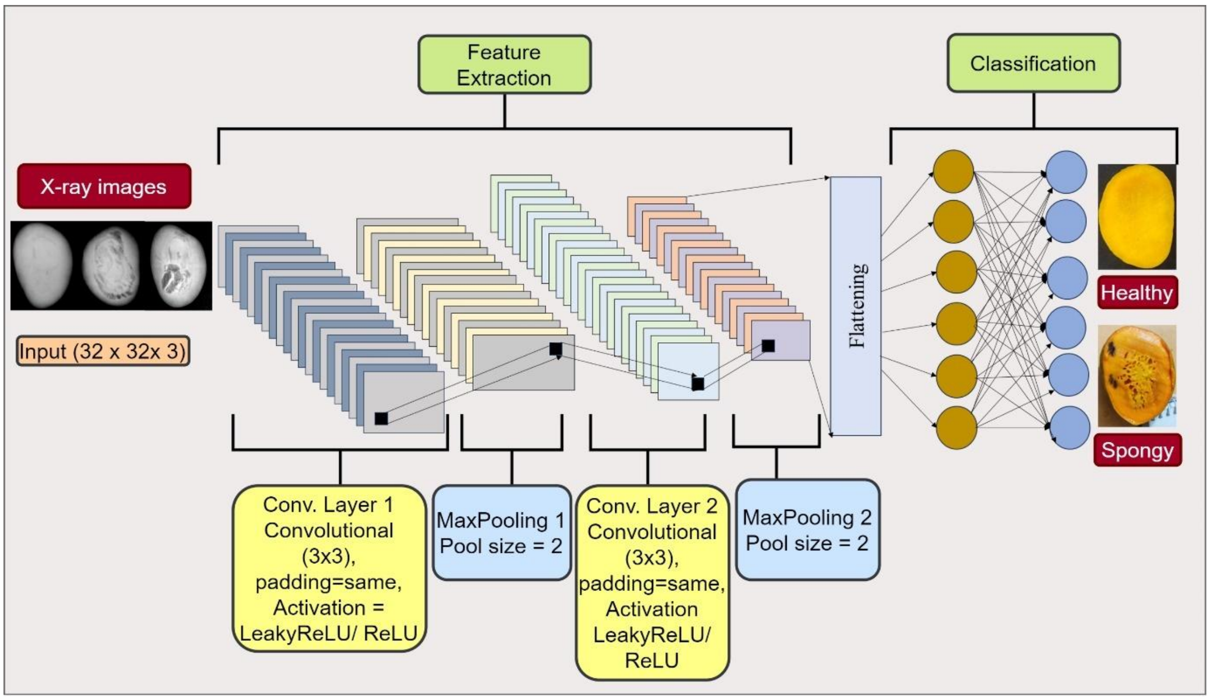

Feature Extraction
Classification
X-ray images
Input (32 x 32x 3)
Flattening
Healthy
Spongy
Conv. Layer 1 Convolutional (3x3), padding=same, Activation = LeakyReLU/ ReLU
MaxPooling 1 Pool size = 2
Conv. Layer 2 Convolutional (3x3), padding=same, Activation LeakyReLU/ ReLU
MaxPooling 2 Pool size = 2

Figure 9: CNN pipeline for Mango Image Fruit classification (Kiran et al. 2024)

Step-by-step CNN pipeline:

1. Convolution layer with 32 filters (3x3x3) → output 62x62x32

2. ReLU activation

3. Max pooling (2x2) → output 31x31x32

4. Flatten → vector of size 3072

5. Fully connected layer → logits

6. Softmax → probabilities [0.8, 0.2]

7. Output class: Healthy Fruit (80% confidence)

Supplementary Section

S1: Abbreviations and Acronyms in Deep Learning

Acronym	Full Form
DL	Deep Learning
ML	Machine Learning
AI	Artificial Intelligence
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
CNN	Convolutional Neural Network
FC	Fully Connected (Layer)
TS	Time Series
MSE	Mean Squared Error
MAE	Mean Absolute Error
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
CE	Cross-Entropy
BPTT	Backpropagation Through Time
POS	Part-of-Speech
QKV	Query, Key, Value (used in attention)
PE	Positional Encoding
d_model	Embedding vector dimension in Transformers
FFN	Feed-Forward Network
NLP	Natural Language Processing
NDVI	Normalized Difference Vegetation Index
ET	Evapotranspiration
PDS	Public Distribution System (in agriculture)
Adam	Adaptive Moment Estimation

S.2. Hyperparameters in Deep Learning

Hyperparameters are external configurations of the model set prior to training and have a significant influence on the performance of deep learning models. Unlike model parameters (weights and biases), Hyperparameters are not learned from the data but must be tuned manually or using automated methods (like grid search or Bayesian optimization).

Table S2: List of Common Hyperparameters

Hyperparameter	Description
Learning Rate (η)	Controls how much weights are updated during backpropagation.
Batch Size	Number of samples processed before model is updated.
Epochs	Full iterations over the entire training dataset.
Optimizer	Algorithm used to update weights (e.g., SGD, Adam, RMSProp).
Loss Function	Objective minimized during training (e.g., MSE for regression, CE for class).
Activation Function	Function applied to neuron outputs (e.g., ReLU, Sigmoid, Tanh).
Dropout Rate	Fraction of neurons randomly ignored to prevent overfitting.
Number of Layers	Number of hidden layers in the network.
Number of Neurons	Number of units per hidden layer.
Weight Initialization	Method for initializing weights (e.g., Xavier, He, Random Normal).
Early Stopping	Technique to halt training once validation loss stops improving.
Regularization (L1/L2)	Penalties added to loss function to reduce overfitting.
Window/Sequence Size	Number of time steps considered in time series models like RNN/LSTM.
Kernel Size (CNN)	Size of filter in convolutional layers.
Stride (CNN)	Step size for moving the kernel/filter over the input.
Pooling Size (CNN)	Dimensions of the pooling operation (e.g., 2×2 max pooling).
Embedding Size	Dimension of dense vector in NLP or time series input representation.
Number of Heads (Transformer)	Parallel attention mechanisms in multi-head attention.

d_model (Transformer)	Dimensionality of input/output of Transformer layers.

DL General Python code for TS forecasting

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import keras_tuner as kt
import matplotlib.pyplot as plt

# === Load and preprocess data ===
df = pd.read_csv("/content/Weekly_Dehradoon_Kalman.csv", parse_dates=['Date'])
#df = df[['Date', 'Price']].dropna()
df['Price'] = df['Price'].interpolate(method='linear')  # Linear interpolation
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# ACF and PACF first
plot_acf(df['Price'], lags=30)
plot_pacf(df['Price'], lags=30)
plt.show()

# Normalize
scaler = MinMaxScaler()
df['Price'] = scaler.fit_transform(df[['Price']])

# Create sequences
def create_sequences(data, seq_length=10):
    xs, ys = [], []
    for i in range(len(data) - seq_length):
        x = data[i:(i + seq_length)]
        y = data[i + seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 10
X, y = create_sequences(df['Price'].values, seq_length)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]
model_type = 'gru'  # Change this as needed
def build_model(hp):
    model = keras.Sequential()
    if model_type == 'rnn':
        model.add(layers.SimpleRNN(units=hp.Int("units", 32, 128, step=32),
                                   return_sequences=False,
                                   input_shape=(seq_length, 1)))
    elif model_type == 'gru':
        model.add(layers.GRU(units=hp.Int("units", 32, 128, step=32),
                             return_sequences=False,
                             input_shape=(seq_length, 1)))
    elif model_type == 'lstm':
        model.add(layers.LSTM(units=hp.Int("units", 32, 128, step=32),
                              return_sequences=False,
                              input_shape=(seq_length, 1)))
    elif model_type == 'cnn':
        model.add(layers.Conv1D(filters=hp.Int("filters", 32, 128, step=32),
                                kernel_size=hp.Choice("kernel_size", [2, 3, 4]),
                                activation='relu',
                                input_shape=(seq_length, 1)))
        model.add(layers.GlobalAveragePooling1D()) # Replaced MaxPooling1D and Flatten
with GlobalAveragePooling1D

    model.add(layers.Dense(1))
    model.compile(optimizer=keras.optimizers.Adam(hp.Choice("lr", [1e-2, 1e-3, 1e-4])),
                  loss='mse', metrics=['mae'])
    return model

tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=5,
    executions_per_trial=1,
    directory='tuning_results',
    project_name=model_type
)

tuner.search(X_train, y_train, validation_split=0.2, epochs=30, verbose=1)
best_model = tuner.get_best_models(1)[0]
best_model.summary()

preds = best_model.predict(X_test)
preds_rescaled = scaler.inverse_transform(preds)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

plt.plot(y_test_rescaled, label="True")
plt.plot(preds_rescaled, label="Predicted")
plt.title(f"{model_type.upper()} Forecasting")
plt.legend()
plt.show()
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error
# Calculate metrics
rmse = np.sqrt(mean_squared_error(y_test_rescaled, preds_rescaled))
mae = mean_absolute_error(y_test_rescaled, preds_rescaled)
mape = np.mean(np.abs((y_test_rescaled - preds_rescaled) / y_test_rescaled)) * 100

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"MAPE: {mape:.2f}%")
```

___________________ Transformer General Python code for TS forecasting__________

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt

# Mock a sample version of the CSV for demonstration
df = pd.read_csv("your_file.csv", parse_dates=['Date'])
df['Price'] = df['Price'].interpolate(method='linear')

# Normalize
scaler = MinMaxScaler()
df['Price'] = scaler.fit_transform(df[['Price']])
# Create sequences
def create_sequences(data, seq_length=10):
    xs, ys = [], []
    for i in range(len(data) - seq_length):
        x = data[i:(i + seq_length)]
        y = data[i + seq_length]
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)
seq_length = 10
X, y = create_sequences(df['Price'].values, seq_length)
X_train, X_test, y_train, y_test = train_test_split(X, y, shuffle=False, test_size=0.2)
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]

# Transformer Block
class TransformerBlock(keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = keras.layers.MultiHeadAttention(num_heads=num_heads,
key_dim=embed_dim)
```

```
        self.ffn = keras.Sequential(
            [keras.layers.Dense(ff_dim, activation="relu"), keras.layers.Dense(embed_dim)]
        )
        self.layernorm1 = keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = keras.layers.LayerNormalization(epsilon=1e-6)
        self.dropout1 = keras.layers.Dropout(rate)
        self.dropout2 = keras.layers.Dropout(rate)

    def call(self, inputs, training=None):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
# Positional Encoding
class PositionalEncoding(keras.layers.Layer):
    def __init__(self, sequence_length, d_model):
        super(PositionalEncoding, self).__init__()
        self.pos_encoding = self.positional_encoding(sequence_length, d_model)

   def get_angles(self, pos, i, d_model):
        angles = pos / np.power(10000, (2 * (i//2)) / np.float32(d_model))
        return angles
    def positional_encoding(self, position, d_model):
        angle_rads = self.get_angles(np.arange(position)[:, np.newaxis],
                                     np.arange(d_model)[np.newaxis, :],
                                     d_model)
        angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
        angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
        return tf.cast(angle_rads[np.newaxis, ...], dtype=tf.float32)
    def call(self, inputs):
        return inputs + self.pos_encoding[:, :tf.shape(inputs)[1], :]

# Build Transformer
def build_transformer_model(seq_len, d_model=64, num_heads=2, ff_dim=128):
    inputs = keras.Input(shape=(seq_len, 1))
    x = keras.layers.Dense(d_model)(inputs)
    x = PositionalEncoding(seq_len, d_model)(x)
    x = TransformerBlock(d_model, num_heads, ff_dim)(x, training=None)
    x = keras.layers.GlobalAveragePooling1D()(x)
    x = keras.layers.Dense(64, activation="relu")(x)
    outputs = keras.layers.Dense(1)(x)

    model = keras.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer="adam", loss="mse", metrics=["mae"])
    return model
```

```
# Train the model
transformer_model = build_transformer_model(X_train.shape[1])
history = transformer_model.fit(X_train, y_train, validation_split=0.2, epochs=30,
verbose=0)

# Predictions and evaluation
preds = transformer_model.predict(X_test)
preds_rescaled = scaler.inverse_transform(preds)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

rmse = np.sqrt(mean_squared_error(y_test_rescaled, preds_rescaled))
mae = mean_absolute_error(y_test_rescaled, preds_rescaled)
mape = np.mean(np.abs((y_test_rescaled - preds_rescaled) / y_test_rescaled)) * 100

print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
print(f"MAPE: {mape:.2f}%")
# Visualization
preds = transformer_model.predict(X_test)
preds_rescaled = scaler.inverse_transform(preds)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

plt.plot(y_test_rescaled, label="True")
plt.plot(preds_rescaled, label="Predicted")
#plt.title(f"{model_type.upper()} Forecasting")
plt.legend()
plt.show()
```

Suggested Readings

Ahmed, S., Nielsen, I. E., Tripathi, A., Siddiqui, S., Ramachandran, R. P. and Rasool, G. (2023). Transformers in time-series analysis: A tutorial. *Circuits, Systems, and Signal Processing*, **42(12)**, 7433-7466.

Avinash, G., Ramasubramanian, V., Ray, M., Paul, R. K., Godara, S., Nayak, G. H. and Iquebal, M. A. (2024). Hidden Markov guided Deep Learning models for forecasting highly volatile agricultural commodity prices. *Applied Soft Computing*, **158**, 111557.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, **31(3)**, 307-327.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. DOI: 10.48550/arXiv.1412.3555.

Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the econometric society*, 987-1007.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, **9(8)**, 1735-1780.

Karpathy, A. (2015, May 21). *The unreasonable effectiveness of recurrent neural networks*. https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Kiran, P. R., Avinash, G., Ray, M., Nigam, S., & Parray, R. A. (2024). Deep learning models for detection and classification of spongy tissue disorder in mango using X-ray images. Journal of Food Measurement and Characterization, 18(9), 7806-7818.

Nayak, G. H., Alam, M. W., Avinash, G., Singh, K. N., Ray, M. and Kumar, R. R. (2024a). N-BEATS Deep Learning Architecture for Agricultural Commodity Price Forecasting. *Potato Research*, 1-21.

Nayak, G. H., Alam, M. W., Singh, K. N., Avinash, G., Kumar, R. R., Ray, M. and Deb, C. K. (2024b). Exogenous variable driven deep learning models for improved price forecasting of TOP crops in India. *Scientific Reports*, **14(1)**, 17203.

Nayak, G. H., Alam, W., Singh, K. N., Avinash, G., Ray, M. and Kumar, R. R. (2024c). Modelling monthly rainfall of India through transformer-based deep learning architecture. *Modeling Earth Systems and Environment*, **10(3)**, 3119–3136.

Olah, C. (2015, August 27). Understanding LSTM networks. Colah's Blog. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Tong, H., & Lim, K. S. (1980). Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society: Series B (Methodological)*, **42(3)**, 245-268.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, **30**.

Hybrid TS Modelling: Applications in Abiotic Stress Management

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, Prabhat Kumar

ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115

Email: santosha.rathod@icar.org.in

Introduction

Time series refer to a set of observations recorded at regular intervals over time. Time series analysis (TSA) is an essential statistical tool used to understand the temporal dynamics of a variable, capture its historical behaviour, and project future outcomes. It is widely applied across disciplines to model and forecast time-dependent phenomena, such as stock prices, energy demand, rainfall, or crop yields. Among the classical approaches, the Box-Jenkins methodology for Autoregressive Moving Average (ARMA) modeling, introduced by Box and Jenkins (1970), is one of the most widely adopted techniques. Its structured three-step approach model identification, parameter estimation, and diagnostic checking—has made it the foundation for many forecasting applications involving linear time series data.

In the context of abiotic stress management in agriculture, time series models are critical for forecasting weather extremes and environmental conditions such as temperature, rainfall, drought indices, evapotranspiration, and soil moisture. These variables often exhibit strong seasonal trends and autocorrelations, making time series modeling a reliable method for predicting stress-prone periods. For instance, predicting low rainfall months ahead of the monsoon season enables proactive water conservation measures, drought preparedness, and the selection of suitable crop varieties.

By enabling the quantitative forecasting of climatic and environmental factors, TSA supports better risk management, resource allocation, and timely agricultural interventions. Moreover, advanced versions like ARIMA, SARIMA, and hybrid machine learning–statistical models provide flexibility to handle non-stationary and nonlinear behaviors common in abiotic stress datasets. Thus, time series modeling is not just a forecasting tool—it is a vital component in building climate-resilient agriculture that can anticipate and adapt to the challenges posed by environmental variability and stressors. A major limitation of traditional linear time series models, particularly the Autoregressive Integrated Moving Average (ARIMA) model, lies in their assumption of linearity. As a consequence, these models are inherently incapable of capturing nonlinear patterns present in many real-world time series datasets. However, in practice, time series often exhibit a combination of both linear and nonlinear structures. Under such circumstances, relying solely on ARIMA or machine learning models like Artificial Neural Networks (ANN) and Support Vector Machines (SVM) may prove inadequate for achieving satisfactory forecasting performance. To address this issue, hybrid modeling strategies have been developed, which aim to integrate the strengths of different approaches. Empirical studies (Zhang 2003; Jha and Sinha 2014; Chen and Wang 2007; Kumar and Prajneshu 2015; Ray et al. 2016) have shown that such hybrid models can significantly enhance forecasting accuracy by effectively modeling both linear and nonlinear components.

Autoregressive Integrated Moving Average (ARIMA) model

One of the most widely adopted classical models for time series analysis is the ARIMA model. Its popularity stems from its solid statistical foundation and the structured model building procedure proposed by Box and Jenkins (1970). Since many time series are non-stationary, the differencing operator d is applied to transform the series into a stationary form. Once the series becomes stationary through differencing, the general ARIMA model can be specified as ARIMA(p,d,q), where p is the order of the autoregressive part, d is the degree of differencing, and q is the order of the moving average part. A time series process YtY_tYt is said to follow an integrated ARMA process if $\Delta Y_t = (1-B)^d \varepsilon_t$, where B is the backward shift operator and $\varepsilon_t$is a white noise error term. The ARIMA model is mathematically expressed as:

$$\emptyset(B)(1-B)^d Y_t = \theta(B)\varepsilon_t \tag{1}$$

Where, $\varepsilon_t \sim WN\ (0,\sigma^2)$ and WN is the white noise. The Box-Jenkins ARIMA model building consists of three steps *viz.*, identification, estimation and diagnostic checking.

Artificial Neural Network for Time series

In contrast to linear models, artificial neural networks provide a flexible framework capable of approximating complex nonlinear relationships within time series data. For time-dependent data, the neural network architecture is adapted accordingly, resulting in what is known as a Time Delay Neural Network (TDNN). In such networks, time lags or delayed observations of the series are included as input features, allowing the model to learn temporal dependencies. A static neural network structure, such as the multilayer perceptron, is extended with dynamic properties by incorporating these time lags into its architecture (Haykin 1999). Thus, a common method for constructing a neural network for time series forecasting involves feeding past values of the series into the input layer. The output Y_t of the TDNN is then a function of these time-lagged inputs, transformed through the layers of the network using nonlinear activation functions. This architecture enables the network to capture intricate nonlinear relationships that are often missed by conventional linear models.

$$Y_t = \alpha_0 + \sum_{j=1}^{q} \alpha_j \, g\left(\beta_{0j} + \sum_{i=1}^{p} \beta_{ij} Y_{t-p}\right) + \varepsilon_t \qquad (2)$$

where, $\alpha_j (j = 0,1,2, \dots, q)$ and $\beta_{ij} (i = 0,1,2, \dots, p,\ j = 0,1,2, \dots, q)$ are the model parameters, also called as the connection weights, p is the number of input nodes, q is the number of hidden nodes and g is the activation function. The architecture of neural network is represented in figure 1.

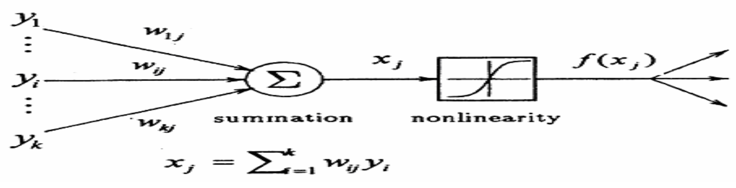


Fig.1: Artificial Neural Network Structure

Support Vector Machine for Time Series

Support Vector Machine (SVM) is a supervised machine learning approach that was initially developed for solving linear classification problems. In 1997, Vapnik extended the application of SVM to regression tasks by introducing the ε-insensitive loss function, thereby giving rise to the Support Vector Regression (SVR) framework (Vapnik, 1997). This innovation allowed the method to handle regression problems effectively, particularly those involving nonlinear relationships, and led to the development of the Nonlinear Support Vector Regression (NLSVR) model. The core idea behind NLSVR is to project the original input data into a higher-dimensional feature space where linear regression can be performed, thus capturing complex, nonlinear patterns in the data. To formalize this approach, let us consider a dataset represented by the vector $Z = \{x_i \, y_i\}_{i=1}^{N}$ where $x_i \in R^n$ denotes the input feature vector, y_i is the corresponding scalar output, and N is the total number of observations in the dataset. In NLSVR, a regression function is constructed in such a way that it approximates the relationship between inputs and outputs in the transformed feature space with minimal error, as defined by the ε-insensitive loss function. The general form of the nonlinear support vector regression estimation function can be expressed as follows:

$$f(x) = W^T\phi\,(x) + b \tag{3}$$

where ϕ(x) is a nonlinear mapping from the input space to a high-dimensional feature space, w is the weight vector, b is the bias term, and ⟨·,·⟩ denotes the dot product in the feature space. The goal of NLSVR is to find the optimal parameters w and b that minimize a regularized risk function while ensuring that the prediction error for each data point remains within a pre-defined ε-tube around the true value.

where $\phi(.)$: $Rn \rightarrow R^{nh}$ is a nonlinear mapping function which map the original input space into a higher dimensional feature space vector. $W \in R^{nh}$ is weight vector, b is bias term and superscript T denotes the transpose.

BDS (Brock-Dechert-Scheinkman) Test for testing Nonlinearity

BDS (Brock *et al.* 1996), test utilizes the concept of spatial correlation from chaos theory. The computational procedure is given as follows

v) Let the considered time series is

$$\{x_i\} = [x_1, x_2, x_3, \ldots, x_N] \tag{4}$$

vi) The next step is to specify a value of m (embedding dimension), embed the time series into m dimensional vectors, by taking each m successive points in the series. This transforms the series of scalars into a series of vectors with overlapping entries

$$\begin{aligned} x_1^m &= (x_1, x_2, \ldots, x_m) \\ x_2^m &= (x_2, x_3, \ldots, x_{m+1}) \\ &. \\ &. \\ &. \\ x_{N-m}^m &= (x_{N-m}, x_{N-m+1}, \ldots, x_N) \end{aligned} \tag{5}$$

vii) In the third step correlation integral is computed, which measures the spatial correlation among the points, by adding the number of pairs of points (i, j), where $1 \leq i \leq$ N and $1 \leq$

$j \leq N$, in the m-dimensional space which are "close" in the sense that the points are within a radius or tolerance ε of each other.

$$C_{\varepsilon,m} = \frac{1}{N_m(N_m - 1)} \sum_{i \neq j} I_{i,j;\varepsilon} \tag{6}$$

Where $I_{i,j;\varepsilon}$= 1 if $\left\| x_i^m - x_j^m \right\| \leq \varepsilon$

= 0 otherwise

viii) If the time series is *i.i.d.* then $C_{\varepsilon,m} \approx [C_{\varepsilon,1}]^m$

ix) The BDS test statistics is as follows

$$BDS_{\varepsilon,m} = \frac{\sqrt{N}[C_{\varepsilon,m} - (C_{\varepsilon,1})^m]}{\sqrt{V_{\varepsilon,m}}} \tag{7}$$

Where,

$$V_{\varepsilon,m} = 4[K^m + 2\sum_{j=1}^{m-1} K^{m-j} C_\varepsilon^{\ 2j} + (m-1)^2 C_\varepsilon^{\ 2m} - m^2 K C_\varepsilon^{\ 2m-2}]$$

$$K = K_\varepsilon = \frac{6}{N_m(N_m - 1)(N_m - 2)} \sum_{i<j<N} h_{i,j,N;\varepsilon}$$

$$h_{i,j,N;\varepsilon} = \frac{[I_{i,j;\varepsilon} I_{j,N;\varepsilon} + I_{i,N;\varepsilon} I_{N,j;\varepsilon} + I_{j,i;\varepsilon} I_{i,N;\varepsilon}]}{3}$$

The choice of m and ε depends on number of data. The null hypothesis is data are independently and identically distributed (*i.i.d.*) against the alternative hypothesis the data are not *i.i.d.* this implies that the time series is non-linearly dependent. BDS test is a two-tailed test; the null hypothesis should be rejected if the BDS test statistic is greater than or less than the critical values.

The Hybrid Methodology

The hybrid method considers the time series y_t as a combination of both linear and non-linear components. This approach follows the Zhang's (2003) hybrid approach, accordingly the relationship between linear and nonlinear components can be expressed as follows

$$y_t = L_t + N_t \tag{8}$$

Let L_t and N_t represent the linear and nonlinear components of a given time series, respectively. In the present study, the linear component is modeled using the Autoregressive Integrated Moving Average (ARIMA) model, while the nonlinear component is captured through Time Delay Neural Network (TDNN) and Nonlinear Support Vector Regression (NLSVR). The proposed hybrid methodology comprises three sequential steps.

In the first step, an ARIMA model is applied to capture and forecast the linear structure of the series. Let the linear forecast obtained from the ARIMA model be denoted by $\hat{L}_t$. In the second step, residuals are computed as $e_t = y_t - \widehat{L}_t$ where y_t represents the original series. These residuals are then tested for nonlinearity using the BDS test (Brock, 1996). If the test confirms the presence of nonlinearity, the residual series is modeled using TDNN and NLSVR. In the final step, the forecasts of the linear and nonlinear components are combined to obtain the aggregate forecast, thereby capturing both the linear and nonlinear dynamics inherent in the original time series.

$$\hat{y}_t = \hat{L}_t + \hat{N}_t \tag{9}$$

Where, $\hat{L}$and $\hat{N}$ represents the predicted linear and nonlinear component respectively. The graphical representation of hybrid methodology is expressed in figure 2 and 3. Finally, the performance of the models under consideration are compared using MSE (Mean Square Error), RMSE (Mean Square Error) and by MAPE (Mean Absolute Percentage Error).

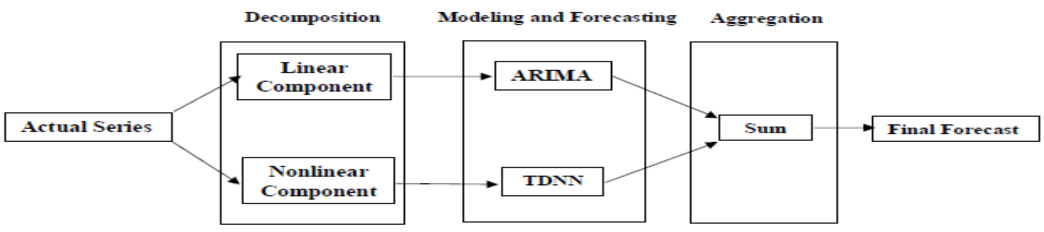


Fig. 2: Schematic representation of ARIMA-TDNN hybrid methodology

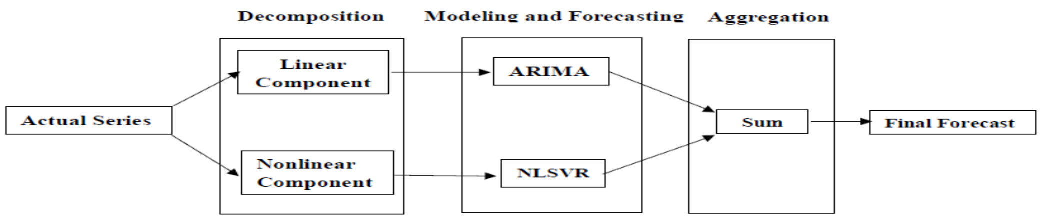


Fig. 3: Schematic representation of ARIMA-NLSVR hybrid methodology

This hybrid methodology approach can be further extended by using some other machine learning techniques for varying autoregressive and moving average orders so that practical validity of the model can be well established. The validity of hybrid models can be generalized by extending this approach to many agricultural real life data sets.

Illustration

Yearly data on mango yield (measured in MT/ha) for the state of Karnataka was obtained from the official database of the National Horticulture Board (NHB) and the website http://www.indiastat.com. To carry out the forecasting exercise, data spanning the period from 1980 to 2014 were considered. The dataset was split such that the observations from 1980 to 2011 were used for model development, while the data from 2012 to 2014 served as the validation set to evaluate the out-of-sample forecasting accuracy of the selected models. A time series plot of the mango yield in Karnataka over the years is shown in Fig. 4, which provides a visual representation of the trend and variability in the yield over time. To model the yield dynamics, an Autoregressive Integrated Moving Average (ARIMA) model was employed. Initial diagnostic analysis revealed that the original yield series was non-stationary, necessitating the application of first-order differencing to achieve stationarity. The appropriate ARIMA model was then selected based on the inspection of the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots. The model that best fitted the training data was found to be ARIMA(0,1,1). The parameters of the selected ARIMA model were estimated using the maximum likelihood estimation (MLE) method, and the corresponding estimates are reported in Table 1. Additionally, the model's predictive performance was assessed using both the training dataset and the validation dataset, with the results summarized in Tables 5 and 6, respectively.

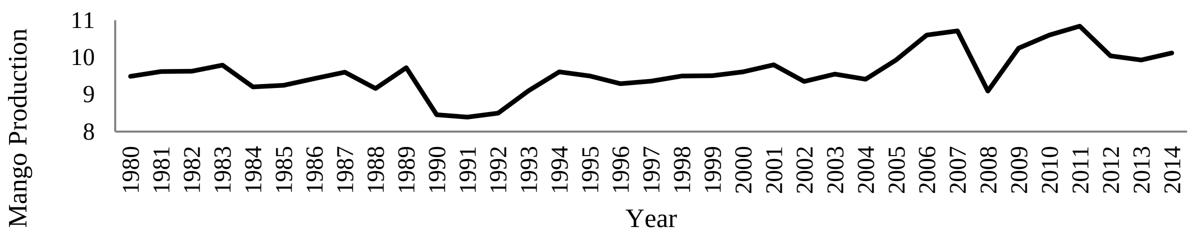


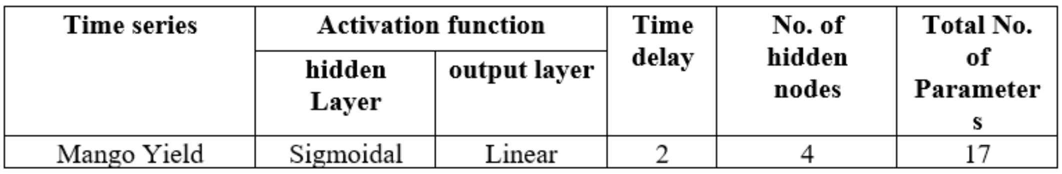
Fig. 4: Time series plot of mango yield time series

Table 1: Parameter estimation of ARIMA (0 1 1) for Mango Yield time series.

Parameter	Estimate	Standard Error	t Value	Approx. Pr > |t|	Lag	P(Resi.) at 6 Lag
Constant	0.033	0.038	0.87	0.382	0	0.240
MA 1	0.581	0.161	3.64	0.003	1	

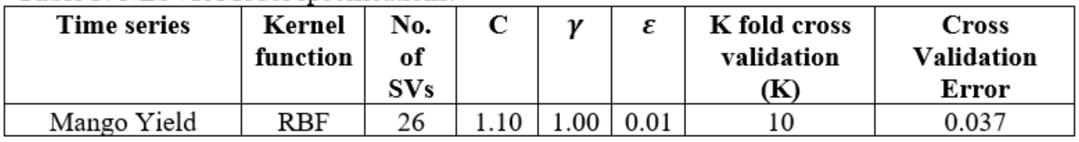
The TDNN and NLSVR models were fitted to mango yield time series of Karnataka and the model specifications are given in table 2 and 3. Further the model performance in training set and testing data set is given in tables 5 and 6.

Table 2: TDNN Model Specifications.



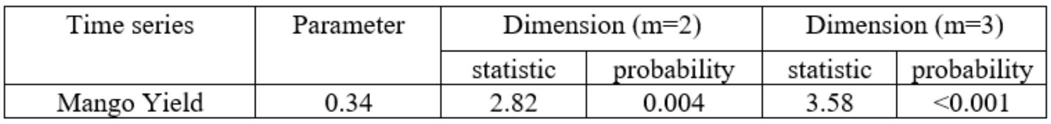
Time series	Activation function		Time delay	No. of hidden nodes	Total No. of Parameter s
	hidden Layer	output layer			
Mango Yield	Sigmoidal	Linear	2	4	17

Table 3: NLSVR Model specifications.



Time series	Kernel function	No. of SVs	C	γ	ε	K fold cross validation (K)	Cross Validation Error
Mango Yield	RBF	26	1.10	1.00	0.01	10	0.037

Table 4: Nonlinearity testing of ARIMA residuals by BDS test



Time series	Parameter	Dimension (m=2)		Dimension (m=3)	
		statistic	probability	statistic	probability
Mango Yield	0.34	2.82	0.004	3.58	<0.001

After fitting the ARIMA model, the residuals were subjected to the Brock-Dechert-Scheinkman (BDS) test to assess the presence of nonlinearity. The results of the BDS test, presented in Table 4, indicate that the residuals obtained from the ARIMA model are nonlinear and statistically significant. As outlined in the hybrid modeling framework, once the residual series is identified as nonlinear, it can be further modeled using nonlinear approaches to capture the remaining structure. In this study, the nonlinear models employed for modeling and forecasting the ARIMA residuals include the Time Delay Neural Network (TDNN) and the Nonlinear Support Vector Regression (NLSVR) models. Following the confirmation of nonlinearity in the ARIMA residuals, both TDNN and NLSVR were independently fitted to the residual series. The resulting forecasts from these models were then added to the forecasts obtained from the original ARIMA model, thereby forming two hybrid models: ARIMA-TDNN and ARIMA-NLSVR. The combined hybrid forecasts are expected to capture both the linear and nonlinear components of the yield time series. The forecasting performance of these hybrid models was evaluated for both the training period and the out-of-sample testing period, and the results are summarized in Tables 5 and 6, respectively.

Table 5: Comparison of forecasting performance of all models in training data set.

Criteria	**ARIMA**	**TDNN**	**NLSVR**	**ARIMA-TDNN**	**ARIMA-NLSVR**
MAPE	**3.83**	**2.89**	**2.81**	**1.98**	**1.73**

Table 6: Comparison of forecasting performance of all models in testing data set.

Year	**Actual**	**Forecast**

		ARIMA	**TDNN**	**NLSVR**	**ARIMA-TDNN**	**ARIMA-NLSVR**
2012	10.84	11.75	9.68	10.71	10.12	10.59
2013	10.04	11.15	10.14	10.73	10.62	10.44
2014	9.93	8.67	10.37	9.25	10.01	10.12
MAPE		**10.71**	**5.37**	**4.97**	**4.40**	**2.73**

As described in the hybrid methodology section, hybrid models offer distinct advantages over individual models by effectively capturing both linear and nonlinear structures present in time series data. Based on the lowest Mean Absolute Percentage Error (MAPE) values observed for both the training (Table 5) and testing (Table 6) data sets, it is evident that the hybrid model combining ARIMA and Nonlinear Support Vector Regression (ARIMA-NLSVR) outperformed all other models considered in the study. Both hybrid models—ARIMA-TDNN and ARIMA-NLSVR—showed improved performance over the standalone models, namely ARIMA, TDNN, and NLSVR. Among these, the ARIMA-NLSVR model delivered the most accurate forecasting results. This enhanced performance can be attributed to the hybrid methodology's ability to address both the linear and nonlinear components inherent in the mango yield time series of Karnataka.

Conclusion

The findings of this study highlight the superiority of hybrid time series modeling over single-model approaches for forecasting agricultural yield data. Given that the mango yield time series exhibited both linear and nonlinear patterns, the hybrid models demonstrated better accuracy in capturing the underlying structure. In particular, the ARIMA-NLSVR model showed consistent and superior forecasting accuracy across both the training and testing periods. The results underscore the practical value of hybrid approaches in time series forecasting, and future extensions of this work could explore the integration of other advanced machine learning techniques or varied autoregressive and moving average model orders to further enhance the predictive performance and robustness of the forecasting framework.

R codes to implement Hybrid TS models

```
nrow(available.packages())
rm(list=ls())
install.packages()
install.packages(c("forecast", "e1071", "tseries", "ggplot2", "fNonlinear", "lmtest"))
library(forecast)
library(e1071)
library(tseries)
library(ggplot2)
library(tidyverse)
library(fNonlinear)
library(lmtest)
g=read.table(file="rf.txt",header=T)
```

```
head(g)
dim(g)
Box.test(g$Rainfall)
rf1=read.table(file="rf1.txt",header=T)
head(rf1)
ggplot(data = rf1, aes(x = Month, y = Rainfall) )+ geom_line(color = "#00AFBB", size = 1) +
 labs(x = "Months", y = "Rainfall") + ggtitle("TS Plot of Monthly Rainfall Data")
bdsTest(g$Rainfall, m = 3, eps = NULL, title = NULL, description = NULL)
dim(g)
a1=g$Rainfall[1:1416]
a2=g$Rainfall[1417:1428]
Box.test(a1)
acf(a1)
pacf(a1)
############# ARIMA Fitting #########
m1=auto.arima(a1)
coeftest(m1)
accuracy(m1)
Box.test(m1$residuals)
fitted1=m1$fitted
write.csv(as.data.frame(fitted1), file="ARIMA_Fitted.csv")
f1=forecast(m1, h=12)
f11=data.frame(f1)
f12=f11$Point.Forecast
mse11=abs(a2-f12)^2
mse1=mean(mse11)
rmse1=sqrt(mse1)
rmse1
write.csv(as.data.frame(f12), file="ARIMA_Forecasted.csv")

################### ANN ##########
m2=nnetar(a1,6, P=1, 10, repeats=25, xreg=NULL, lambda=NULL, model=NULL,
subset=NULL, scale.inputs=TRUE,  maxit=150)
m2
accuracy(m2)
fitted2=m2$fitted
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
Box.test(m2$residuals)
f2=forecast(m2, h=12)
f21=data.frame(f2)
f22=f21$Point.Forecast
mse21=abs(a2-f22)^2
mse2=mean(mse21)
rmse2=sqrt(mse2)
rmse2
write.csv(as.data.frame(f22), file="ANN_Forecasted.csv")
m3=nnetar(a1)
accuracy(m3)
```

```
m3
fitted3=m3$fitted
f3=forecast(m3, h=12)
f31=data.frame(f3)
f32=f31$Point.Forecast
mse31=abs(a2-f32)^2
mse3=mean(mse31)
rmse3=sqrt(mse3)
rmse3
Box.test(m3$residuals)
write.csv(as.data.frame(fitted2), file="ANN_Fitted.csv")
write.csv(as.data.frame(f32), file="ANN_Forecasted.csv")

#################### SVR ##########
X1=g$Rainfall[1:1416]
Y1=g$Rainfall[2:1417]
X2=g$Rainfall[1416:1427]
Y2=g$Rainfall[1417:1428]
m4=svm(X1,Y1,degree = 3,cost = 45.69, nu=0.5,tolerance = 0.00001,epsilon = 0.00001)
summary(m4)
fitted4 <- fitted(m4)   ## Fitted values
mse41=abs(Y1-fitted4)^2
mse4=mean(mse41)
rmse4=sqrt(mse4)
rmse4
Box.test(m4$residuals)
s3=predict(m4,X2)
mse61=abs(Y2-s3)^2
mse6=mean(mse61)
rmse6=sqrt(mse6)
rmse6
############# ARIMA ###########
##########Significance Comparison ##########
########## For testing set ######
dm.test(m1$residuals, m2$residuals)
dm.test(m1$residuals, m3$residuals)
dm.test(m1$residuals, m4$residuals)
######## You have to do it for testing set also #####
########### Hybrid Modeling ##########
r1=m1$residuals
bdsTest(r1, m = 3, eps = NULL, title = NULL, description = NULL)
n1=nnetar(r1)
n1f=n1$fitted
c1=(m1$fitted)+n1f
c11=c1[32:1416]
a11=a1[32:1416]
mse51=abs(a11-c11)^2
mse5=mean(mse51)
```

```
rmse5=sqrt(mse5)
rmse5
############# Comparison###########
accuracy(m1)
accuracy(m2)
rmse4
rmse5
################### Fitted Plots ##########
rm(list=ls())
library(tidyverse)
library(readxl)
library(ggplot2)
Data1<-as.data.frame(read_excel("Fitted_Plot.xlsx", col_names = TRUE,sheet = "data"))
head(Data1)
Date <- seq(as.Date("2020/1/06"), as.Date("2020/06/30"), "day")
head(Data1)
RF=Data1$RF
Actual=Data1$Actual
Model1=Data1$Model1
Model2=Data1$Model2
Model3=Data1$Model3
Data2=data.frame(Date, RF, Actual,Model1, Model2, Model3)
df <- Data2 %>%
  select(Date, Actual, Model1, Model2, Model3) %>%
  gather(key = "Models", value = "RF", -Date)
tail(df)
p1<-ggplot(df, aes(x = Date, y = RF)) +
  geom_line(aes(color = Models), size = 1) + scale_x_date(date_labels = "%d/%b-%Y")+
labs(x = "Date", y = "RF")+ ggtitle("Actual v/s Fitted plot RF")+
  theme(plot.title = element_text(size = 11))
p1+geom_vline(xintercept = as.Date("2020-06-24"), color="blue4")
```

Suggested Readings

Box, G.E.P. and Jenkins, G. (1970). Time series analysis, Forecasting and control, Holden-Day, San Francisco, CA.

Brock, W.A., Dechert, W.D., Scheinkman, J.A, and lebaron, B. (1996). A test for independence based on the correlation dimension, *Econometric reviews*, 15:197-235.

Chen, K. Y. and Wang, C. H. (2007). Support vector regression with genetic algorithm in forecasting tourism demand. *Tourism Management*, 28:215-226.

Gujarati, D. N., Porter, D. C. and Gunasekar, S. (2013). Basic Econometrics (Fifth Edition), Tata McGraw-Hill Education Pvt. Ltd, ISBN 10: 0071333452 / ISBN 13: 9780071333450.

Haykin, S. (1999). Neural Networks: A Comprehensive Foundation, New York.

Jha, G. K. and Sinha, K. (2012) Time-delay neural networks for time series prediction: an application to the monthly wholesale price of oilseeds in India, *Neural Computing and Applications*, 24(3), 563-571

Kumar, T. L. M and Prajneshu. (2015) Development of Hybrid Models for Forecasting Time-Series Data Using Nonlinear SVR Enhanced by PSO. *Journal of Statistical Theory and Practice*. 9(4), 699-711.

Naveena, K., Rathod, S., Shukla, G. and Yogish, K.J. 2014. Forecasting of coconut production in India: A suitable time series model, International Journal of Agricultural Engineering, 7(1):190-193.

Naveena, K., Singh, S., Rathod, S., and Singh, A. 2017. Hybrid ARIMA-ANN Modelling for Forecasting the Price of Robusta Coffee in India. International Journal of Current Microbiology and Applied Sciences, 6(7): 1721-1726.

Naveena, K., Singh, S., Rathod, S., and Singh, A. 2017. Hybrid Time Series Modelling for Forecasting the Price of Washed Coffee (Arabica Plantation Coffee) in India. International Journal of Agriculture Sciences, 9(10): 4004-4007.

Rathod, S. and Mishra, G.C. (2018). Statistical Models for Forecasting Mango and Banana Yield of Karnataka, India. Journal of Agricultural Science and Technology. 20(4) July 2018.

Rathod, S., Singh, K, N., Paul, R.K., Meher, R.K., Mishra, G.C., Gurung, B., Ray, M. and Sinha, K. 2017. An Improved ARFIMA Model using Maximum Overlap Discrete Wavelet Transform (MODWT) and ANN for Forecasting Agricultural Commodity Price. Journal of the Indian Society of agricultural Statistics. 71(2): 103–111.

Ray, M., Rai, A., Ramasubramanian, V. and Singh, K.N. (2016). ARIMA-WNN Hybrid Model for Forecasting Wheat Yield Time-Series Data. *Journal of the Indian society of agricultural statistics*. 70(1): 63-70.

Vapnik, V., Golowich, S., and Smola, A. (1997). Support vector method for function approximation, regression estimation, and signal processing, In Mozer, M., Jordan, M and Petsche, T. (Eds) Advances in Neural Information Processing Systems, 9:281-287, Cambridge, MA, MIT Press.

Zhang, G.P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50, 159-175.

Ensemble Time Series Framework for Agricultural Price Forecasting

Kapil Choudhary
College of Agriculture, Sumerpur (Pali)- 306902, Agriculture University, Jodhpur
Email: kapiliasri@gmail.com

1. Introduction

Prices of agricultural commodities play a vital role in producers' incentives to produce and consumers' economic access to food, leading to a usual dilemma for policy planners. Accurate forecasts of agricultural commodity prices reflecting cumulative information held by different economic agents can play a crucial role in marketing strategy and investment decisions and offer suggestions for agricultural policy planning. However, the agricultural commodity market is influenced by several factors such as climate variability, including seasonality of production, the derived nature of demand, market imperfections, economic globalization, and a series of administrative regulations, making the price series extremely complex with nonlinearity, non-stationarity, and chaotic characteristics. All these complexities lead to the price prediction of agricultural commodities, an extremely challenging task.

Extensive investigation of literatures confirms abundant studies trying to tackle and analyse the complexities of price series for better forecasting. The models used in those studies can be categorised under statistical models and artificial intelligence (AI) models. Statistical models employed for agricultural price forecasting include models like ARIMA (Box et al., 2017) and its constituent models. However, due to the pre-assumed linearity and fixed temporal constraint among data, these statistical models did not meet the expected accuracy in predicting such nonlinear and complex price series.

Whereas AI models, with their great self-learning capabilities, have evolved as important and reliable means for the task. Various AI models being practiced for price forecasting include time-delay neural network (TDNN), wavelet neural network (WNN), support vector machine (SVM), extreme learning machine (ELM), etc. Although these techniques are established as effective measures for any time series prediction, AI techniques suffer from some limitations like problems of local minima, parameter sensitiveness, overfitting, the requirement of a large dataset for better training, etc. However, there is no simple, effective way to build and select a neural network. Thus a trial-and-error or cross-validation experiment is often adopted to find the best model. There are several limitations with this keep-the-best (KTB) approach in choosing a neural network model. First, the best selected model may not be a true optimal model because of choice of different factors of a network may lead to choose an alternative model. Second, neural networks are data-driven methods, so the selected best model may overfit the specific sample data.

To overcome these drawbacks, the potential of combining several neural network models to form an ensemble for forecasting has been examined. According to Hibon and Evgeniou (2005), an ensemble is more effective and stable than a single model (KTB). The basic idea of model combination is to use each model's unique feature to capture different patterns in the data. Here combined forecast may not be good always but less risky to forecast compared to the individual forecast method.

Combining different techniques to construct hybrid model has been preferred in the literature to take advantage of each method. Among hybrid models, decomposition-based models are important techniques in which the original complex series is first decomposed into subseries with specific patterns and then built forecasting models for each subseries. Traditionally, two component model-based techniques, additive and multiplicative decomposition models, have been used in the field of time series forecasting. These techniques decompose a time series into trend, cyclic, seasonal and irregular components. The additive method assumes that the components are orthogonal, whereas the multiplicative methods assume that the trend and seasonal components have a proportional relationship. To overcome these limitations, frequency domain analysis (FDA) based decomposition methods are being used. FDA methods have demonstrated better performance in dealing with the nonlinear, high-frequency time series data. Among FDA, although Fourier spectral analysis has provided a general method to analyze time-series data, there are some crucial restrictions of this transformation, i.e. the data must be linear and strictly periodic or stationary. As the degree of nonlinearity and nonstationary in a time series increases, Fourier decomposition's result often makes little physical sense. Another decomposition technique in this category is wavelet decomposition, which is an effective and widely used approach for analyzing the price series in both time and frequency domain (Antoniadis, 1997). Although wavelet analysis has many advantages in analyzing nonstationary time series data, it still suffers limitations like the prior selection of a filter function due to its non-adaptive nature. To overcome these limitations, the adaptive empirical mode decomposition (EMD) method for analysis of nonlinear and nonstationary time series through a divide and conquer concept was developed (Huang *et al.*, 1998). EMD method decomposes time-series data into several independent intrinsic mode functions (IMFs) with different amplitude and frequencies and a residue. However, EMD has proved to be a versatile technique in a wide range of applications. However, it suffers from a major limitation of mode mixing, which means that a single IMF contains sparsely distributed timescales, or similar timescales are broken down into different IMFs. To address the problem of EMD, ensemble empirical mode decomposition (EEMD) method was developed (Wu and Huang, 2009), which significantly reduces the chance of mode mixing and represents a substantial improvement over the original EMD.

The next step in developing hybrid methodology includes forecasting decomposed components. The artificial neural networks (ANNs) is used to forecast each component individually. One significant advantage of neural network models over other classes of nonlinear model is that ANNs are universal approximators which can approximate any continuous function with the desired accuracy in case of an adequate training dataset.

1. Decomposition Techniques

A decomposition technique is first used to decompose a complex time series into simpler or more meaningful components. These components are then modelled individually using a suitable forecasting model and the final forecasts are provided by ensembled the individual forecasts. Some of the most powerful decomposition techniques are explained below:

2.1 Empirical mode decomposition (EMD)

The empirical mode decomposition (EMD) technique has been proposed by N.E. Huang et al. (1998), with a view to analyze time-frequency distribution of nonlinear and nonstationary data. It is an adaptive decomposition with which any complicated series can be decomposed into its intrinsic mode functions (IMFs). IMFs have well-behaved Hilbert transforms, from which the instantaneous frequencies can be calculated. Thus, we can localize any event on the time as well as the frequency axis. The decomposition can also be viewed as an expansion of the data in terms of the IMFs. Then, these IMFs, based on and derived from the data, can serve as the basis of that expansion which can be linear or nonlinear as dictated by the data, and it is complete and almost orthogonal. Most important of all, it is adaptive. The principle of this basis construction is based on the physical time scales that characterize the oscillations of the phenomena. The local energy and the instantaneous frequency derived from the IMFs through the Hilbert transform can give us a full energy-frequency-time distribution of the data. Such a representation is designated as the Hilbert spectrum; it would be ideal for nonlinear and nonstationary data analysis.

After full decomposition we can get original series in such form

$$
\begin{aligned}
y_t &= c_t(1) + r_t(1) \\
&= c_t(1) + c_t(2) + r_t(2) \\
&= c_t(1) + c_t(2) + c_t(3) + r_t(3) \\
&\vdots \\
&= \sum_{j=1}^{N} c_t\,(j) + r_t(N)
\end{aligned}
$$

2.2 Ensemble EMD (EEMD)

Although the EMD shows great advantages in processing nonstationary and nonlinear energy prices, there is still a disadvantage of the traditional EMD algorithm, i.e. the decomposition results may be mode mixing, which means that a single IMF contains sparsely distributed timescales, or similar timescales are broken down into different IMFs.

In order to overcome this shortcoming, Wu and Huang (2009) propose the EEMD algorithm. The algorithm flow of EEM is as follows to find out j^{th}IMF:

i. Introduce a number of Gaussian white noises $n_t(i)$into data series y_t

$$\text{Where } n_t(i) \sim N(0, \sigma^2) \quad \text{So } y_t(i) = y_t + n_i(t)$$

ii. Conduct the EMD decomposition on $y_t(i)$ respectively, and obtaining a set of IMFs $c_t(ij)$and a residue $r_t(i)$

Where $c_t(ij)$is the j^{th} IMF decomposed by EMD after adding the Gaussian white noise for an i^{th} time.

iii. Repeat the above-mentioned steps. The ensemble average of corresponding IMFs is seen as the final decomposition result:

$$c_t(j) = \frac{1}{P}\sum_{i=1}^{P} c_t\,(ij)$$

where P is the ensemble size.

2. Time Delay Neural Network (TDNN) Model

The artificial neural network, inspired by the functioning of the human brain, consists of abstractions of processing elements in the form of mathematical functions called artificial neurons or nodes. The group of neurons operating together forms a layer of neurons and in general, three distinct layers are formed in a standard ANN model. These three layers namely the input layer, hidden layer and output layer are so interconnected with their nodes that each layer receives input from its preceding layer and passes the output to the subsequent layer. The input layer consists of the input series, the hidden layer performs the function of capturing the pattern and features from the data and finally, the output layer gives the final output as prediction or classification. The nodes of the hidden layer and output layer use a function called activation function which can be the same or different in these two layers. The activation function is used to introduce the nonlinearity in the model and also to limit the range of the output.

ANN models are regarded as data-driven, nonlinear and non-parametric statistical methods which capture the features and dependencies in a time series even when the relationship among data points is unknown. The ANN models need proper training at first and then the trained models are used for any application purpose. The information learned through training is stored in the nodes in the form of weights and biases which are used while producing the required outputs. Usually, a neural network is used effectively for pattern classification mainly for unstructured static data (not related by time constraint). But for temporal data, its training and pattern recognition is harder as the patterns evolve.

TDNN is a type of feed-forward neural network model that is being used for price series forecasting successfully. This neural network model builds a short-term memory, in particular, heteroassociative memory (Haykin, 2010), in its network through the use of time delays of a univariate time series to capture the temporal dimension of the series. To achieve this, a time series is first converted into a supervised learning format as a collection of samples. Each sample constitutes an input component (**X**) and an output component (**Y**). For example, if a time series contain N observations $y_1, y_2, y_3, \ldots, y_N$ and the model is to be made using p lagged values as input nodes, then to get one step ahead prediction the first sample will contain $y_1, y_2, y_3, \ldots, y_p$ as input components and y_{p+1} will be the output component. In this way, a set of N-p samples are generated each consisting input vector (**X**) and an output variable (Y). These samples are fed into an algorithm to learn the mapping function from the input to the output i.e. $Y = f(\boldsymbol{X})$. The task of the algorithm is to approximate or learn the real underlying mapping function so well that it can predict the output variables with maximum possible accuracy whenever new input data is given to it. The whole process of learning is called the training of the neural network.

Training of neural network consists of several iterations of the propagation of signals in both forward and backward direction. In forward propagation, the example sets having input and output values obtained using the supervised learning format are fed into the networks. The number of input values of the samples determines the number of input nodes of the network. The input values are then passed to the nodes of the hidden layer as their weighted sum where some affine transformations are done by nonlinear activation functions of the hidden nodes. Mathematically, the forward propagation can be expressed as:

$$y_{t+1} = \gamma\left(\sum_{m=1}^{q} \partial_m \psi\left(\sum_{j=0}^{p} w_{mj} y_{t-j} + b_m\right)\right)$$

where, $y_t, y_{t-1}, y_{t-2}, y_{t-3}, \ldots, y_{t-p}$ are the input patterns, w_{mj} is the synaptic weight between j^{th} input neuron and m^{th} hidden neuron, ∂_m is the weight between m^{th} hidden neuron and the output neuron, b_m is the bias, $\psi(.)$ and $\gamma(.)$ are the activation functions of hidden and output nodes, respectively and y_{t+1} is the output of the neuron at $t+1$ time step. In backward propagation, a loss function or cost function is computed by comparing the model output and desired output. This loss function is traversed backwards in the network computing its gradient or partial derivative with respect to all the weights in a particular order as a chain rule. The partial derivative with negative direction accounts for gradient descent which updates the synaptic weights of the network intending to minimize the loss function as small as possible. These several iterations of forward and backward propagation of information of all the examples or samples are called an epoch of the training. Thus, the training of the model requires the optimization of several hyperparameters like the number of hidden layers, the number of hidden nodes in the layers, type of activation functions etc. Tuning of these hyperparameters is problem dependent and determined through experimentation and grid search method on the given data.

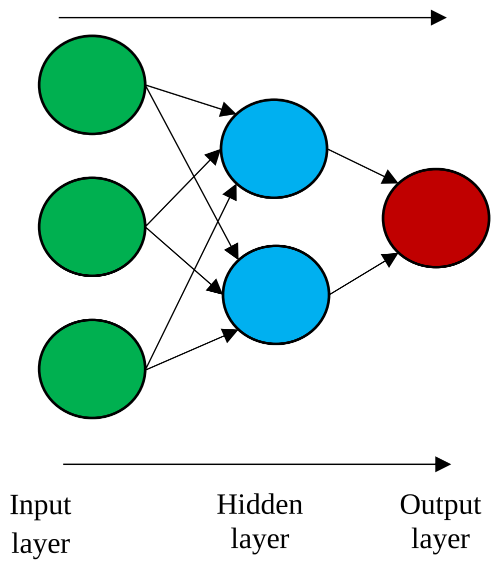


Fig.: Feed-forward Neural Network: Information propagates in the forward direction only i.e. from input to output;

3. Hybrid method

Given a price series y_t, the modelling procedure consists of three main steps:

Step 1: Decomposition. The price series y_t is decomposed into several meaningful and simpler components using any of the decomposition techniques. Different techniques yield different types of components based on their characteristics.

Step 2: Individual forecasts. Different forecasting techniques are used to model and forecast the individual components obtained after decomposition. The multi-step ahead forecasts of these components are obtained through iterative procedure using previous forecasted value as an input for forecasting the future value.

Step 3: Ensemble. The final step is to ensemble the individual forecasts of each components obtained by forecasting technique using addition to produce the final forecasts of the price series taken.

4. R package for practical implications

eemdTDNN
EMDTDNN (Empirical Mode Decomposition Based Time Delay Neural Network Model)
Description

The EMDTDNN function computes forecasted value with different forecasting evaluation criteria for Empirical Mode Decomposition based Time Delay Neural Network Model.

Usage

EMDTDNN(xt, stepahead = 10, s.num = 4L, num.sift = 50L)

Arguments

xt	Input univariate time series (ts) data.
stepahead	The forecast horizon.
s.num	Integer. Use the S number stopping criterion for the EMD procedure with the given values of S. That is, iterate until the number of extrema and zero crossings in the signal differ at most by one, and stay the same for S consecutive iterations.
num.sift	Number of siftings to find out IMFs.

Examples

data("Data_Maize")

EMDTDNN(Data_ Maize)

(a) EEMDTDNN **(**Ensemble Empirical Mode Decomposition Based Time Delay Neural Network Model)

Description

The EEMDTDNN function computes forecasted value with different forecasting evaluation criteria for Ensemble Empirical Mode Decomposition based Time Delay Neural Network Model.

Usage

EEMDTDNN(xt,stepahead=10,num.IMFs=emd_num_imfs(length(data)),s.num=4L,
num.sift=50L, ensem.size=250L, noise.st=0.2)

Arguments

xt	Input univariate time series (ts) data.
stepahead	The forecast horizon.
num.IMFs	Number of Intrinsic Mode Function (IMF) for input series.
s.num	Integer. Use the S number stopping criterion for the EMD procedure with the given values of S. That is, iterate until the number of extrema and zero crossings in the signal differ at most by one, and stay the same for S consecutive iterations.

num.sift	Number of siftings to find out IMFs.
ensem.size	Number of copies of the input signal to use as the ensemble.
noise.st	Standard deviation of the Gaussian random numbers used as additional noise.

Examples

```
Data("Data_Maize")

EEMDTDNN(Data_Maize)
```

References

Box, G.E.P. and Jenkins, G.M. (1970). Time Series analysis: Forecasting and control. Holden-Day, San Francisco, CA.

Choudhary, K., Jha, G. K., Kumar, R. R. and Jaiswal, R. (2021). *eemdTDNN :*EEMD and its variant based time-delay neural network model. https://doi.org/https://cran.rstudio.com/web/packages/eemdTDNN/eemdTDNN.pdf

Choudhary, Kapil, Jha, G. K., Kumar, R. R. and Mishra, D. C. (2019). Agricultural commodity price analysis using ensemble empirical mode decomposition: A case study of daily potato price series. *Indian Journal of Agricultural Sciences*, **89(5)**, 882–886.

Haykin, S. (2009). Neural Networks and Learning Machines, Person Education. In *Inc., Upper Saddle River, NJ, USA* (3rd ed., Vols. 1–3, Issue 6). PHI Learning.

Huang, N. E., Shen, Z., Long, S. R., Wu, M. C., Snin, H. H., Zheng, Q., Yen, N. C., Tung, C. C. and Liu, H. H. (1998). The empirical mode decomposition and the Hubert spectrum for nonlinear and nonstationary time series analysis. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **454,** 903–995. https://doi.org/10.1098/rspa.1998.0193

Jha, G. K. and Sinha, K. (2014). Time-delay neural networks for time series prediction: An application to the monthly wholesale price of oilseeds in India. *Neural Computing and Applications*, **24(3–4)**, 563–571. https://doi.org/10.1007/s00521-012-1264-z

Wu, Z. and Huang, N. E. (2009). Ensemble empirical mode decomposition: A noise-assisted data analysis method. *Advances in Adaptive Data Analysis*, **1(1)**, 1–41. https://doi.org/10.1142/S1793536909000047

ML Optimization: Particle Swarm Optimization

Santosha Rathod, Nobin Chandra Paul, Ponnaganti Navyasree, K. Ravi Kumar, Prabhat Kumar
ICAR-National Institute of Abiotic Stress Management, Baramati, Pune-413115
Email: santosha.rathod@icar.org.in

Introduction:

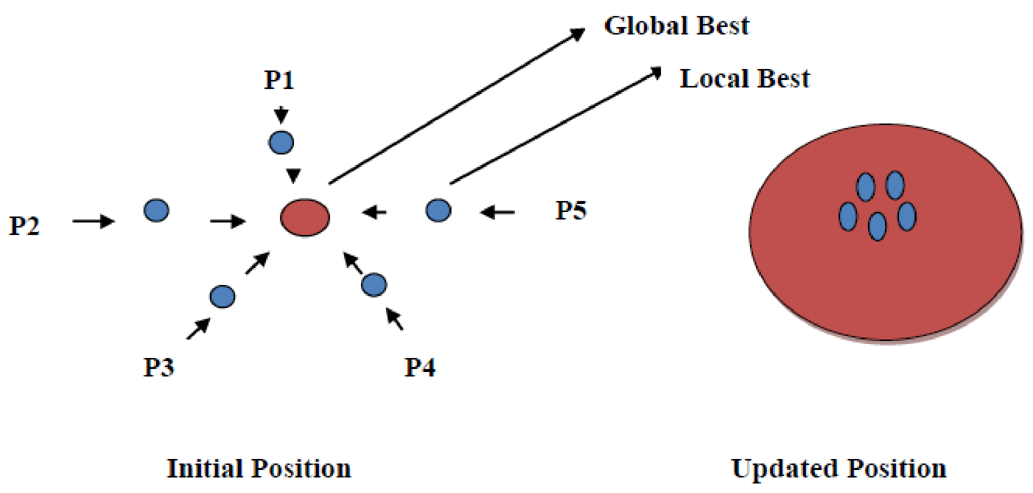
Particle Swarm Optimization (PSO) is a nature-inspired, evolutionary optimization technique developed to address computationally intensive or complex optimization problems. It is a stochastic, population-based optimization method that draws inspiration from the social behavior of organisms that move in groups or swarms, such as birds or fish. Introduced by James Kennedy and Russ Eberhart in 1995, PSO models the way these organisms communicate and adjust their paths based on both their own experience and the behavior of others in the group. Over the years, PSO has been successfully applied to a broad range of search and optimization problems by abstracting the natural dynamics of swarm intelligence.

PSO shares some conceptual similarities with other evolutionary algorithms like Genetic Algorithms (GA), in that both rely on a population of candidate solutions. However, the two differ in their philosophical approach. While evolutionary algorithms are rooted in the principle of survival of the fittest—emphasizing competition—PSO operates on a cooperative principle. In PSO, all individuals or particles in the swarm are allowed to survive and evolve. The success of one particle can influence and benefit others in the swarm, reflecting a collaborative learning mechanism.

The basic unit of PSO is a particle, which represents a candidate solution that flies through the search space in pursuit of the global optimum. Each particle updates its position based on its own best-known position and the best-known position among its neighbors. A swarm consists of *n* such particles, and they exchange information—either directly or indirectly—to guide their movement through the solution space. During each iteration, the position and velocity of each particle are updated, taking into account both the particle's own past performance and the performance of its neighbors. This dual influence allows the swarm to balance exploration and exploitation, ultimately converging towards the optimal solution.

PSO Vectors:

X vector: Current location (current position) of the particle in search space, **P vector (pbest):** Location of best solution found so far by the particle and **V vector:** Gradient (direction) for which particle will travel in, if undisturbed. All these vectors are continuously updated.



Let, $A \subset R^n$ be search space and the swarm is defined as a set $S = \{X_1, X_2, \ldots, X_M\}$ of M particles (candidate solution), where M is a user-defined parameter of the algorithm. Then i^{th} particle dimension of d is defined as $X_i = (X_{i1}, \ldots, X_{id})^T$, $i = 1,2, \ldots, M$. Each particle is a potential solution to a problem, characterized by three quantities: velocity $V_i = (V_{i1}, \ldots, V_{id})^T$, current position $X_i = (X_{i1}, \ldots, X_{id})^T$ and personal best position $pbest_i = (pbest_{i1}, \ldots, pbest_{id})^T$. Let, t denote current iteration and $gbest$ denote its global best position achieved so far by any of its particles. Initially, swarm is randomly dispersed within search space and random velocity is assigned to each particle. Particles interact with one another by sharing information to discover optimal solution. Each particle moves in the direction of its personal best position ($pbest$) and its global best position ($gbest$). To search optimal solution, each particle changes its velocity according to the cognitive and social parts given by:

$$V_{ij}(t+1) = w(t)V_{ij}(t) + c_1R_1\left[pbest_{ij}(t) - X_{ij}(t)\right] + c_2R_2\left[gbest_j(t) - X_{ij}(t)\right]$$

Where, $i = 1,2, \ldots, M$ and $j = 1,2, \ldots, d$. However, in case of swarm explosion effect, corresponding velocity component is restricted to following closest velocity bound:

$V_{ij}(t+1) = -V_{max}$ if $V_{ij}(t+1) < -V_{max}$

$= V_{max}$ If, $V_{ij}(t+1) > V_{max}$

After updating its velocity, each particle moves to a new potential solution by updating its position as follows

$X_{ij}(t+1) = X_{min}$ if $X_{ij}(t+1) < X_{min}$

$= X_{ij}(t)+\beta V_{ij}(t+1)$, if $X_{min} \leq X_{ij}(t+1) \leq X_{max}$

$= X_{max},$ if $X_{ij}(t+1) > X_{max}$

Where, $i = 1,2, \dots, M$; $j = 1,2, \dots, d$. In the above equations V_{ij} , X_{ij} and $pbest_{ij}$ are respectively velocity, current position and personal best position of particle i on the j^{th}dimension, and $gbest_j$ is the j^{th}dimension global best position achieved so far among all particles at iteration t. R_1 and $R_2$are random values, which are mutually independent and uniformly distributed over $[0,1]$, β is a constraint factor used to control velocity weight, whose value is usually set equal to 1. Positive constants c_1 and c_2 are usually called "acceleration factors". Factor c_1 is sometimes referred to as "cognitive" parameter, while c_1 is referred to as "social" parameter. Inertia weight at iteration t is $w(t)$ and is used to balance global exploration and local exploitation. This can be determined by:

$$w(t) = w_{up} - \left(w_{up} - w_{low}\right)t/T_{max}$$

Where,t is current iteration number, w_{up} and w_{low} are desirable lower and upper limits of w and T_{max} is maximum number of iterations.

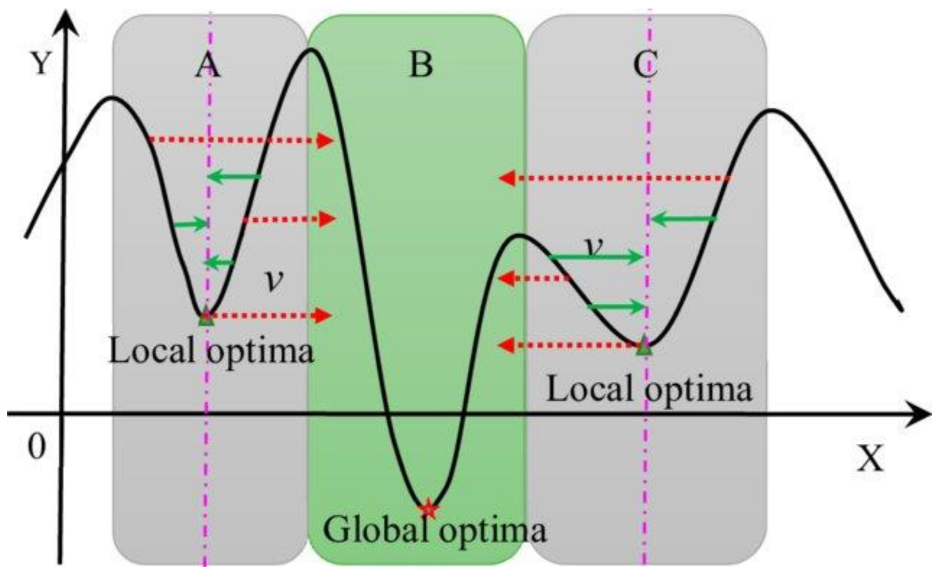
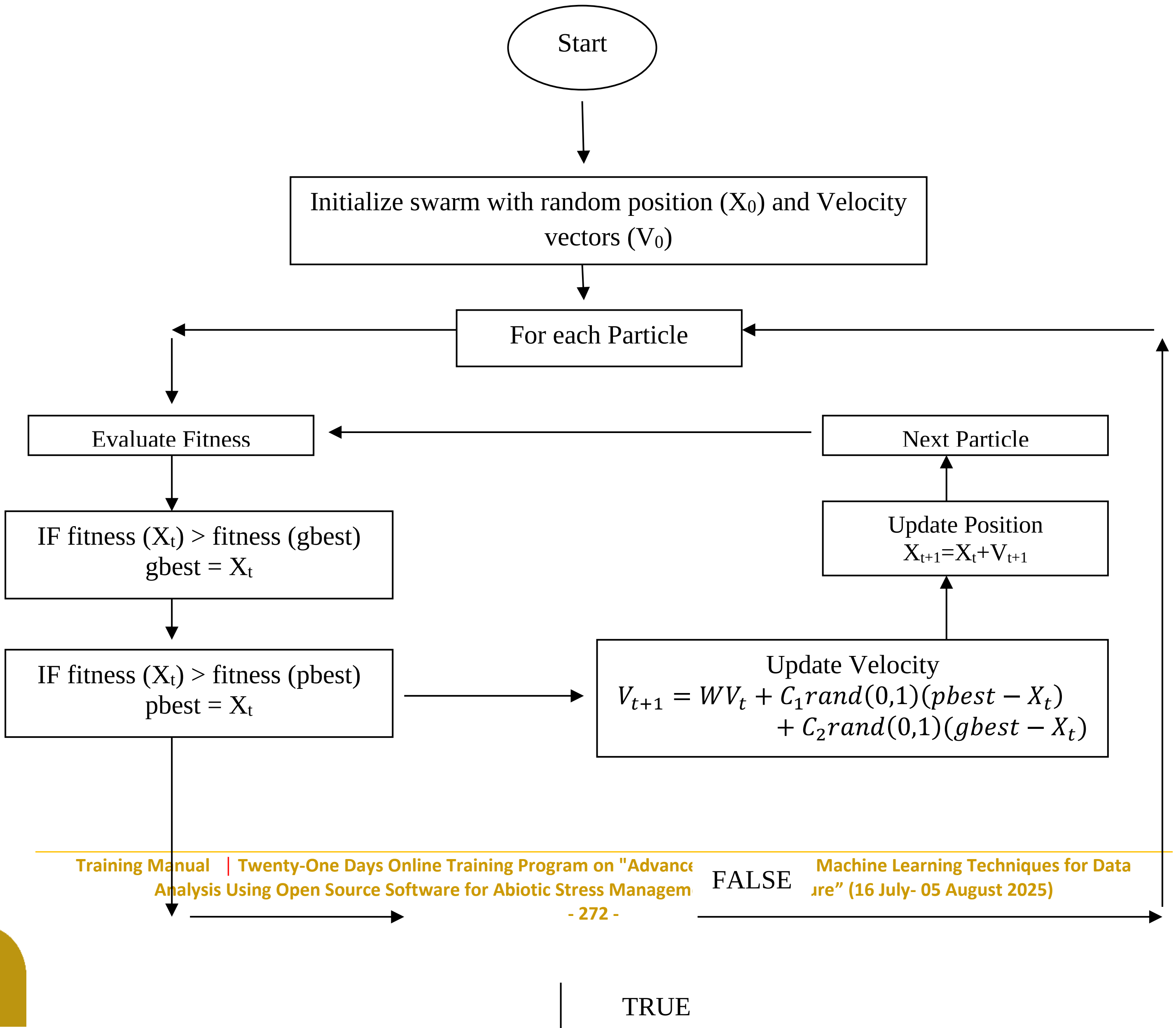
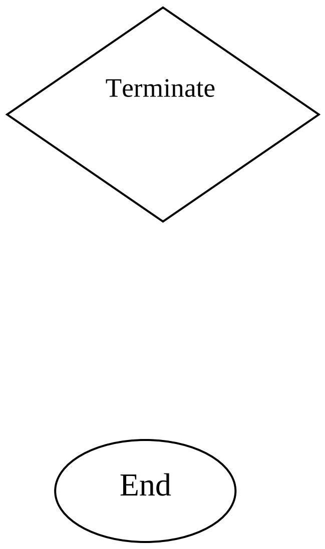


Fig.: Schematic diagram of particles' velocity.

Frame work of PSO:


Start
Initialize swarm with random position (X_0) and Velocity vectors (V_0)
For each Particle
Evaluate Fitness
Next Particle
IF fitness (X_t) > fitness (gbest) gbest = X_t
Update Position $X_{t+1}=X_t+V_{t+1}$
IF fitness (X_t) > fitness (pbest) pbest = X_t
Update Velocity $V_{t+1} = WV_t + C_1 rand(0,1)(pbest - X_t) + C_2 rand(0,1)(gbest - X_t)$
FALSE
TRUE

Training Manual | Twenty-One Days Online Training Program on "Advance Machine Learning Techniques for Data Analysis Using Open Source Software for Abiotic Stress Managem ure" (16 July- 05 August 2025)



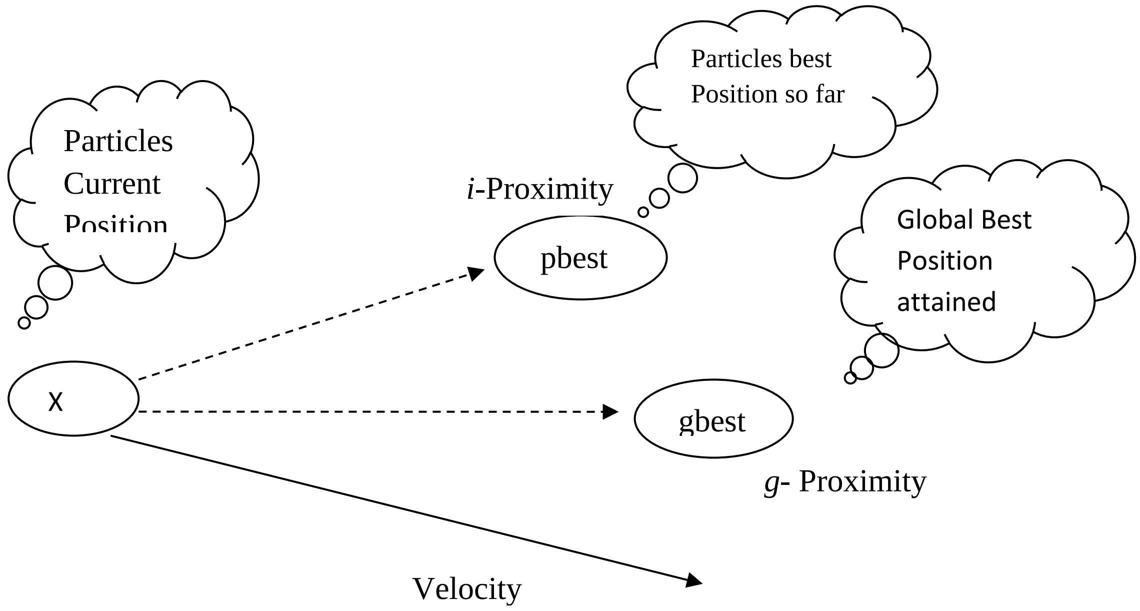
Algorithm Implementation:

Step 1 involves the initialization of parameters where each particle is randomly assigned a position and velocity within the defined search space. This random initialization helps ensure that the entire solution space is explored effectively.
Step 2 requires evaluating each particle’s current position using a predefined fitness function. This fitness value indicates how close a particle is to the optimal solution.
Step 3 is a comparison phase. First, the current fitness value of each particle is compared with its personal best fitness value (pbest). If the current value is better, then pbest is updated. Second, the fitness value of each particle is compared with the global best fitness value (gbest), and if it outperforms the previous gbest, the global best is updated accordingly.
Step 4 updates each particle’s velocity and position. The update process is influenced by both the particle’s personal experience (pbest) and the overall best experience of the swarm (gbest), incorporating stochastic elements to enhance exploration.
Step 5 checks whether the stopping condition has been met. This condition may be reaching a maximum number of iterations or achieving a desired fitness level. If not, the algorithm returns to Step 2 for another iteration.

The core idea behind PSO is to guide each particle towards its own best-known position and the best-known position found by the swarm, using a combination of deterministic and random components. The position update rule is straightforward: the new position $X_{i+1}$is obtained by adding the current velocity V_i to the current position X_i, that is, $X_{i+1} = X_i + V_i$. After moving, the particle re-evaluates its position, and if the new fitness value is better than its previous personal best, it updates its pbest accordingly.

Psychosocial compromise:

Each particle updates its new position by compromising its local best towards the global best as depicted schematically in the following diagram.


Particles Current Position
Particles best Position so far
Global Best Position attained
i-Proximity
pbest
X
gbest
g- Proximity
Velocity

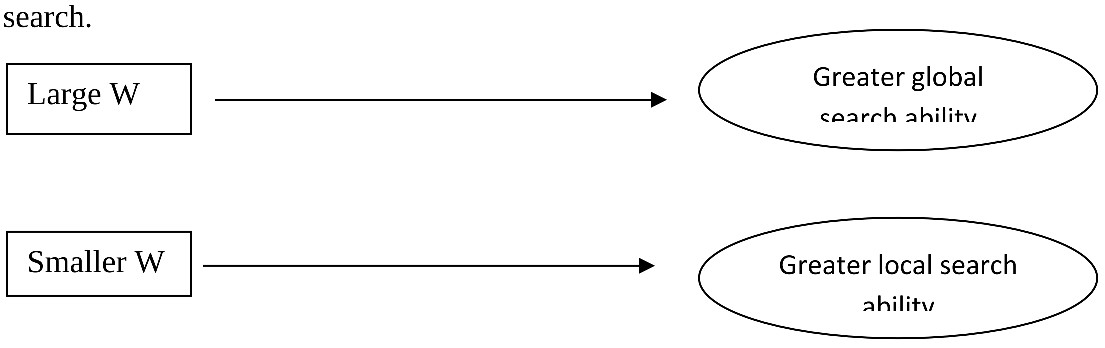
$$Position\ change\ is\ V_{t+1} = WV_t + C_1 rand(0,1)(pbest - X_t) + C_2 rand(0,1)(gbest - X_t)$$

User defined parameters:

Initial parameters such as swarm size, position of particles, velocity of particles and maximum number of iterations; and control parameters such as swarm size, inertial weight, acceleration coefficients C_1 and C_2 and number of iterations are very much important to begin with optimization algorithm. One has to define them in such a way that obtained parameter error should be less then target error.

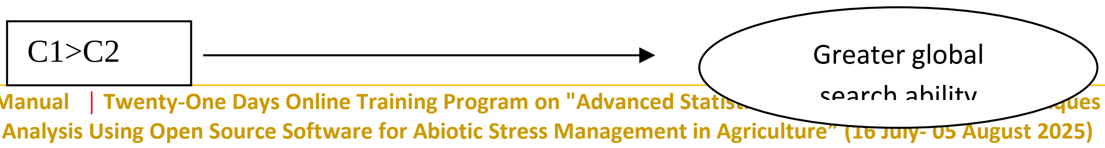
Innertial weight (W):

A large inertia weight (W) facilitates a global search while a small inertia weight facilitates a local search.



Acceleration coefficients:

An acceleration coefficient determines the inclination of search, greater the C1, greater will be the global search ability, greater the C2, greater will be the local search ability.



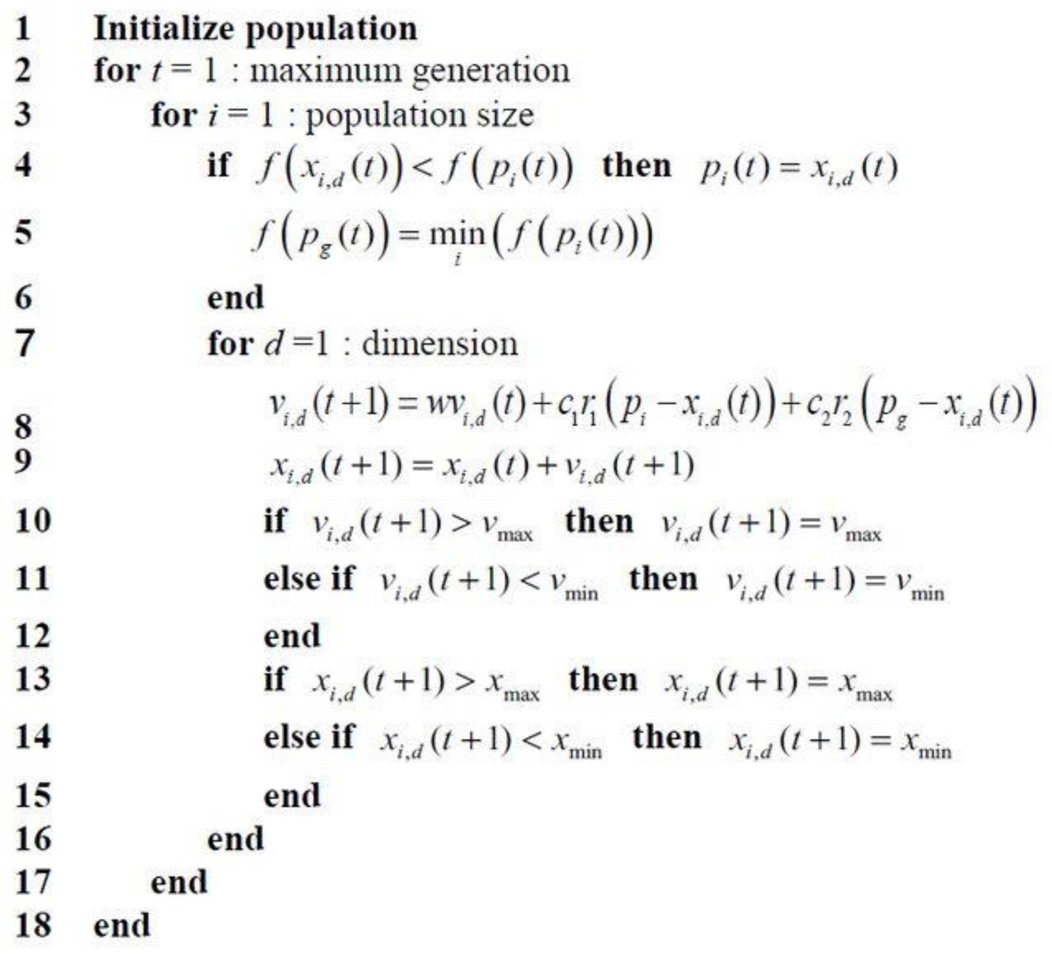
Training Manual | Twenty-One Days Online Training Program on "Advanced Statis ques for Data Analysis Using Open Source Software for Abiotic Stress Management in Agriculture" (16 July- 05 August 2025)

C2>C1
Greater local search ability

Pseudo code of PSO:

```
For each particle
{
    Initialize particle
}
Do until maximum iterations or minimum error criteria
{
    For each particle
    {
        Calculate Data fitness value
        If the fitness value is better than pBest
        {
            Set pBest = current fitness value
        }
        If pBest is better than gBest
        {
            Set gBest = pBest
        }
    }
        For each particle
    {
        Calculate particle Velocity
        Use gBest and Velocity to update particle Data
    }
}
```

Pseudocode in mathematical representation:



```
Initialize population
for $t = 1$ : maximum generation
    for $i = 1$ : population size
        if $f(x_{i,d}(t)) < f(p_i(t))$ then $p_i(t) = x_{i,d}(t)$
            $f(p_g(t)) = \min_i (f(p_i(t)))$
        end
        for $d = 1$ : dimension
            $v_{i,d}(t+1) = wv_{i,d}(t) + c_1 r_1 (p_i - x_{i,d}(t)) + c_2 r_2 (p_g - x_{i,d}(t))$
            $x_{i,d}(t+1) = x_{i,d}(t) + v_{i,d}(t+1)$
            if $v_{i,d}(t+1) > v_{\max}$ then $v_{i,d}(t+1) = v_{\max}$
            else if $v_{i,d}(t+1) < v_{\min}$ then $v_{i,d}(t+1) = v_{\min}$
            end
            if $x_{i,d}(t+1) > x_{\max}$ then $x_{i,d}(t+1) = x_{\max}$
            else if $x_{i,d}(t+1) < x_{\min}$ then $x_{i,d}(t+1) = x_{\min}$
            end
        end
    end
end
```

Numerical Example 1:

[Reference: Mohanty, P. (2018). NTPL online certification course on selected topics on decision modelling, Particle Swarm Optimization, IIT Khargapur. https://www.youtube.com/watch?v=uwXFnzWaCY0]

Consider a maximization problem for maximization of the function $f(x) = 1 + 2x - x^2$

Let us consider the control parameters W=0.70, C1=0.20, C2=0.60 and n=5 (Swarm particle). Consider, random numbers used for updating velocity of particle be

r1 = [0.4657, 0.8956, 0.3877, 0.4902, 0.5039]

r2 = [0.5319, 0.8185, 0.8331, 0.7677, 0.1708]

Note: We keep the random numbers fixed for all the iterations throughout and each random number is corresponding to each particle.

Initialization of swarm particles: We initialize fitness of all the particles as zeros;

Current position of all the particles as;

Cp(0)=10*[r1-0.5]

Cp(0)=10*{[0.4657, 0.8956, 0.3877, 0.4902, 0.5039]-0.5}

So, Cp(0)=[-0.3425, 3.9558, -1.128, -0.0981, 0.0385]

Note: Multiplied by 10 to initialize at least some particles to be >1 and subtracted 0.5 sides to generate both positive and negative random numbers.

Initialization of velocity:

V(0)=r2-0.5

V(0)={[0.5319, 0.8185, 0.8331, 0.7677, 0.1708]-0.5}

We get,

V(0)=[0.0319, 0.3185, 0.3331, 0.2677, -0.3292]

Note: one should see that velocity should not be too high or too low.

Current position and current fitness:

Iteration 1:

Current position (Cp) of each particle is what we initialize

Cp(1)= Cp(0)= [-0.3425, 3.9558, -1.128, -0.0981, 0.0385]

Current velocity V(1)=V(0)

=[0.0319, 0.3185, 0.3331, 0.2677, -0.3292]

Current fitness CF(1)= $f(Cp(1)) = 1 + 2Cp(1) - Cp(1)^2$

= [0.1976, -6.7368, -2.5061, 0.7942, 1.0755]

Note: $Cp(1)^2$ is obtained by squaring individual elements of Cp(1). As of now, we obtained current velocity, current position and current fitness.

Local best position (LBP)of each particle up to first iteration is just its current position.

LBP(1)=Cp(1)=[-0.3425, 3.9558, -1.128, -0.0981, 0.0385]

Local Best fitness of each particle up to iteration 1=current fitness of iteration 1

Local Best Fitness (LBF)

LBF(1)=CF(1)=[0.1976, -6.7368, -2.5061, 0.7942, 1.0755]

Global Best Fitness of iteration 1= Max (LBF(1));

GBF(1)=1.0755 $\rightarrow$ for 5th particle

Global Best Position of iteration 1

GBP(1)=Corresponding current position of 5th particle in cp(1)

=0.0385

Velocity of iteration 2

Velocity for next iteration

$$V_{t+1} = WV_t + C_1 rand(0,1)(LBP(i) - Cp(i)) + C_2 rand(0,1)(GBP(i) - Cp(i))$$

We have from iteration 1

V(1)=[0.0319, 0.3185, 0.03331, 0.2677, -0.3292]

For 1st particle: r_1=0.4657 ,r_2=0.5319, CP(1)=-0.3425, LBP(1)=-0.3425 and GBP(1)=0.0385

So, for the iteration 2, for the particle 1st: $V_2 = 0.7V(1) + 0.2 * rand(0,1)\big(LBP(i) - Cp(i)\big) + 0.6 * rand(0,1)(GBP(i) - Cp(i))$ =0.1439

Thus we have for iteration 2

V(2)=[0.1439, -1.7008, 0.8136, 0.2503, -0.2304]

Current position and current fitness

Current position for next iteration

$$Cp(i+1) = cp(i) + V(i+1)$$

WKT,

CP(1)=[-0.3425, 3.9558, -1.1228, -0.0981, 0.0385] & V(2)=[0.1439, -1.7008, 0.8136, 0.2503, -0.2304]

Hence, CP(2)=[-0.1986, 2.2550, -0.3092, 0.1522, -0.1919]

Current fitness for next iteration

CF(i)= $f(Cp(i)) = 1 + 2Cp(i) - Cp(i)^2$

Hence, CF(2)=[0.5634, 0.4250, 0.2860, 1.2812, 0.5794]

We know that Local Best Fitness is LBF(1)=[0.1976, -6.7368, -2.5061, 0.7942, 1.0755]

Hence,

LBF(2)=Max[CF(2), LBF(1)] = [0.5634,0.4250, 0.2860, 1.2812, 1.0755]

Local Best & Global Best

We have for iteration 2:

CP(2)=[-0.1986, 2.2550, -0.3092, 0.1522, -0.1919] and LBF(2)= [0.5634,0.4250, 0.2860, 1.2812, 1.0755]

Hence Global Best Fitness in iteration 2,

GBF(2)= Max(LBF(2))=1.2812

So, Global Best Position in iteration 2, GBP(2)= 0.1522(4th particle position in CP(2))

Local Best Position of each particle in iteration 2

CP(1)=[-0.3425, 3.9558, -1.1228, -0.0981, 0.0385] and LBF(1)=[0.1976, 0.4250, 0.2860, 1.2816, 0.5794]

So, LBP(2)= position w.r.t. LBF(2)=[-0.1976, 2.2550, -0.3092, 0.1522, 0.0385]

Current position is best for first 4 particle, but not for 5th last one is better

Summary: Iteration 1 & 2

Iteration	V(i) & CP (i)	CF(i) & LBF (i)	GBF(i)	LBP(i) & GBP(i)
1	V(1)=[0.0319, 0.3185, 0.03331, 0.2677, -0.3292] CP(1)=[-0.3425, 3.9558, -1.1228, -0.0981, 0.0385]	CF(1)=[0.1976, -6.7368, -2.5061, 0.7942, 1.0755] LBF(1)=[0.1976, -6.7368, -2.5061, 0.7942, 1.0755]	GBF(1) =1.0755	LBP(1)=[-0.3425, 3.9558, -1.1228, -0.0981, 0.0385] GBF(1)=0.0385
2	V(2)=[0.1439, -1.7008, 0.8136, 0.2503, -0.2304] CP(2)=[-0.1986, 2.2550, -0.3092, 0.1522, -0.1919]	CF(2)= [0.5634,0.4250, 0.2860, 1.2812, 0.5794] LBF(2)= [0.5634,0.4250,	GBF(2) =1.2812	LBP(2)=[-0.1986, 2.2550, -0.3092, 0.1522, 0.0385] GBP(2)=0.1522

		0.2860, 1.2812, 1.0755]		

Summary: Iteration 3 & 4

3	V(3)=[0.02127, -2.2232, 0.8001, 0.1752, -0.1120] CP(3)=[0.0141, 0.0318, 0.4909, 0.3274, -0.2944]	CF(3)=[1.0279,1.0625,1.7410, 1.5464, 0.3246] LBF(3)=[1.0279, 1.0625, 1.7410, 1.5464, 1.0755]	GBF(3)=1.7410	LBP(3)=[0.0141, 0.0318, 0.4909, 0.3274, 0.0385] GBP(3)=0.4909
4	V(4)=[0.3011, -1.3308, 0.5601, 0.1980, 0.0420] CP(4)=[0.3152, -1.2990, 1.0510, 0.5254, -0.2523]	CF(4)=[1.5312, -3.2861, 1.9974, 1.7740, 0.4317] LBF(4)=[1.5312, 1.0625, 1.9974, 1.7740, 1.0755]	GBF(4)=1.9974 (Best fitness)	LBP(4)=[0.3152, 0.0318, 1.0510, 0.5254, 0.0385] GBP(4)=1.0510 (Best position)

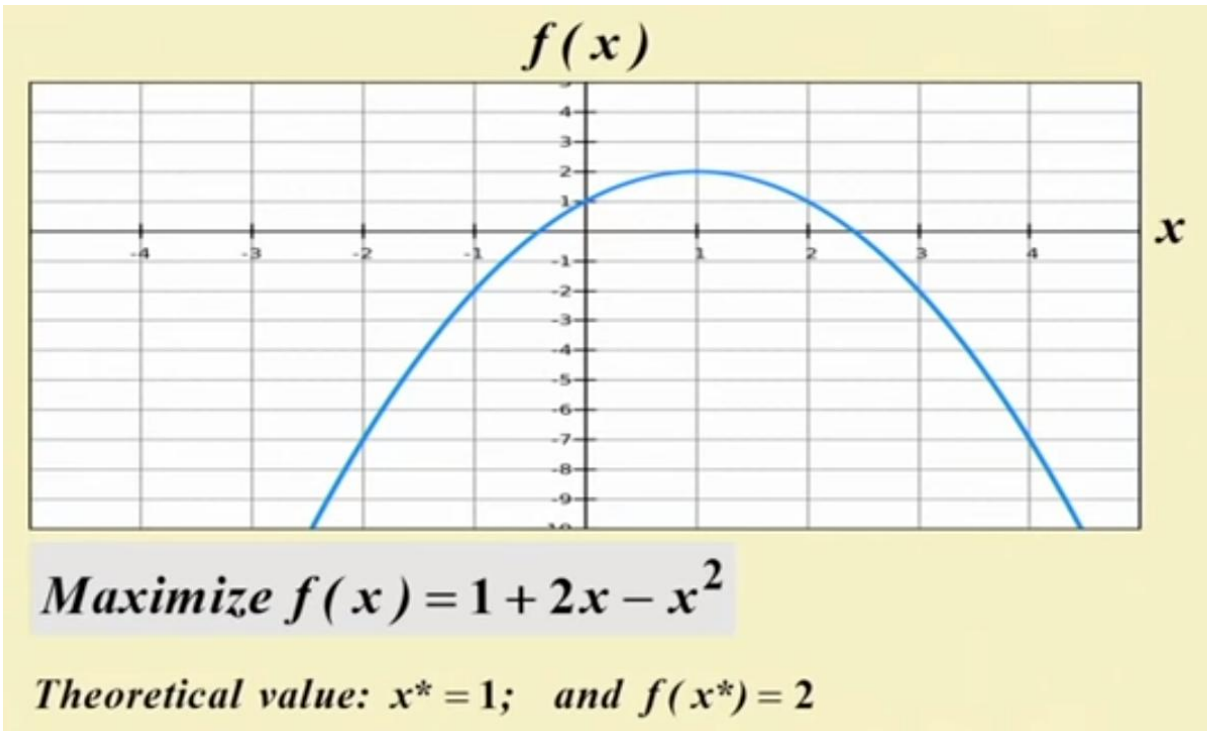
$$V_{t+1} = WV_t + C_1 rand(0,1)(pbest - X_t) + C_2 rand(0,1)(gbest - X_t)$$

$$Cp(i+1) = cp(i) + V(i+1)$$

$$LBF(i+1) = Max[CF(i+1), LBF(i)]$$

$$GBF(i) = Max[LBF(i)]$$

Final solution:



From iteration 4, we have, Global Best Position GBP(4)=1.0510 & Global Best Fitness

GBF(4)=1.9974 Hence the final solution obtained as x*=1.0510 and f(x*)=1.9974.

Numerical Example 2 – Robust Regression with Particle Swarm Optimisation

[Reference: Enrico Schumann. Robust Regression with Particle Swarm Optimisation. https://cran.r-project.org/web/packages/NMOF/vignettes/PSlms.pdf]

```
#R code for – Robust Regression with Particle Swarm Optimisation
install.packages("NMOF")
install.packages("MASS")
library("NMOF")
library("MASS")
set.seed(11223344)
createData <- function(n, p, constant = TRUE,
                       sigma = 2, oFrac = 0.1) {
  X <- array(rnorm(n * p), dim = c(n, p))
  if (constant)
    X[, 1L] <- 1L
  b <- rnorm(p)
  y <- X %*% b + rnorm(n)*0.5
  nO <- ceiling(oFrac*n)
  when <- sample.int(n, nO)
  X[when, -1L] <- X[when, -1L] + rnorm(nO, sd = sigma)
  list(X = X, y = y, outliers = when)
}
n <- 100L ## number of observations
```

```
p <- 10L ## number of regressors
constant <- TRUE; sigma <- 5; oFrac <- 0.1
h <- 75L ## ... or use something like floor((n+1)/2)
aux <- createData(n, p, constant, sigma, oFrac)
X <- aux$X; y <- aux$y
Data <- list(y = as.vector(y), X = X, h = h)
plot(Data)
plot(X,y)
plot(y, type="l")
par(bty = "n", las = 1, tck = 0.01, mar = c(4,4,1,1))
plot(X[ ,2L], type = "h", ylab = "X values", xlab = "observation")
lines(aux$outliers, X[aux$outliers ,2L], type = "p", pch = 21,
        col = "blue", bg = "blue")
OF <- function(param, Data) {
  X <- Data$X; y <- Data$y
  aux <- y - X %*% param
  aux <- aux * aux
  aux <- apply(aux, 2L, sort, partial = Data$h)
  colSums(aux[1:Data$h, ]) ## LTS
}
popsize <- 100L; generations <- 500L
ps <- list(min = rep(-10,p),
             max = rep( 10,p),
             c1 = 0.9,
             c2 = 0.9,
             iner = 0.9,
             initV = 1,
             nP = popsize,
             nG = generations,
             maxV = 5,
             loopOF = FALSE,
             printBar = FALSE,
             printDetail = FALSE)
system.time(solPS <- PSopt(OF = OF, algo = ps, Data = Data))
solPS <- PSopt(OF = OF, algo = ps, Data = Data)
solPS
```

Suggested Readings:

- Dai, H.-P.; Chen, D.-D.; Zheng, Z.-S. Effects of Random Values for Particle Swarm Optimization Algorithm. *Algorithms* 2018, *11*, 23. https://www.mdpi.com/1999-4893/11/2/23
- Enrico Schumann. Robust Regression with Particle Swarm Optimisation. https://cran.r-project.org/web/packages/NMOF/vignettes/PSlms.pdf]
- Gilli, M., D. Maringer and E. Schumann. (2011). *Numerical Methods and Optimization in Finance*. Elsevier.

- J. Kennedy, The particle swarm: social adaptation of knowledge, IEEE International Conference on Evolutionary Computation, 1997Indianapolis, IN. https://ieeexplore.ieee.org/document/592326
- Manfred Gilli, Dietmar Maringer, and Enrico Schumann. Numerical Methods and Optimization in Finance. Elsevier/Academic Press, 2011. URL http://enricoschumann.net/NMOF
- Mohanty, P. (2018). NTPL online certification course on selected topics on decision modelling, Particle Swarm Optimization, IIT Khargapur. https://www.youtube.com/watch?v=uwXFnzWaCY0]
- Soumya D. Mohanty (2012). Particle Swarm Optimization and regression analysis – I, Astronomical Review, 7:2, 29-35, DOI: 10.1080/21672857.2012.11519700.

ML Optimization: Spider Monkey Optimization for Agriculture

Prof. Dharavath Ramesh
Department of Computer Science and Engineering, Indian Institute of Technology Dhanbad
Email:drramesh@iitism.ac.in

1. Introduction

Agriculture is a vital sector that feeds the world's population, but it is increasingly under pressure from climate change, limited resources, and the need for sustainable practices. Traditional farming methods often struggle to efficiently allocate resources like water, fertilizers, and labor. Therefore, advanced computational approaches are needed to make agriculture smarter and more resilient. Among nature-inspired metaheuristic techniques, the Spider Monkey Optimization (SMO) algorithm has emerged as a flexible, adaptive method for tackling complex optimization tasks in agriculture. This document demonstrates how SMO can be practically applied to real-world agricultural scenarios, supporting farmers and policymakers in decision-making processes.
Agriculture faces numerous challenges in the modern world, including increasing productivity, optimizing resource usage, and ensuring sustainability. To address these issues, nature-inspired optimization algorithms have become popular due to their efficiency in solving complex problems. One such algorithm is the Spider Monkey Optimization (SMO) algorithm, inspired by the social behavior of spider monkeys. This document explores how SMO can be applied in agriculture through a detailed case study.

2. What is Spider Monkey Optimization (SMO)?

Spider Monkey Optimization is a swarm intelligence-based algorithm inspired by the fission-fusion social structure of spider monkeys. Spider monkeys dynamically split and merge their groups to forage for food efficiently. This behavior is modeled mathematically to solve complex optimization problems by searching large solution spaces effectively.

Spider Monkey Optimization (SMO) is a stochastic technique based on the social behavior of spider monkeys. This methodology provides a fascinating research opportunity in the field of optimization. The algorithm imitates the foraging behavior of spider monkeys that has been identified as a Fission-Fusion Social Structure (FFSS) based animal. SMO is similar to other population-based algorithms where each SM represents a potential solution for the considered problem. The working of SMO consists of four steps. Initially, in the first step, the group of spider monkeys starts food foraging and analyzes the distance from the food. Second, the group member updates their position based on the distance from the food and again evaluates the distance from food. Third, the local leader updates its best position within the group. If the best position is not updated within the defined threshold value, then all the members of the local groups start food foraging in other directions. In the last step, the ever-best position of the global leader is updated.

Furthermore, the group splits into smaller size subgroup to avoid stagnation. All these aforementioned steps are repeated several times until the desired result is achieved. SMO introduces two important control variables named LocalLeaderLimit (LLL) and GlobalLeaderLimit(GLL) to avoid stagnation in the local leader and the global leader phase. SMO is inspired by the intelligent foraging behavior of animals and follows the principle of division of labor property and self-organization. Modelling the SMO problem for workflow scheduling is a two-step process. The First step involves the identification of search space and the representation of the solution, i.e., how the defined problem is encoded. The second step defines the fitness function which is used to measure the quality of the solution.

In order to define the encoding of a solution, it is required to initialize the population of spider monkeys randomly. Afterwards, in each iteration, the fitness value of each spider monkey is evaluated using fitness function defined as optimization and scheduling constraints. Spider monkey repeatedly updates their position based on the LocalLeader, Local group members, and GlobalLeader experience to achieve the best fitness value. These steps are repeated until the algorithm attains the desired output.

Key Features:

- Decentralized decision-making.
- Dynamic grouping and regrouping.
- Balance between exploration and exploitation.
- Pest control strategy optimization.
- Yield prediction models.

3. SMO Algorithm Steps

✓Initialization: Define population size, group size, and objective function.
✓Local Leader Phase: Individuals follow their local leader to exploit the search space.
✓Global Leader Phase: Groups are influenced by a global leader to explore new areas.
✓Local Leader Learning Phase: Local leaders are updated based on the group's performance.
✓Global Leader Learning Phase: Global leader changes if needed.
✓Decision to Split or Merge: Groups split or merge based on performance and diversity.

(i) Initialization:

- Initialize population of spider monkeys (possible solutions).
- Set parameters:
- Number of groups,
- Number of spider monkeys per group,
- Maximum number of iterations,
- Probability of local leader and global leader learning.

(ii) Fitness Evaluation:

• Evaluate each spider monkey's position using the **objective function** (e.g., prediction error for crop yield).

(iii) Identify Leaders:

• Global Leader: The spider monkey with the best fitness in the whole population.

Local Leaders: The best monkey in each group.

(iv) Update Position:

Each spider monkey updates its position using:

$$X_i^{t+1} = X_i^t + r_1(LL_k^t - |X_i^t|) + r_2(GL^t - |X_i^t|)$$

- X_i^t: Current position.
- LL_k^t: Local leader's position of group k.
- GL^t: Global leader's position.
- r_1, r_2: Random numbers between 0 and 1.

(v) Local Leader Phase:

• If a group doesn't improve for a set number of iterations, reinitialize that group's members or make them explore new areas.

(vi) Global Leader Phase:

• If the global leader doesn't improve for a set number of iterations, increase the number of groups (fission) to enhance exploration.

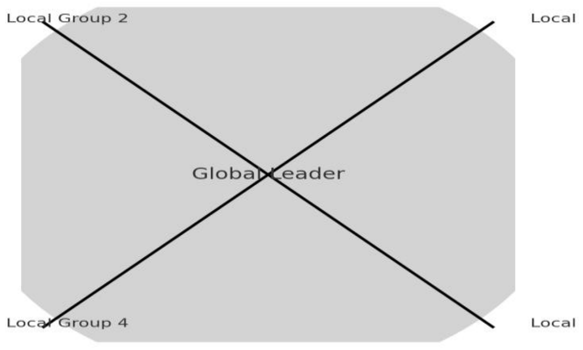
(vii) Merge (fusion):

• If exploration stagnates or the maximum number of groups is reached, groups may merge to share information.

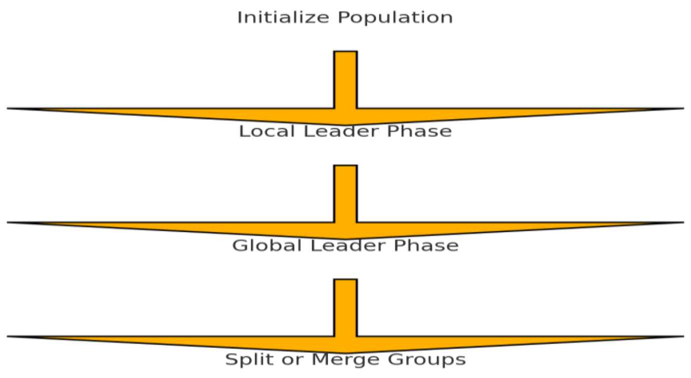
(viii) Termination:

• Repeat steps ii–vii until:
• Maximum iterations reached, or
• Desired fitness achieved.

SMO Process Flow:



SMO Social Structure:



4. Case Study: SMO for Crop Yield Prediction

Problem Statement: Predict wheat yield based on:

- Rainfall,
- Temperature,
- Soil moisture,
- Fertilizer use.

Objective: Minimize prediction error.

Step-by-Step Explanation

Initialization:

- Suppose we have 30 spider monkeys (solutions).
- Each monkey represents a possible set of model parameters for a regression model (like SVR, ANN, or even coefficients in a custom yield model).

Fitness Function:

- Use Mean Squared Error (MSE) between actual and predicted yield.
  Leaders:
- Identify the monkey with the lowest MSE globally and the best in each subgroup.

Update:

- Each monkey adjusts its model parameters based on leaders.

earning Phases:

- If a local leader's group doesn't improve, its members search wider.
- If the global best stagnates, new groups split off to explore different parameter regions.

Stopping:

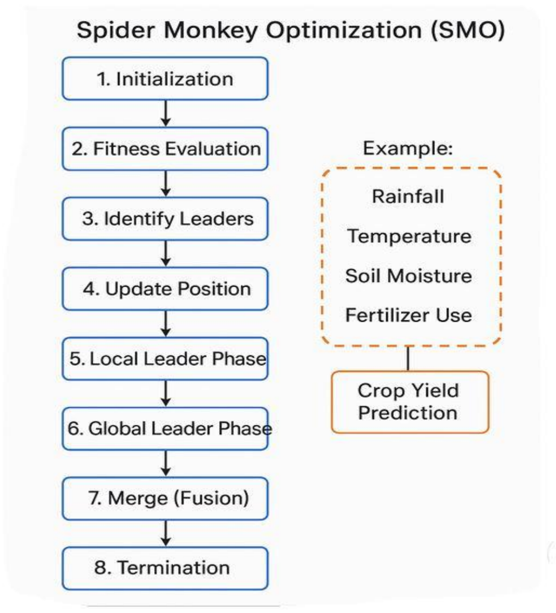
- Stop if the minimum MSE is acceptable or after, say, 200 iterations.

Result:
The best monkey's parameters are used to make accurate crop yield predictions.

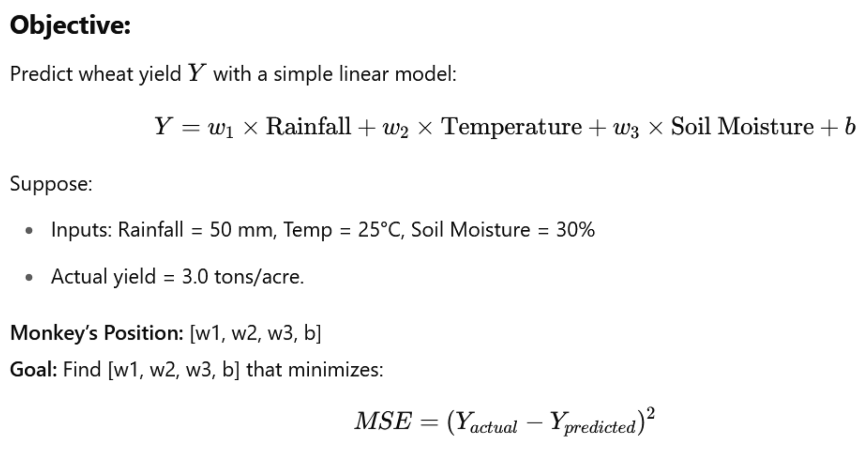
Key Benefit

SMO adaptively balances **exploration (global search)** and **exploitation (local refinement)**, making it well-suited for optimizing complex, nonlinear models like those used in **precision agriculture**.

SMO Flowchart:



Numerical Example:



Objective:

Predict wheat yield Y with a simple linear model:

$$Y = w_1 \times \text{Rainfall} + w_2 \times \text{Temperature} + w_3 \times \text{Soil Moisture} + b$$

Suppose:

- Inputs: Rainfall = 50 mm, Temp = 25°C, Soil Moisture = 30%
- Actual yield = 3.0 tons/acre.

Monkey's Position: [w1, w2, w3, b]
Goal: Find [w1, w2, w3, b] that minimizes:

$$MSE = (Y_{actual} - Y_{predicted})^2$$

Example:

- Monkey A: [0.02, 0.05, 0.03, 0.5]
- Predicted Y = 0.02×50 + 0.05×25 + 0.03×30 + 0.5

= 1.0 + 1.25 + 0.9 + 0.5 = 3.65

MSE = $(3.0 - 3.65)^2 = 0.4225$

Each monkey represents different weights → best one has the lowest MSE.

Simple Python code snippet:

```
import numpy as np

# Initialize
num_monkeys = 10
num_params = 4  # w1, w2, w3, b
max_iter = 50

# Example input and output
X = np.array([50, 25, 30])
Y_actual = 3.0

# Initialize monkey population randomly
population = np.random.uniform(-1, 1, (num_monkeys, num_params))

# Fitness function: MSE
def fitness(monkey):
    Y_pred = np.dot(monkey[:3], X) + monkey[3]
    return (Y_actual - Y_pred) ** 2

for iteration in range(max_iter):
    fitness_vals = np.array([fitness(m) for m in population])
    global_leader = population[np.argmin(fitness_vals)]
    for i in range(num_monkeys):
        r1, r2 = np.random.rand(), np.random.rand()
        population[i] += r1 * (global_leader - np.abs(population[i]))

    # Optional: Local leader logic can be added here

best_monkey = population[np.argmin([fitness(m) for m in population])]
print("Best weights found:", best_monkey)
```

5. Advantages of Using SMO in Agriculture

- Can adapt to real-time data inputs.
- Handles uncertainty in environmental parameters.

- Reduces operational costs by optimizing inputs.
- Improves sustainability and resource management.

Spider Monkey Optimization (SMO) offers several distinct advantages when applied to agriculture. Its nature-inspired design makes it highly effective for handling complex, nonlinear agricultural problems where multiple factors such as soil conditions, weather, pests, and crop genetics interact dynamically. The unique fission-fusion social structure of SMO enables an adaptive balance between exploration and exploitation, helping it avoid local optima and find better solutions for tasks like crop yield prediction, irrigation scheduling, fertilizer application, and precision pest management. SMO can be easily integrated with data-driven models, including machine learning frameworks, making it suitable for optimizing parameters in predictive analytics and decision-support systems in smart farming. Its scalability allows it to handle large datasets from IoT sensors and remote sensing technologies, which are increasingly common in modern precision agriculture. Moreover, SMO is robust against uncertainties such as unpredictable weather or market fluctuations, thanks to its collective learning behavior that mimics real-life social decision-making. Lastly, SMO is relatively easy to implement and adapt, and can be hybridized with other metaheuristic techniques like Particle Swarm Optimization (PSO) or Genetic Algorithms (GA) to enhance its performance for specific agricultural applications.

6. **Challenges and Future Scope**

While Spider Monkey Optimization (SMO) shows great promise for diverse agricultural applications, its practical deployment also faces certain challenges. One key challenge is the need for high-quality, real-time agricultural data, as inaccurate or sparse data can limit the algorithm's effectiveness and lead to unreliable predictions or suboptimal decisions. Additionally, fine-tuning SMO's parameters — such as group sizes, learning probabilities, and stopping criteria — can be complex and may require domain expertise to adapt the algorithm to different crops, regions, and seasons. Computational cost can be significant for large-scale problems, especially when SMO is combined with high-dimensional models or real-time IoT sensor networks. Another challenge lies in the interpretability of the solutions; farmers and stakeholders may find it difficult to trust black-box optimization outputs without clear explanations or user-friendly interfaces.

Despite these challenges, the future scope for SMO in agriculture is highly encouraging. Advances in precision agriculture, remote sensing, and IoT are creating rich, real-time data streams that can feed SMO-driven decision systems, improving their accuracy and adaptability. Hybrid approaches that combine SMO with other metaheuristic algorithms, machine learning models, or domain-specific constraints could deliver even better performance and robustness. Integration with digital twins of farms, climate-smart farming systems, and autonomous machinery is another promising direction, enabling SMO to optimize dynamic operations in real time. Furthermore, user-friendly decision support tools, mobile applications, and cloud-based platforms can help translate SMO's complex computations into actionable insights for farmers and policymakers. Continued research into explainable optimization, scalable implementations, and cross-disciplinary collaboration will be key to unlocking SMO’s full potential in achieving sustainable, data-driven agriculture.

7. Conclusion

In summary, Spider Monkey Optimization (SMO) stands out as a promising nature-inspired metaheuristic algorithm for addressing the multifaceted challenges of modern agriculture. By mimicking the adaptive and cooperative social behavior of spider monkeys, SMO effectively balances global exploration and local exploitation, making it highly suitable for solving complex, nonlinear problems such as crop yield prediction, irrigation scheduling, and resource optimization. Its ability to integrate with machine learning models and process large volumes of data from IoT and remote sensing technologies positions it as a valuable tool for smart, data-driven farming practices.

However, successful implementation of SMO in agricultural contexts also depends on overcoming practical challenges such as data availability, parameter tuning complexity, computational demands, and the interpretability of results for end-users. Addressing these challenges through hybrid algorithm designs, user-friendly decision-support systems, and scalable digital infrastructure can unlock SMO’s full potential.

Looking ahead, the integration of SMO with emerging technologies like digital twins, autonomous farming equipment, and explainable AI offers exciting opportunities to enhance sustainability, resilience, and efficiency in agriculture. With continued research, collaboration, and technological advancements, Spider Monkey Optimization could play a significant role in shaping the future of precision and climate-smart farming, ultimately contributing to global food security and sustainable development goals.

"It is better to live your own destiny imperfectly than to live an imitation of somebody else's life with perfection.

भाकृअनुप
ICAR

राअस्ट्रैप्रसं
NIASM

Twenty-One Day Online Training Program
on

Advanced Statistical and Machine Learning Techniques for Data Analysis Using Open-Source Software for Abiotic Stress Management in Agriculture

16 July – 5 August 2025

Chief Patron
Dr. K Sammi Reddy, Director, ICAR-NIASM

Patron
Dr. Nitin P Kurade, Head, SSSPS, ICAR-NIASM

Course Directors

Dr. Santosha Rathod
Dr. Nobin Chandra Paul
Ms. Ponnaganti Navyasree
Mr. K Ravi Kumar

Google
Earth Engine

Organised by:

School of Social Science and Policy Support
ICAR-National Institute of Abiotic Stress Management Baramati, Maharashtra - 413115

About ICAR-NIASM

ICAR-NIASM is the premier institute of ICAR established in 2009 at Baramati. The institute aims at exploring the avenues for the management of abiotic stresses affecting the very sustainability of national food production systems. Besides focusing on developing climate resilient solutions through cutting-edge technologies for managing abiotic stresses, NIASM also aims to enhance scientific capacity through multidisciplinary research and capacity building programs.

About the Training Program

This 21-day online training program offers hands-on experience in advanced statistical, machine learning, and deep learning techniques for analyzing agricultural data. The training is not limited to abiotic stress management; it is applicable across all research disciplines where data analysis plays a critical role. Participants will work with large datasets using open-source tools such as R, Python, QGIS, VassarStats, and BlueSky Statistics through practical, application-oriented sessions.

Key Objectives

The training program aims to:

- Train the participants in multivariate statistics, AI- ML, and deep learning, agroecological modeling tools, remote sensing &GIS
- Provide hands-on experience with open-source software
- Enable independent application of these techniques in research work

Course Content

The program combines theoretical foundations with hands-on practical sessions, enabling participants to apply these techniques to their own datasets efficiently.

Module 1: Software Tools for Data Analysis

- Pre-training session on Installation guide to R/Python/other tools
- Introduction to R
- Introduction to Python
- Introduction to Bluesky Statistics & VassarStats
- Data Visualization in R
- R Shiny and R packages

Module 2: Regression & Multivariate Statistical Methods

- Regression Analysis
- Regression for Categorical Data
- Nonlinear Growth Models
- Regularization Techniques in Regression Models
- Panel Data Regression
- Non-Parametric Analysis
- Data Classificatory Techniques (CA, DA)
- Data Reduction Techniques (FA, PCA)

Module 3: Design of Experiments & Statistical Genetics

- Analysis of Complete and Incomplete Block Designs
- Analysis of Incomplete Block Designs
- Analysis of Groups of Experiments (GOE)
- Response Surface Methodology
- Generation Mean Analysis
- Mating Designs
- Path Analysis
- Stability Analysis
- QTL Analysis
- Transcriptomic Analysis
- Genome Wide Association Studies (GWAS)
- Genomic Selection
- Selection Index
- Meta-QTL Analysis
- Meta-Genomics

Module 4: Machine Learning & Deep Learning Techniques

- Introduction to Machine Learning
- k-nearest neighbor (KNN)
- Artificial Neural Network
- Support Vector Machine
- CART and Decision Tree
- Random Forest Regression
- Extreme Learning Machine
- XGBoost
- Deep Learning: RNN, GRU, CNN, LSTM, Transformer DL
- ML Optimization Techniques
- Yield Forecasting using AI

Module 5: Time Series & Forecasting Methods

- Trend Analysis
- Time Series Analysis
- ARCH Family of Models
- Bayesian Forecasting Models
- Count Time Series Models
- Spatiotemporal Time Series Modelling
- Hybrid Modelling
- Ensemble Modelling
- VAR and Cointegration Analysis

Module 6: Spatial & Environmental Data Analysis

- Introduction to RS & GIS
- Introduction to QGIS
- Introduction to Google Earth Engine
- Spatial Interpolation Techniques
- Introduction to Sampling & Spatial Sampling Strategy
- Application of ML in RS & GIS (ASIS portal)
- Application of UAVs in agricultural data modelling

Module 7: Agro-Ecological Modelling

- Biomass Modelling &Carbon Sequestration using Allometric Models
- CMIP6 GCM Models
- Crop Simulation Modelling (DSSAT and APSIM)
- High-throughput Plant Phenotyping
- Assessment of Extreme Weathers

Module 8: Emerging & Interdisciplinary Topics

- Importance of Data Science in Agricultural Research
- Meta Analysis
- AHP and Grey Model: Technology Forecasting
- Markov Chain Analysis
- Social Network Analysis
- Bibliometric Analysis
- Economic Index Development
- Impact Assessment Modelling, Trend Impact Analysis
- Statistical Modelling in Disease Epidemiology
- Fuzzy Regression Analysis

Expected Learning Outcomes

- By completing this program, participants will master in advanced statistical, machine learning, and deep learning techniques to analyze complex agricultural and environmental datasets.
- They will gain hands-on experience with open-source tools (R, Python, QGIS and others) for data analysis, image processing, and designing efficient workflows to address real-world abiotic stress challenges.

Who Can Apply?

- Researchers and scientists from agriculture, climate science, environmental studies, and allied fields
- Data analysts looking for transition from classical statistics to ML/DL approaches
- Academicians and students seeking proficiency in open-source statistical and geospatial tools

Registration Fee

- ₹ 1000/- for students and research scholars
- ₹ 2000/- for scientists, researchers, faculty members, and working professionals from public organizations
- ₹ 5000/- for participants from private industries

ICAR UNIT NIASM BARAMATI

SCAN & PAY

UPI ID: icarniasmbmt@sbi

Bank Account Details

Account Holder Name: ICAR UNIT-NIASM, Baramati
Account Number: 30862846914
Name of the Bank: State Bank of India
Branch Address: Afzalpurkar Building, Bhigwan Road,
Baramati, Maharashtra-413102
IFSC: SBIN0000321
UPI Code : icarniasmbmt@sbi

Important Dates

- Last Date for Receipt of Applications: 30th June 2025
- Information to Selected Candidate: 2nd July 2025

Registration Link: https://forms.gle/5cMmTxnS19DvWoc48

Contact for Registration Related Queries

Dr. Santosha Rathod
Senior Scientist (Agricultural Statistics)
School of Social Science and Policy Support
ICAR-NIASM, Baramati
Mob: 9900912188

Dr. Nobin Chandra Paul
Scientist (Agricultural Statistics)
School of Social Science and Policy Support
ICAR-NIASM, Baramati
Mob: 8851954194

Ms. Navyasree Ponnaganti
Scientist (ABM)
School of Social Science and Policy Support
ICAR-NIASM, Baramati
Mob: 8639110291

Mr. K. Ravi Kumar
Scientist (Agricultural Extension)
School of Social Science and Policy Support
ICAR-NIASM, Baramati
Mob: 9133120921

Contact Email Id:ssspsniasm@gmail.com

भाकृअनुप
ICAR

हर कदम, हर डगर
किसानों का हमसफर
भारतीय कृषि अनुसंधान परिषद

Agrisearch with a human touch

Application Form

Twenty-One Day Online Training Program

on

Advanced Statistical and Machine Learning Techniques for Data Analysis Using Open-Source Software for Abiotic Stress Management in Agriculture

16 July to5 August 2025

1.	Full Name (in BLOCK letters)				
2.	Highest degree with specialization				
3.	Present Institute Name				
4.	Address for Correspondence				
5.	E-mail address: *Telephone Number Mob/O/R:*				
6.	Date of Birth				
7.	Sex (Male/Female/other)				
8.	Education Qualification:				
	Degree	Subject	Year of passing	Class / Division / Equivalent	University / Institute
	Bachelors Masters Ph.D. Any Other				
9.	Level of Knowledge in Statistics			Beginner / Intermediate / Expert	
10.	Level of Knowledge in R/ Python/ other software			Beginner/Expert	
11.	Area of ongoing research work				
13	Expectations from the training				

**Candidate must fill in all the details*

Signature of the Applicant with date

CERTIFICATE

It is certified that information furnished above is correct.

Signature of the Recommending Authority / Head of the Department/ Institute along with Seal

भाकृअनुप
ICAR

हर कदम, हर डगर
किसानों का हमसफर
भारतीय कृषि अनुसंधान परिषद

Agrisearch with a human touch