From Stationarity to Deep Learning: A Comparative Framework for Stock Price and Volatility Forecasting

Vishvas Ranjan

Table of contents

Time-Series Model

Introduction

A time series model specifies how the random variables $\{X_t\}$ behave jointly over time. This can be done in two main ways:

1. **Full Joint Distribution:** The model might completely specify the joint probability distribution of $\{X_t\}$.
2. **Moments (Means and Covariances):** In many practical situations, especially when dealing with linear processes or Gaussian assumption: it is sufficient to specify just the means and covariances (or autocovariances) of the process. This is because for Gaussian processes, the mean and covariance completely determine the joint distribution.

> **!** Definition
>
> A **time series model** for the observed data $\{x_t\}$ is a specification of the joint distributions (or possibly only the means and covariances) of a sequence of random variables $\{X_t\}$ of which $\{x_t\}$ is postulated to be a realization. Here, x_t is the single outcome of stochastic process X_t.

Zero-mean models

IID noise:

No trend or seasonal component, and the observations are simply independent and identically distributed (iid) random variables with zero mean.

$$P[X \leq x_1, \cdots, X_n \leq x_n] = P[X_1 \leq x_1] \cdots P[X_n \leq x_n] = F(x_1) \cdots F(x_n)$$

F is the cumulative distribution function of each of the iids.

$$P[X_{n+h} \leq x | X_1 = x_1, \cdots, X_n = x_n] = P[X_{n+h} \leq x]$$

Simple Symmetric Random Walk:

A **simple symmetric random walk** is a foundational stochastic process that models a path formed by successive random steps. Assume, we start at $S_0 = 0$. For $t \geq 1$, define

$$S_t = X_1 + X_2 + \cdots + X_t$$

where $\{X_t\}$ are independent and identically distributed (iid) random variables. Here, each binary step is defined as:

$$X_t = \begin{cases} +1 & \text{with probability 0.5 “heads”,} \\ -1 & \text{with probability 0.5 “tails”.} \end{cases}$$

So, we start at position 0 and move ± 1 at each time t based on a fair coin toss.

```
# For reproducibility we set the seed

set.seed(123)
steps <- sample(c(-1, 1), 200, replace = TRUE)
s <- c(0, cumsum(steps))
plot(0:200, s, type = "l",
     xlab = "Time (t)", ylab = expression(S[t]),
     main = "Simple Symmetric Random Walk")
```

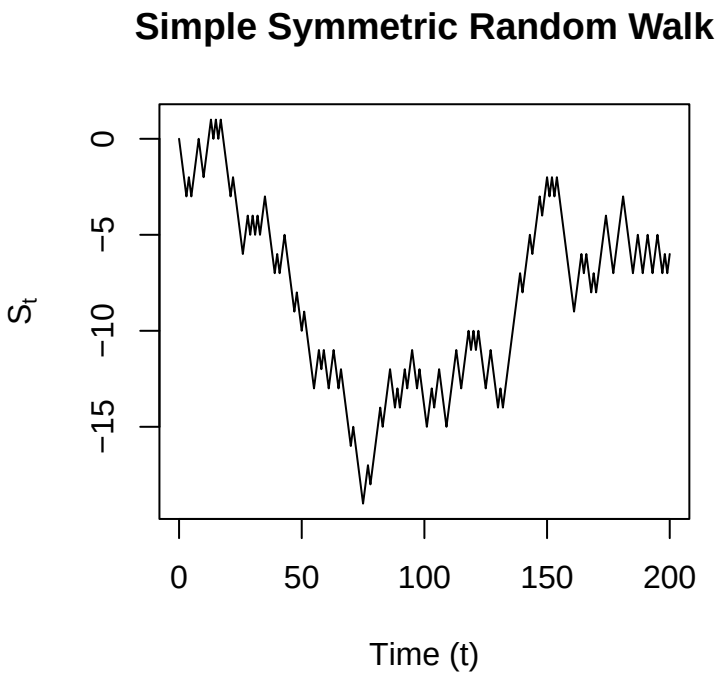


Figure 1: Simulation of simple symmetric RW

Models with trends and seasonality

Using a zero-mean model for data is inappropriate. When analyzing time series data with trends or seasonality, a zero-mean model is often inadequate because such data inherently violates the assumption of a constant mean over time. Instead, the series should be decomposed into distinct components: a **trend** (m_t), capturing slow, long-term changes; **seasonality** (s_t), representing periodic fluctuations; and **stationary residuals** (Y_t), which account for random, zero-mean noise. To model this structure, we can look for parametric methods like **linear regression** or nonparametric techniques (e.g., moving averages).

Assume there is a clear increasing trend in the population of USA. So, we can suggest a model of the form:

$$X_t = m_t + Y_t$$

- m_t (trend component): A slowly changing deterministic function (e.g., linear, quadratic, or exponential).

- Y_t (residuals): A zero-mean stochastic process, ideally stationary.

We can estimate m_t by minimizing the sum of squared residuals:

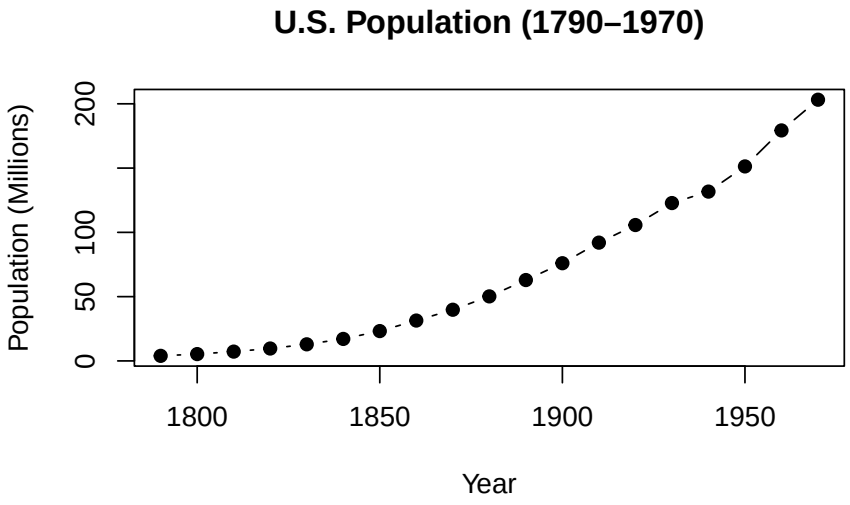
$$\sum_{i=1}^{n}(X_t - m_t)^2$$

We can choose a functional form for m_t such as:

- **Linear**: $m_t = a + bt$
- **Quadratic**: $m_t = a + bt + ct^2$
- **Exponential**: $m_t = ae^{bt}$ (requires linearization via log transforms).

Population of the USA, 1790-1990

```
data <- read.csv("uspop.csv")
ts_data <- ts(data$value, start = 1790, deltat = 10)

# Plot the time series with points
plot(ts_data,
     main = "U.S. Population (1790-1970)",
     xlab = "Year",
     ylab = "Population (Millions)",
     type = "b",  # "b" for both points and lines
     pch = 19)    # Use solid circles as points
```




```r
# Load necessary packages
library(ggplot2)
```

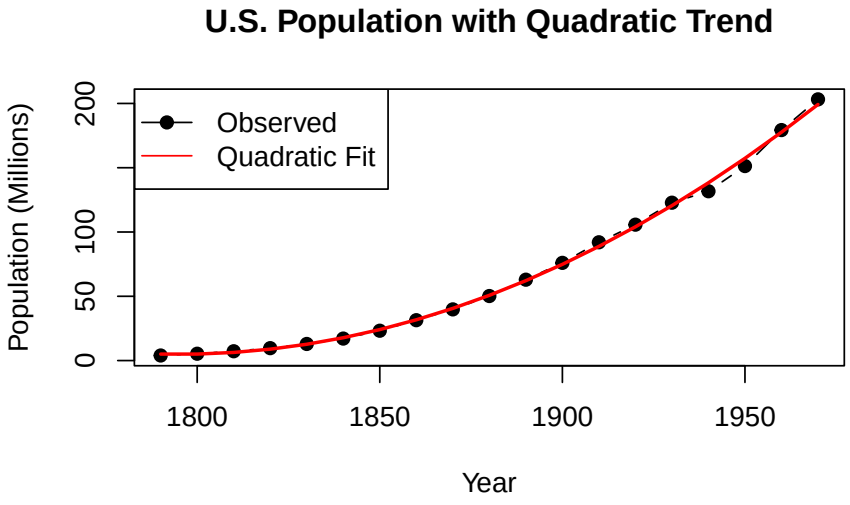
```
Warning: package 'ggplot2' was built under R version 4.4.3
```

```r
# Read and prepare data
data <- read.csv("uspop.csv")
data$t <- data$time - 1790  # Create time variable starting at 0 for 1790

# Fit quadratic model
quad_model <- lm(value ~ t + I(t^2), data = data)

# Add fitted values to dataframe
data$fitted <- predict(quad_model)

# Show model coefficients
summary(quad_model)
```

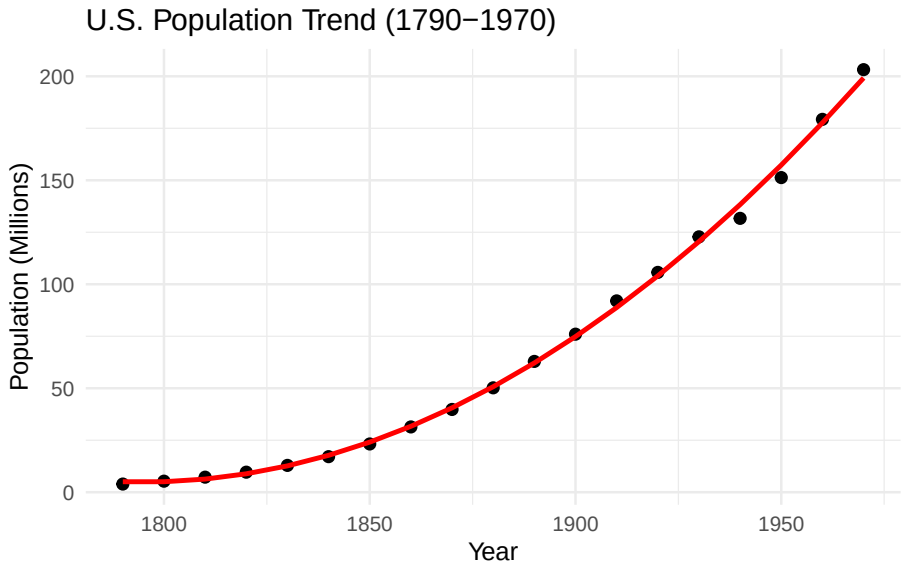
```

Call:
lm(formula = value ~ t + I(t^2), data = data)

Residuals:
    Min      1Q  Median      3Q     Max 
-6.5997 -0.7105  0.2669  1.4065  3.9879 
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.0416692  1.7280922   2.917   0.0101 *
t           -0.0633015  0.0445092  -1.422   0.1742
I(t^2)       0.0063446  0.0002387  26.584 1.14e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.78 on 16 degrees of freedom
Multiple R-squared:  0.9983,    Adjusted R-squared:  0.9981
F-statistic:  4645 on 2 and 16 DF,  p-value: < 2.2e-16
```

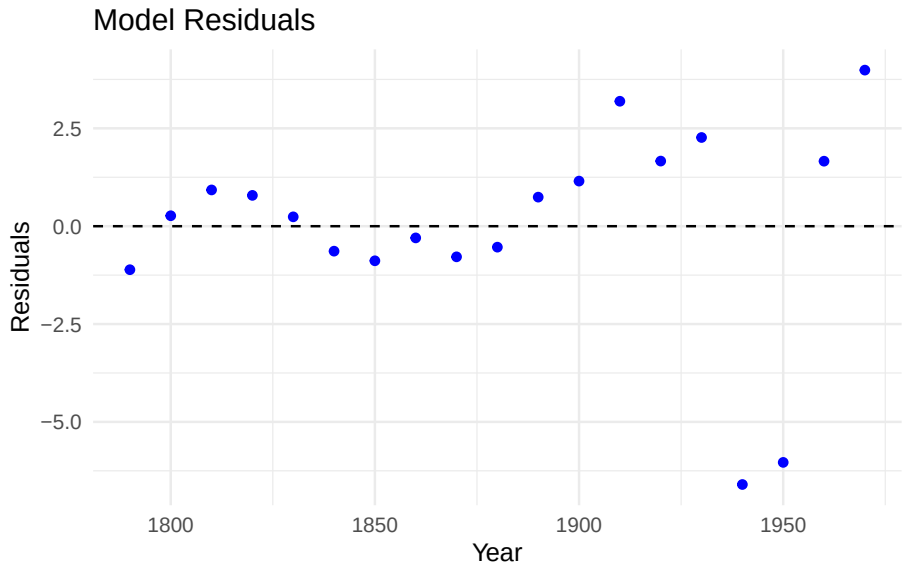
```r
# Base R plot version
plot(ts_data,
     main = "U.S. Population with Quadratic Trend",
     xlab = "Year",
     ylab = "Population (Millions)",
     type = "b", pch = 19)
lines(data$time, data$fitted, col = "red", lwd = 2)
legend("topleft",
       legend = c("Observed", "Quadratic Fit"),
       col = c("black", "red"),
       lty = 1, pch = c(19, NA))
```



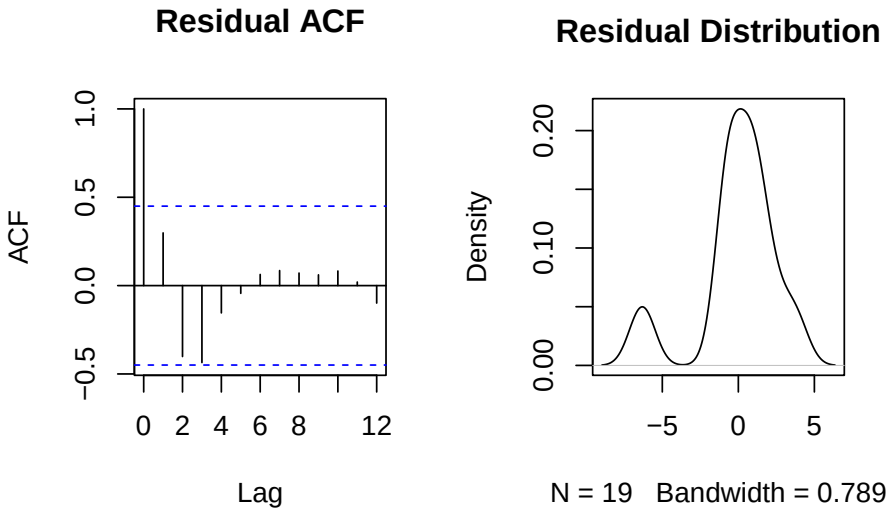
```r
# ggplot2 version
ggplot(data, aes(x = time)) +
  geom_point(aes(y = value), size = 2) +
  geom_line(aes(y = fitted), color = "red", linewidth = 1) +
  labs(title = "U.S. Population Trend (1790-1970)",
       x = "Year",
       y = "Population (Millions)") +
  theme_minimal()
```




```r
# Add residuals
data$residuals <- residuals(quad_model)

# Residual plot
ggplot(data, aes(x = time, y = residuals)) +
  geom_point(color = "blue") +
  geom_hline(yintercept = 0, linetype = "dashed") +
  labs(title = "Model Residuals",
       x = "Year",
       y = "Residuals") +
  theme_minimal()
```




```r
# Residual diagnostics
par(mfrow = c(1,2))
acf(data$residuals, main = "Residual ACF")
plot(density(data$residuals), main = "Residual Distribution")
```



(a) Trend

The U.S. population data exhibits a **strong upward non-linear trend**. The

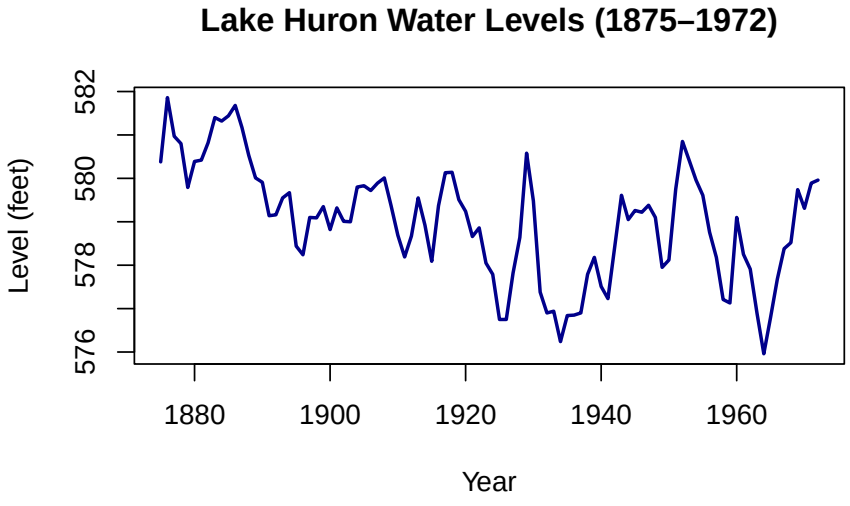
quadratic model (with a significant quadratic term, $p < 0.001$) captures accelerating growth, explaining 99.83% of the variance. The curve reflects increasing growth rates over time, consistent with historical population expansion.

(b) Seasonal Component
There is **no seasonal component** in the data. Measurements are decennial (10-year intervals), and residuals show no cyclical patterns. Seasonality, as seen in monthly or quarterly data, does not apply here.

(c) Sharp Changes in Behavior
The series displays **no abrupt shifts or discontinuities**. Population growth follows a smooth, accelerating trajectory aligned with the quadratic trend. No sudden spikes, drops, or structural breaks are evident.

Level of Lake Huron 1875–1972

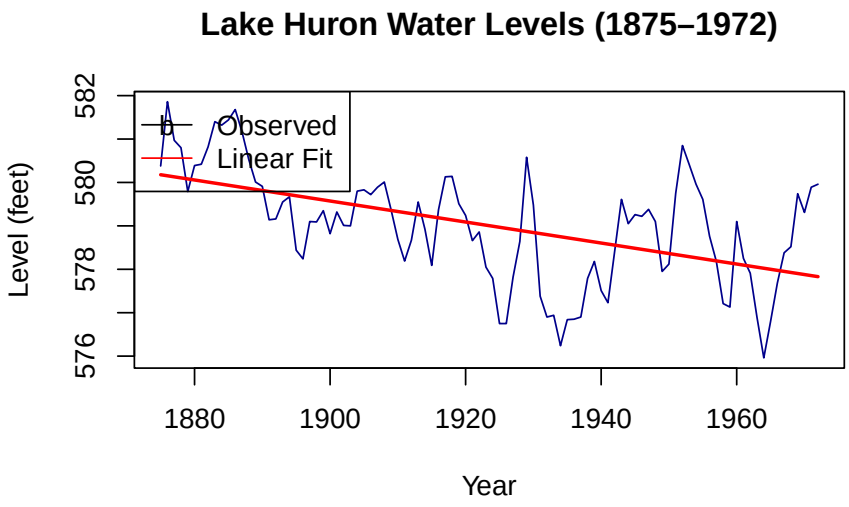
```
lake_data <- read.csv("LakeHuron.csv")
lake_ts <- ts(lake_data$level,
              start = 1875,
              frequency = 1)  # Annual data
plot(lake_ts,
     main = "Lake Huron Water Levels (1875-1972)",
     xlab = "Year",
     ylab = "Level (feet)",
     col = "darkblue",
     lwd = 2)
```



```r
# Load necessary packages
library(ggplot2)
lake_data <- read.csv("LakeHuron.csv")
lake_data$t <- lake_data$year - 1875  # Create time variable starting at 0

# Fit linear model
model <- lm(level ~ t, data = lake_data)
lake_data$fitted <- predict(model)
# Show model summary
summary(model)
```

```
Call:
lm(formula = level ~ t, data = lake_data)

Residuals:
     Min       1Q   Median       3Q      Max
-2.50997 -0.72726  0.00083  0.74402  2.53565

Coefficients:
             Estimate Std. Error  t value Pr(>|t|)
(Intercept) 580.177835   0.226616 2560.182  < 2e-16 ***
t            -0.024201   0.004036   -5.996 3.55e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.13 on 96 degrees of freedom
Multiple R-squared:  0.2725,    Adjusted R-squared:  0.2649
F-statistic: 35.95 on 1 and 96 DF,  p-value: 3.545e-08
```

```r
# Base R plot version
plot(lake_ts,
     main = "Lake Huron Water Levels (1875-1972)",
     xlab = "Year",
     ylab = "Level (feet)",
     col = "darkblue",pch = 98)
lines(lake_data$year, lake_data$fitted, col = "red", lwd = 2)
legend("topleft",
       legend = c("Observed", "Linear Fit"),
       col = c("black", "red"),
       lty = 1, pch = c(98, NA))
```



(a) Trend
The data exhibits a **significant linear downward trend** ($\hat{a}_1 = -0.0242$), p-value (3.55×10^{-8}). The fitted model $\hat{m}_t = 580.178 - 0.0242t$ (where $t = \text{Year} - 1875$) indicates a gradual decline in water levels over time. The trend explains **27.25%** of the variance ($R^2 = 0.2725$), reflecting moderate but meaningful linearity.

(b) Seasonal Component
There is **no seasonal component** in the series. The data is annual, and residuals show no cyclical patterns. Seasonality is irrelevant for yearly measurements.

(c) Sharp Changes in Behavior
The series displays **no abrupt changes**. The decline is smooth and consistent, with no sudden drops or spikes. The residuals also lack discontinuities, supporting a stable linear trend.

Stationary Models

A time series is **stationary** if its statistical properties (mean, variance, covariance) do not change over time. This simplifies modeling, as patterns remain consistent.

Mean Function:

Let $\{X_t\}$ be a time series with $E(X_t^2) < \infty$. The **mean function** of $\{X_t\}$ is

$$\mu_X(t) = E(X_t)$$

Covariance Function:

The **covariance function** of $\{X_t\}$ is

$$\gamma_X(r,s) = Cov(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))]$$

for all integers **r** and **s**.

Weakly Stationary:

$\{X_t\}$ is **weakly stationary** if

(i) $\mu_X(t)$ is independent of t,

and

(ii) $\gamma_X(t+h,t)$ is independent of t for each h.

Strictly Stationary:

$\{X_t\}$ is said to be **strictly stationary** if the joint distribution of any set of observations $(X_{t_1}, X_{t_2}, \cdots, X_{t_n})$ is identical to the shifted set $(X_{t_1+h}, X_{t_2+h}, \cdots, X_{t_n+h})$ for all h.

Note:

Whenever we use the term stationary we shall mean **weakly stationary**, unless we specifically indicate otherwise.

Autocovariance Function (ACVF)

For a stationary time series $\{X_t\}$, the **autocovariance function (ACVF)** at lag h is defined as:

$$\gamma_X(h) = Cov(X_{t+h}, X_t)$$

It depends only on the *lag* h, not on the specific time t (due to stationarity). It measures the linear dependence between observations separated by h time units. At lag $h = 0$, $\gamma_X(0) = Var(X_t)$, which is the variance of the series.

Autocorrelation Function (ACF)

The **autocorrelation function (ACF)** normalizes the ACVF by the variance:

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = Cor(X_{t+h}, X_t)$$

$\rho_X(h)$ is the correlation coefficient between X_{t+h} and X_t, ranging between -1 and 1. For example, If $\rho_X(1) = 0.8$ then observations one lag apart are strongly positively correlated.

Linearity Property of Covariances

For random variables X, Y, Z with finite variance and constants a, b, c :

$$Cov(aX + bY + c, Z) = a.Cov(X, Y) + b.Cov(Y, Z)$$

Examples:

IID Noise:

Given a sequence $\{X_t\}$ is **independent and identically distributed (IID) noise**, so X_t and X_s are independent for $t \neq s$ and all $\{X_t\}$ have the same probability distribution with **zero mean** and **finite variance**. Therefore,

$$E[X_t] = 0; \quad Var(X_t) = E[X_t^2] = \sigma^2 < \infty$$

So,

$$\mu_X(t) = E[X_t] = 0$$

and

$$\gamma_X(h) = Cov(X_{t+h}, X_t) = E[(X_{t+h} - \mu_X(t+h))(X_t - \mu_X(t))]$$

$$\gamma_X(h) = E[X_{t+h}.X_t]$$

If $h = 0$, then

$$\gamma_X(0) = E[X_t.X_t] = \sigma^2$$

If $h \neq 0$, then X_{t+h} and X_t are independent, hence

$$\gamma_X(h) = E[X_{t+h}].E[X_t] = 0.0 = 0$$

Therefore,

$$\gamma_X(h) = \begin{cases} \sigma^2 & \textbf{if } h = 0 \\ 0 & \textbf{if } h \neq 0 \end{cases}$$

So, $\gamma_X(h)$ does not depend on t, hence **IID Noise** is stationary. Therefore,

$$\rho_X(h) = \begin{cases} 1 & \textbf{if } h = 0 \\ 0 & \textbf{if } h \neq 0 \end{cases}$$

No autocorrelation (except at lag 0) makes it a "memoryless" process. This means IID noise has **no memory**: past values do not influence future values. Few examples of IID Noise are **White Noise (e.g., Gaussian white noise)** and **Fair coin tosses (heads = +1, tails = -1)**.

IID noise is the simplest stationary process. Many time series models (e.g., regression errors) assume residuals behave like IID noise. Real-world data often violates IID assumptions (e.g., trends, autocorrelation).

The Random Walk:

A **random walk** $\{S_t\}$ is constructed by cumulatively summing iid noise $\{X_t\}$ with $E[X_t] = 0$ and $Var(X_t) = \sigma^2$. For example, if X_t represents steps of $+1$ or -1 (as in a simple symmetric random walk), the position S_t at time t is the sum of all steps up to t:

$$S_t = X_1 + X_2 + \cdots + X_t, \quad S_0 = 0$$
$$E[S_t] = 0; \quad Var(S_t) = Var(X_1 + X_2 + \cdots + X_t) = t.\sigma^2 < \infty$$

So,

$$\mu_S(t) = E[S_t] = 0$$

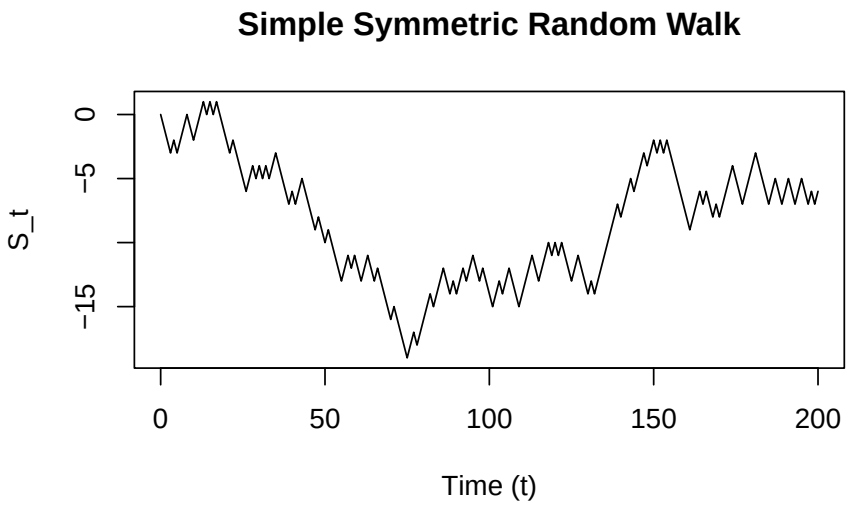
and

$$\gamma_S(h) = Cov(S_{t+h}, S_t) = Cov(S_t + X_{t+1} + X_{t+2} + \cdots + X_{t+h}, S_t)$$

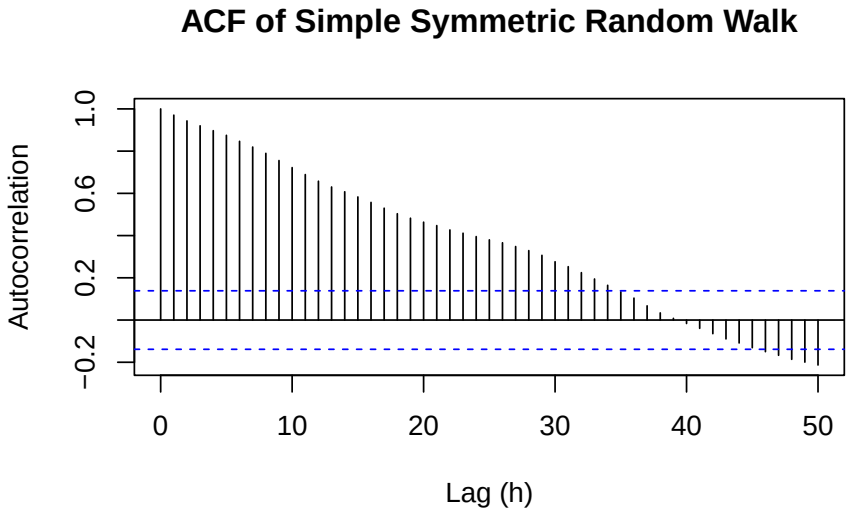
$$\gamma_S(h) = Cov(S_t, S_t) = Var(S_t) = t\sigma^2$$

Since ACVF depends on t, so a simple random walk is **non-stationary**.

```r
# For reproducibility
set.seed(123)
steps <- sample(c(-1, 1), 200, replace = TRUE)
s <- c(0, cumsum(steps))  # s[1:201] contains S =0 to S
plot(0:200, s, type = "l",
     xlab = "Time (t)", ylab = "S_t",
     main = "Simple Symmetric Random Walk")
```




```r
# Compute ACF for the realization (excluding initial S =0)
acf(s[-1],
    main = "ACF of Simple Symmetric Random Walk",
    lag.max = 50,  # Show up to lag 50
    ylab = "Autocorrelation",
    xlab = "Lag (h)")
```



The autocorrelation remains significantly positive for many lags, reflecting the non-stationarity of the random walk. Each value S_t depends heavily on all prior values S_{t-1}, S_{t-2}, $\cdots$ leading to high autocorrelation at all lags.

First-order moving average MA(1) process:

The **first-order moving average (MA(1))** process is defined as:

$$X_t = Z_t + \theta.Z_{t-1}, \quad \{Z_t\} \sim WN(0, \sigma^2)$$

where $\{Z_t\}$ is white noise (uncorrelated, zero mean, variance σ^2) and θ is a constant.

$$\mu_X(t) = E[X_t] = E[Z_t] + \theta.E[Z_{t-1}] = 0$$

$$Var(X_t) = Var(Z_t) + \theta^2.Var(Z_{t-1}) = \sigma^2(1+\theta^2)$$

So, the ACVF:

$$\gamma_X(h) = \begin{cases} \sigma^2(1+\theta^2) & \textbf{if } h = 0 \\ \sigma^2\theta & \textbf{if } h = \pm 1 \\ 0 & \textbf{if } |h| > 1 \end{cases}$$

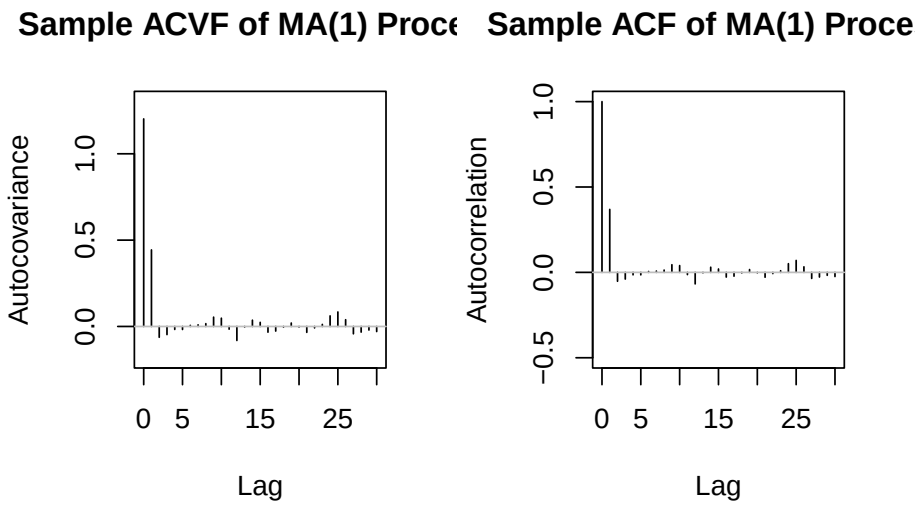
and, the ACF:

$$\rho_X(h) = \begin{cases} 1 & \textbf{if } h = 0 \\ \frac{\theta}{1+\theta^2} & \textbf{if } h = \pm 1 \\ 0 & \textbf{if } |h| > 1 \end{cases}$$

Hence, MA(1) is stationary.

```r
# Load required package
library(stats)

# Simulate MA(1) process with   = 0.5
set.seed(123)
n <- 1000  # Number of observations
theta <- 0.5
sigma <- 1
ma1_process <- arima.sim(model = list(ma = theta), n = n, sd = sigma)

# Compute ACVF and ACF
acvf <- acf(ma1_process, type = "covariance", plot = FALSE)
```

```r
acf_values <- acf(ma1_process, plot = FALSE)

# Plot ACVF and ACF side by side
par(mfrow = c(1, 2))  # 1 row, 2 columns

# Plot ACVF
plot(acvf$lag, acvf$acf, type = "h",
     main = "Sample ACVF of MA(1) Process",
     xlab = "Lag", ylab = "Autocovariance",
     ylim = c(min(acvf$acf)-0.1, max(acvf$acf)+0.1))
abline(h = 0, col = "gray")

# Plot ACF
plot(acf_values$lag, acf_values$acf, type = "h",
     main = "Sample ACF of MA(1) Process",
     xlab = "Lag", ylab = "Autocorrelation",
     ylim = c(-0.5, 1))
abline(h = 0, col = "gray")
```



First-order autoregression or AR(1) process

The **AR(1) process** is a foundational time series model where each observation depends linearly on its immediate past value plus random noise.

$$X_t = \phi X_{t-1} + Z_t, \quad t = 0, \pm 1, \cdots,$$

$\{Z_t\} \sim WN(0, \sigma^2)$
where:

- ϕ is the autoregressive coefficient ($|\phi| < 1$ for stationarity).
- $\{Z_t\}$ is white noise (uncorrelated, zero mean, variance σ^2).

Computing expectation:

$$E[X_t] = E[\phi X_{t-1} + Z_t]$$

$$E[X_t] = \phi E[X_{t-1}] + E[Z_t]$$

Since $E[Z_t] = 0$,

$$E[X_t] = \phi E[X_{t-1}]$$

Let $E[X_t] = \mu$ and the process $\{X_t\}$ is stationary, so $E[X_t] = E[X_{t-1}]$:

$$\mu = \phi\mu$$

$$\mu(1 - \phi) = 0$$

Since $|\phi| < 1$:

$$\mu = 0$$

So, $E[X_t] = 0$.
Computing ACVF:

$$\gamma_X(h) = Cov(X_t, X_{t-h}) = Cov(\phi X_{t-1}, X_{t-h}) + Cov(Z_t, X_{t-h})$$

$$\gamma_X(h) = \phi\gamma_X(h-1) + 0 = \cdots = \phi^h\gamma_X(0)$$

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = \phi^{|h|}, \quad h = 0, \pm 1, \cdots$$

Computing $\gamma_X(0)$:

$$\gamma_X(0) = Cov(X_t, X_t) = Cov(\phi X_{t-1} + Z_t, \phi X_{t-1} + Z_t)$$

$$\gamma_X(0) = \phi.Cov(X_{t-1}, \phi X_{t-1}+Z_t)+Cov(Z_t, \phi X_{t-1}+Z_t) = \phi^2.Cov(X_{t-1}, X_{t-1})+Cov(Z_t, Z_t)$$

$$\gamma_X(0) = \phi^2\gamma_X(0) + \sigma^2$$

$$\gamma_X(0) = \frac{\sigma^2}{1-\phi^2}$$

Hence,

$$\gamma_X(h) = \frac{\sigma^2\phi^{|h|}}{1-\phi^2}$$

Since, ACVF is independent of t, hence $\{X_t\}$ is stationary.

Sample Autocorrelation Function

When analyzing a time series $\{x_1, x_2, \cdots, x_n\}$, one of the key tools is the **sample autocorrelation function (sample ACF)**. It provides insights into the correlation of the time series with itself at different time lags.

Sample Mean:

Given observations $\{x_1, x_2, \cdots, x_n\}$, the **sample mean** is:

$$\overline{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$$

This is just the average of all observed values and is used when centering the data (subtracting the mean) before computing the sample autocovariance and autocorrelation.

Sample Autocovariance Function:

The **sample autocovariance** at lag h (denoted $\hat{\gamma}(h)$) is defined by:

$$\hat{\gamma}(h) = \frac{1}{n} \sum_{t=1}^{n-|h|} (x_{t+|h|} - \overline{x})(x_t - \overline{x}), \quad -n < h < n$$

and $\hat{\gamma}(h) = \hat{\gamma}(-h)$ for negative h.

Autocovariance measures how the time series covaries with itself at different lags. For example, $\hat{\gamma}(1)$ roughly measures how x_t relates to x_{t+1}, $\hat{\gamma}(2)$ relates x_t to x_{t+2}, etc.

We define **sample covariance matrix** as $\hat{\Gamma} n := [\hat{\gamma}(i-j)]_{i,j=1}^{n}$ and using n as divisor ensures that this matrix is **non-negative definite**.

Sample Autocorrelation Function:

The **sample autocorrelation function** at lag h (denoted $\hat{\rho}(h)$) is:

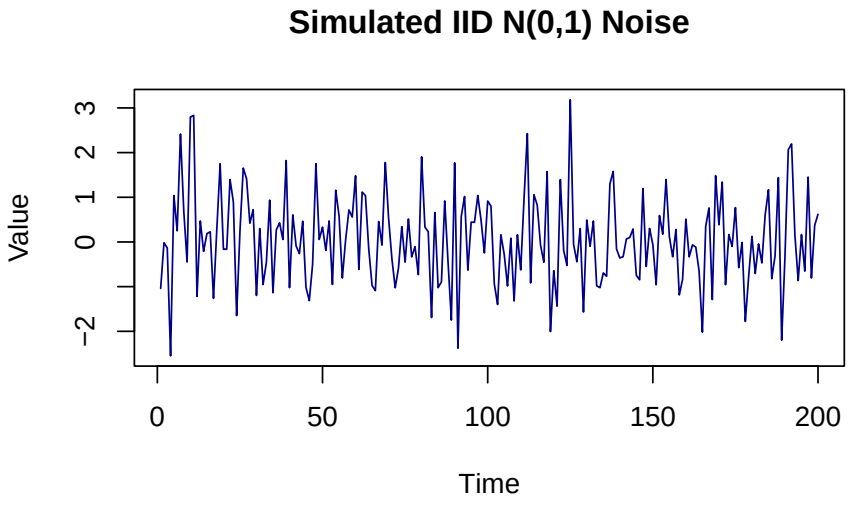
$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$$

We define **sample correlation matrix** as $\hat{R} n := [\hat{\rho}(i-j)]_{i,j=1}^{n}$ and it is also **non-negative definite**. Each of its diagonal elements is **1**.

Illustration of Sample ACF through IID N(0,1) noise

We assume $X_t \sim N(0,1)$, i.e., Gaussian white noise with mean 0 and variance 1. We first simulate 200 values of $N(0,1)$ noise.

```
# Set seed for reproducibility

# Generate 200 observations of IID N(0,1) noise
n <- 200
iid_noise <- rnorm(n, mean = 0, sd = 1)

# Plot the simulated series (Figure 1.12)
plot.ts(iid_noise,
        main = "Simulated IID N(0,1) Noise",
        ylab = "Value",
        col = "darkblue")
```



Computing **sample mean**, **sample ACF** using above definitions.

$$\bar{X} = \frac{\sum_{t=1}^{200} X_t}{200}$$

$$\hat{\rho}(h) = \frac{\sum_{t=1}^{200-|h|}(X_{t+|h|} - \overline{X})(X_t - \overline{X})}{\sum_{t=1}^{200}(X_t - \bar{X})^2}$$

```
# Sample mean
n<-200
x_bar <- sum(iid_noise) / n
cat("Sample Mean =",x_bar, "\n")
```

```
Sample Mean = 0.05846075
```

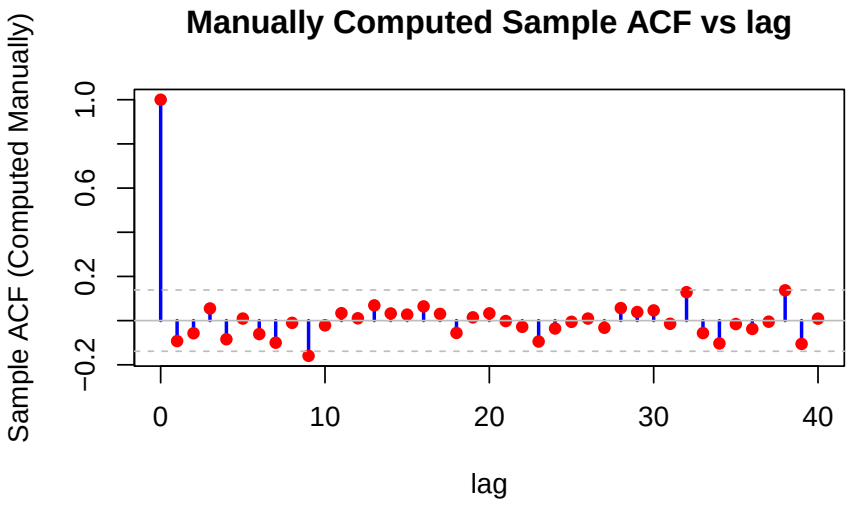
```
acvf <- function(h) {
  sum((iid_noise[(1 + h) : (n - h) ] - x_bar)*(iid_noise[1: (n - h) ]-x_bar)) / n
}
lag <- 0:40
acfs_values <- sapply(lag, acvf)
```

```
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
```

```
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
```

```
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
Warning in (iid_noise[(1 + h):(n - h)] - x_bar) * (iid_noise[1:(n - h)] - :
longer object length is not a multiple of shorter object length
```

```r
acfs_manual <- acfs_values/acfs_values[1]

# Create the plot using type "h" for vertical lines.
plot(lag, acfs_manual, type = "h", lwd = 2, col = "blue",
     xlab = "lag", ylab = "Sample ACF (Computed Manually)",
     main = "Manually Computed Sample ACF vs lag")
points(lag, acfs_manual, pch = 16, col = "red")
abline(h = 0, col = "gray")

# Add Confidence Bounds AFTER the plot is created.
conf_bound <- 1.96 / sqrt(n)
abline(h = c(-conf_bound, conf_bound), col = "gray", lty = 2)
```



For large number of simulations n, $X \approx 0$. Hence,

$$\hat{\rho}(h) \approx \frac{\sum_{t=1}^{n-|h|} X_{t+|h|} X_t}{\sum_{t=1}^{n} (X_t - \bar{X})^2}$$

$$\hat{\rho}(h) \approx \frac{1}{n\sigma^2} \sum_{t=1}^{n-|h|} X_{t+|h|} X_t$$

Now, we will look at the $E[\hat{\rho}(h)]$ and $Var(\hat{\rho}(h))$:

We will use the fact that X_{t+h} and X_t are independent.

$$E[\hat{\rho}(h)] \approx \frac{1}{n\sigma^2} \sum_{t=1}^{n-|h|} E[X_{t+h} X_t] = 0$$

$$Var(\hat{\rho}(h)) \approx \frac{1}{n^2\sigma^4} \sum_{t=1}^{n-|h|} Var(X_{t+h} X_t)$$

$$Var(X_{t+h} X_t) = E[(X_{t+h} X_t)^2] - (E[X_{t+h} X_t])^2 = E[X_{t+h}^2 X_t^2] - 0$$

$$Var(X_{t+h} X_t) = E[X_{t+h}^2] E[X_t^2] = (Var(X_{t+h}) + (E[X_{t+h}])^2).(Var(X_t) + (E[X_t])^2)$$

$$Var(X_{t+h}, X_t) = \sigma^4$$

Hence,

$$Var(\hat{\rho}(h)) \approx \frac{1}{n^2\sigma^4} n\sigma^4 \approx \frac{1}{n}$$

By the **Central Limit Theorem (CLT)**, the $\hat{\rho}(h)$ converges to a normal distribution for large n. Thus, for IID noise, the sample ACF at any *lag* $h \neq 0$ is approximately:

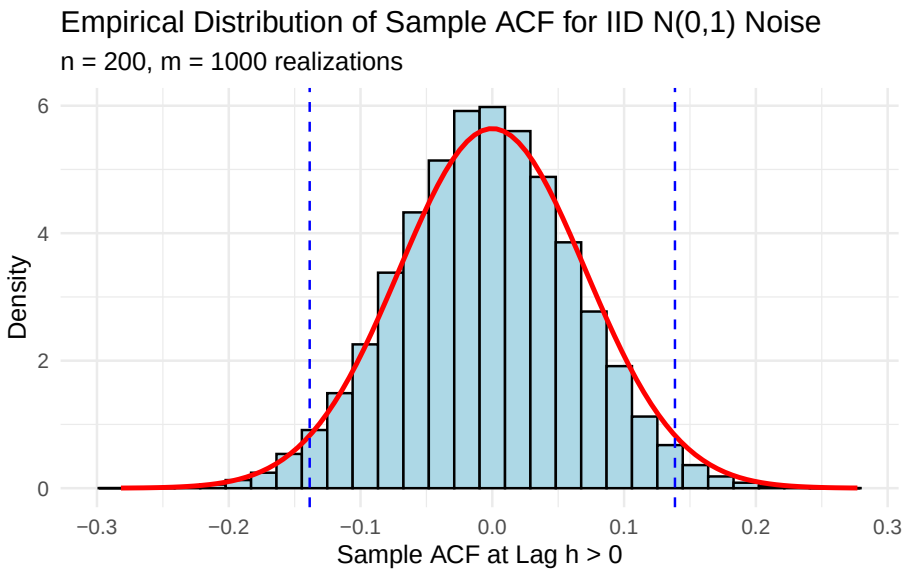
$$\hat{\rho}(h) \sim N(0, \frac{1}{n})$$

Verification: We first simulate m independent datasets of length n from an IID $N(0,1)$ process. Large m ensures the Central Limit Theorem (CLT) applies. For each dataset, calculate the sample ACF $\hat{\rho}(h)$ at lags $h = 1, 2, \cdots, H$. Aggregate all $\hat{\rho}(h)$ values across lags ($h > 0$) and datasets. This creates an empirical distribution of sample ACFs. Now, we can plot a histogram of the pooled ACF values. This should overlay the theoretical normal density $N(0, \frac{1}{n})$ and 95% confidence bounds $\pm 1.96/\sqrt{n}$.

```r
# Load required package
library(ggplot2)

# Set parameters
set.seed(123)     # Reproducibility
n <- 200          # Length of each time series
m <- 1000         # Number of realizations (large for CLT)
H <- 40           # Maximum lag
conf_level <- 0.95

# Simulate m realizations of IID N(0,1) noise
acf_values <- matrix(NA, nrow = m, ncol = H)

for (k in 1:m) {
  # Generate IID N(0,1) noise
  noise <- rnorm(n, mean = 0, sd = 1)

  # Compute sample ACF up to lag H
  acf_result <- acf(noise, lag.max = H, plot = FALSE)$acf

  # Store ACF values at lags 1 to H
  acf_values[k, ] <- acf_result[2:(H + 1)]
}
```

```r
# Pool all ACF values (across lags and realizations)
pooled_acf <- as.vector(acf_values)

# Theoretical parameters under H0 (IID noise)
mu <- 0
sigma <- 1 / sqrt(n)

# Compute proportion of values outside confidence bounds
conf_bound <- qnorm((1 + conf_level)/2) * sigma
out_of_bounds <- mean(abs(pooled_acf) > conf_bound)

# Create histogram with theoretical normal curve
ggplot(data.frame(x = pooled_acf), aes(x = x)) +
  geom_histogram(aes(y = ..density..),
                 bins = 30,
                 fill = "lightblue",
                 color = "black") +
  stat_function(fun = dnorm,
                args = list(mean = mu, sd = sigma),
                color = "red",
                linewidth = 1) +
  geom_vline(xintercept = c(-conf_bound, conf_bound),
             color = "blue",
             linetype = "dashed") +
  labs(title = "Empirical Distribution of Sample ACF for IID N(0,1) Noise",
       subtitle = sprintf("n = %d, m = %d realizations", n, m),
       x = "Sample ACF at Lag h > 0",
       y = "Density") +
  theme_minimal()
```

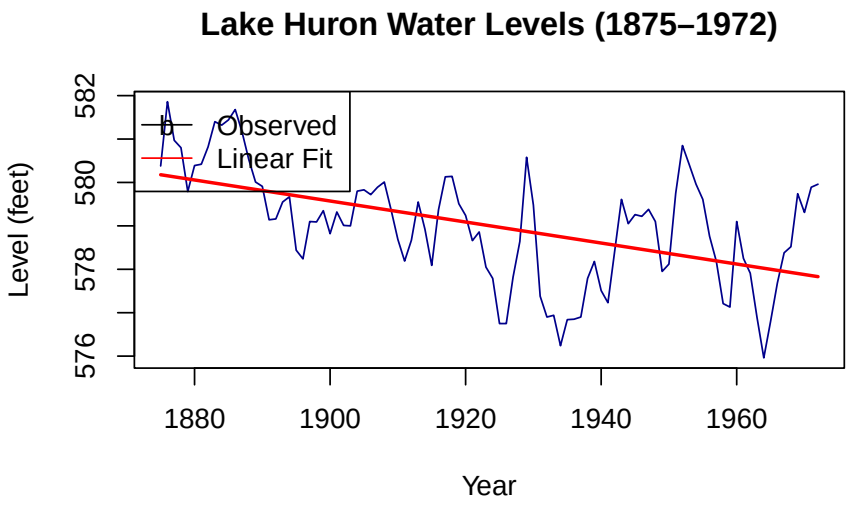
```
Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(density)` instead.
```



Lake Huron Residuals Modeling Process

Earlier, the Lake Huron water level data (1875–1972) was analyzed to identify a suitable time series model:

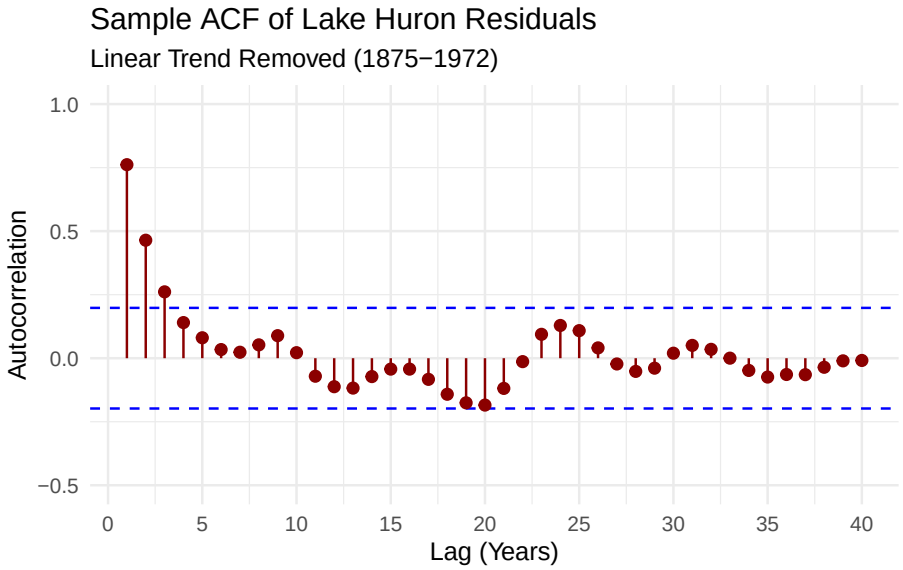
```
# Load necessary packages
library(ggplot2)
lake_data <- read.csv("LakeHuron.csv")
lake_data$t <- lake_data$year - 1875  # Create time variable starting at 0

# Fit linear model
model <- lm(level ~ t, data = lake_data)
lake_data$fitted <- predict(model)
# Base R plot version
plot(lake_ts,
     main = "Lake Huron Water Levels (1875-1972)",
     xlab = "Year",
     ylab = "Level (feet)",
     col = "darkblue",pch = 98)
lines(lake_data$year, lake_data$fitted, col = "red", lwd = 2)
legend("topleft",
       legend = c("Observed", "Linear Fit"),
       col = c("black", "red"),
       lty = 1, pch = c(98, NA))
```



We can see that our initial model constitutes of a straight line fitted to the data (linear trend: $\hat{m}_t = a + bt$). We then analyzed the residuals $\{y_1, y_2, \cdots, y_{98}\}$. We now compute Sample ACF for the Lake Huron Residuals:

```
lake_data$residuals <- residuals(model)  # Store residuals

# Compute sample ACF of residuals
acf_res <- acf(lake_data$residuals, plot = FALSE, lag.max = 40)

# Convert ACF results to dataframe for ggplot
acf_df <- with(acf_res, data.frame(lag, acf))

# Create confidence bounds (95% CI)
n <- nrow(lake_data)
conf_level <- 1.96/sqrt(n)

# Plot ACF with ggplot
ggplot(acf_df[-1,], aes(x = lag, y = acf)) +  # [-1,] removes lag 0
  geom_hline(yintercept = c(-conf_level, conf_level),
             color = "blue", linetype = "dashed") +
  geom_segment(aes(xend = lag, yend = 0), color = "darkred") +
  geom_point(color = "darkred", size = 2) +
  labs(title = "Sample ACF of Lake Huron Residuals",
       subtitle = "Linear Trend Removed (1875-1972)",
       x = "Lag (Years)",
       y = "Autocorrelation") +
  theme_minimal() +
```

```
  scale_x_continuous(breaks = seq(0, 40, 5)) +
  ylim(c(-0.5, 1))
```




```
r1 <- acf_df[1,2]
r2 <- acf_df[2,2]
r3 <- acf_df[3,2]
r4 <- acf_df[4,2]
gradual_Decay <- list()
gradual_Decay[1] <- r2 / r1
gradual_Decay[2] <- r3 / r2
gradual_Decay[3] <- r4 / r3
gradual_Decay
```

```
[[1]]
[1] 0.7615963

[[2]]
[1] 0.6097112

[[3]]
[1] 0.5622723
```

```
r1
```

```
[1] 1
```

We can see, after fitting a linear trend, the residuals $\{y_t\}$ violated the IID noise assumption. The ACF showed a gradual decay, with $\frac{\hat{\rho}(h+1)}{\hat{\rho}(h)} \approx 0.7$, characteristic of autoregressive (AR) processes. The geometric decay $\hat{\rho}(h) \approx \phi^{|h|}$ suggested an **AR(1) process**. The decay rate $\phi \approx 0.7$ was inferred from $\hat{\rho}(1) \approx 0.7$.

```
# Fit AR(1) to residuals
ar1_model <- arima(lake_data$residuals, order = c(1,0,0), include.mean = FALSE)
# Model summary
cat("AR(1) Model Summary:\n")
```

```
AR(1) Model Summary:
```

```
print(ar1_model)
```

```

Call:
arima(x = lake_data$residuals, order = c(1, 0, 0), include.mean = FALSE)

Coefficients:
         ar1
      0.7826
s.e.  0.0635

sigma^2 estimated as 0.4975:  log likelihood = -105.32,  aic = 214.65
```

```
cat("\nPhi estimate:", ar1_model$coef, "\n")
```

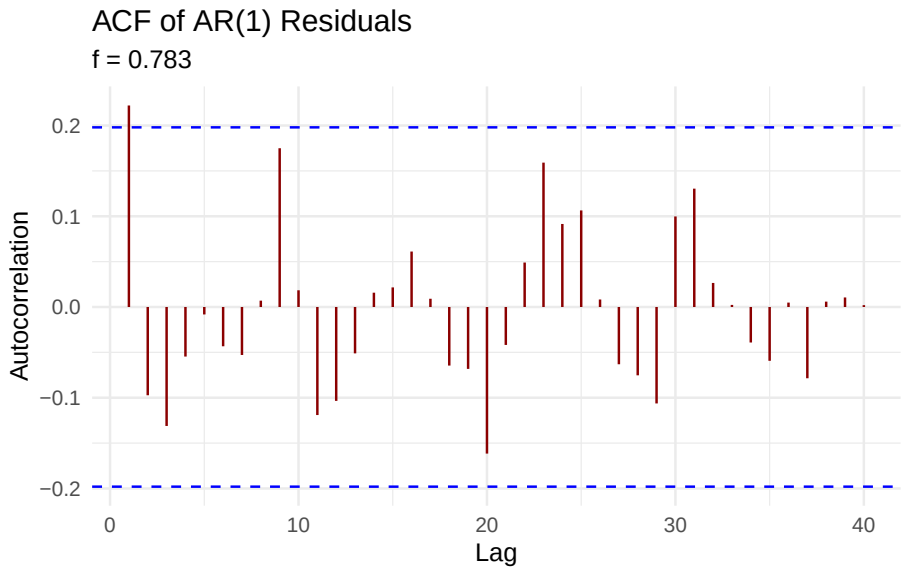
```

Phi estimate: 0.7826118
```

```
cat("Noise variance:", ar1_model$sigma2, "\n")
```

```
Noise variance: 0.4975348
```

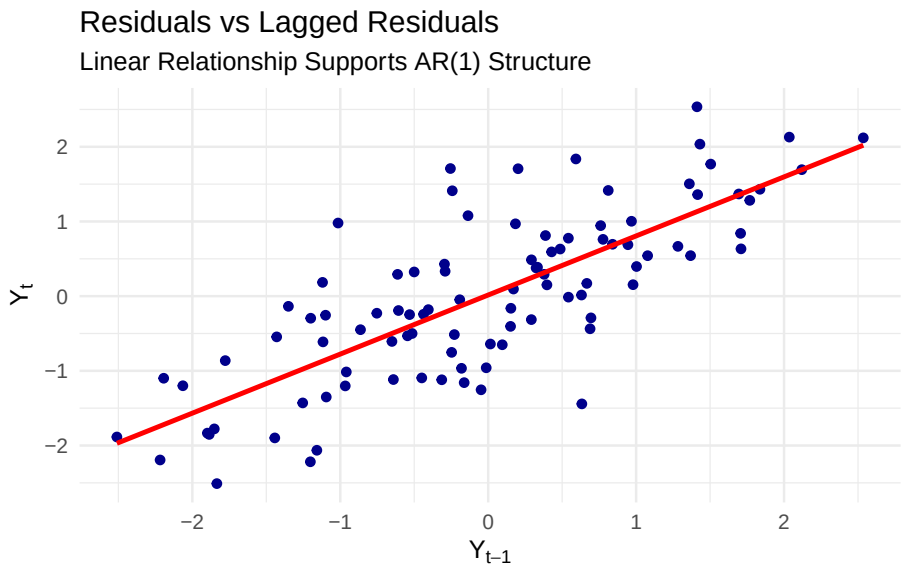
```
# Get AR(1) residuals
ar1_residuals <- residuals(ar1_model)

# Plot 1: ACF of AR(1) residuals
acf_ar1 <- acf(ar1_residuals, plot = FALSE, lag.max = 40)
conf_bound <- 1.96/sqrt(length(ar1_residuals))

ggplot(data.frame(lag = acf_ar1$lag, acf = acf_ar1$acf)[-1,],
```

```
       aes(x = lag, y = acf)) +
  geom_hline(yintercept = c(-conf_bound, conf_bound),
             color = "blue", linetype = "dashed") +
  geom_segment(aes(xend = lag, yend = 0), color = "darkred") +
  labs(title = "ACF of AR(1) Residuals",
       subtitle = paste("  =", round(ar1_model$coef, 3)),
       x = "Lag", y = "Autocorrelation") +
  theme_minimal()
```




```
# Plot 2: Scatterplot of Y_t vs Y_{t-1}
lake_data$lag1_residual <- c(NA, lake_data$residuals[-nrow(lake_data)])

ggplot(lake_data, aes(x = lag1_residual, y = residuals)) +
  geom_point(color = "darkblue") +
  geom_smooth(method = "lm", formula = y ~ x, se = FALSE, color = "red") +
  labs(title = "Residuals vs Lagged Residuals",
       subtitle = "Linear Relationship Supports AR(1) Structure",
       x = expression(Y[t-1]),
       y = expression(Y[t])) +
  theme_minimal()
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_smooth()`).

Warning: Removed 1 row containing missing values or values outside the scale range
(`geom_point()`).
```



The linear relationship between Y_t and Y_{t-1} in the above figure supported the AR(1) structure $Y_t = \phi Y_{t-1} + Z_t$. Least squares regression of Y_t on Y_{t-1} gave $\hat{\phi} = 0.7826$. Residual variance ($\sigma^2 = 0.4975$) was computed, but the residuals still showed slight autocorrelation.

Hence, the fitted model:

$$Y_t = 0.78Y_{t-1} + Z_t, \quad Z_t \sim IID(0, 0.50)$$

Residual ACF of **AR(1)** showed slight excess correlation at lag 1 ($\hat{\rho}(1)$ is outside the bound only). This hinted that the AR(1) model might not fully capture dependencies. So, we can go for another model.

A better fit could be done by using **AR(2) process**. The **AR(1)** residuals had a sample ACF value outside bounds at lag 1, indicating unresolved dependence. This suggested the need for a higher-order model.

```
# Fit AR(2) to residuals
ar2_model <- arima(lake_data$residuals, order = c(2,0,0), include.mean = FALSE)
# Model summary
cat("AR(2) Model Summary:\n")
```

```
AR(2) Model Summary:
```

```
print(ar2_model)
```

```
Call:
arima(x = lake_data$residuals, order = c(2, 0, 0), include.mean = FALSE)

Coefficients:
         ar1      ar2
      1.0050  -0.2925
s.e.  0.0976   0.1002

sigma^2 estimated as 0.4572:  log likelihood = -101.26,  aic = 208.51
```

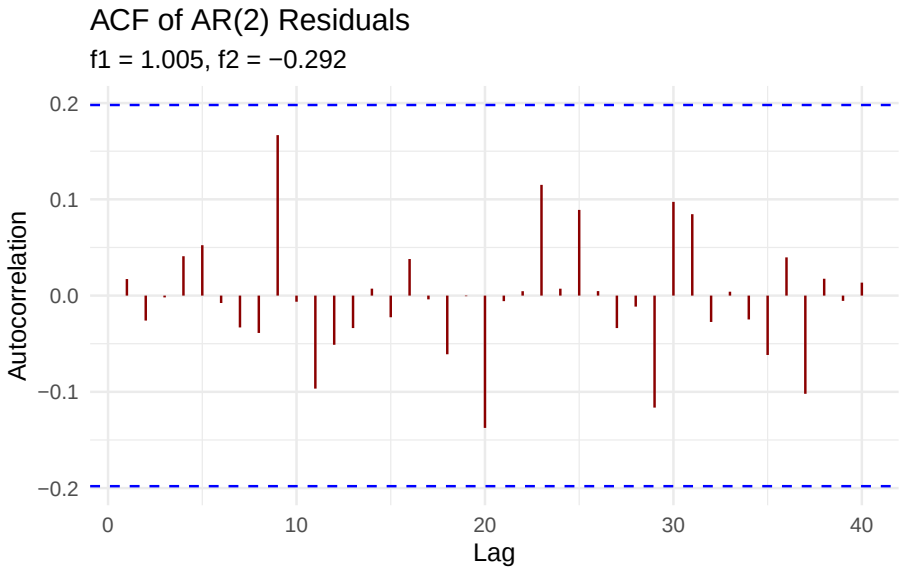
```r
cat("\nPhi estimate:", ar2_model$coef, "\n")
```

```
Phi estimate: 1.005013 -0.2924751
```

```r
cat("Noise variance:", ar2_model$sigma2, "\n")
```

```
Noise variance: 0.4571513
```

```r
# Get AR(1) residuals
ar2_residuals <- residuals(ar2_model)

# Plot 1: ACF of AR(1) residuals
acf_ar2 <- acf(ar2_residuals, plot = FALSE, lag.max = 40)
conf_bound <- 1.96/sqrt(length(ar2_residuals))

ggplot(data.frame(lag = acf_ar2$lag, acf = acf_ar2$acf)[-1,],
       aes(x = lag, y = acf)) +
  geom_hline(yintercept = c(-conf_bound, conf_bound),
             color = "blue", linetype = "dashed") +
  geom_segment(aes(xend = lag, yend = 0), color = "darkred") +
  labs(title = "ACF of AR(2) Residuals",
       subtitle = sprintf(" 1 = %.3f,  2 = %.3f",
                          ar2_model$coef[1],
                          ar2_model$coef[2]),
       x = "Lag", y = "Autocorrelation") +
  theme_minimal()
```



We found residual variance ($\sigma^2 = 0.4571$) to be lesser than before and also from the ACF, we can see no autocorrelation. The fitted model:

$$Y_t = 1.005Y_{t-1} - 0.292Y_{t-1} + Z_t, \quad Z_t \sim IID(0, 0.457)$$

$\phi_1 = 1.005$ indicates strong persistence from the immediate past and $\phi_2 = -0.292$ indicates negative correction for over-persistence, introducing mean reversion. All lags within confidence bounds confirmed noise independence.

The Lake Huron residuals were best modeled by an AR(2) process, which reduced residual variance by 11.8% compared to AR(1) and also eliminated significant autocorrelation in residuals.

Estimation and Elimination of Trend and Seasonal Components

The first step in time series analysis is **visual inspection** of the data to identify structural breaks, outliers, or patterns. If trends or seasonality are present, the **classical decomposition model** is often applied:

$$X_t = m_t + s_t + Y_t$$

where m_t is a slowly varying **trend component**, s_t is a periodic **seasonal component** with known period d, and Y_t represents stationary **noise**. This

decomposition isolates systematic patterns from random fluctuations, enabling focused analysis of each component (e.g., modeling trends separately or validating stationarity in residuals).

Estimation and Elimination of Trend in the Absence of Seasonality

A non-seasonal model with trend will be of form:

$$X_t = m_t + Y_t, \quad t = 1, 2, \cdots$$

m_t is the trend component and Y_t is zero-mean noise.

Trend Estimation

Smoothing with a finite moving average filter:

A **moving-average filter** is the simplest sort of "low-pass" smoother: it replaces each point in your series by the average of its neighbors over a fixed window, thereby stripping away the rapidly-fluctuating (high-frequency) component and leaving us with a slowly-varying estimate of the trend. Here, we will see **two-sided** moving-average filter.

The two-sided moving average for a time series X_t is defined as:

$$\widehat{m}_t = \frac{1}{2q+1} \sum_{j=-q}^{q} X_{t-j}, \quad q+1 \leq t \leq n-q$$

where q determines the window size (e.g., $q = 2$ uses 5 terms: $X_{t-2}, X_{t-1}, X_t, X_{t+1}, X_{t+2}$).

For $t \leq q$, we use $X_t := X_q$ and for $t \geq n - q$, we use $X_t := X_n$. The trend m_t is approximately linear over the window $[t-q, t+q]$. The noise Y_t averages to zero within the window.

By averaging values within a window of size $2q+1$, rapid fluctuations (high frequencies) cancel out due to their randomness. Slow-moving trends (low frequencies) remain largely unaffected if they are approximately linear or polynomial over the window.

A moving average (MA) can be expressed as a **linear filter** or **linear operator** by defining it as a weighted sum of the time series observations. Mathematically, this is represented using a **convolution** of the time series with a set of filter weights. We can think of $\widehat{m}_t$ defined above as a linear filter that applies a set of weights $\{a_j\}$ o lagged values of the time series $\{X_t\}$ to produce a smoothed series $\{\widehat{m}_t\}$:

$$\hat{m}_t = \sum_{j=-\infty}^{\infty} a_j X_{t-j}$$

Here, the weights a_j are non-zero only over a finite window $[-q, q]$, making it a **finite impulse response (FIR) filter**.

For symmetric moving average, we consider weights to be equal.

$$a_j = (2q+1)^{-1}, \quad |j| \leq q$$

For large q, provided $(2q+1)^{-1} \sum_{j=-q}^{q} Y_{t-j} \approx 0$ it not only will attenuate noise but at the same time will allow linear trend functions to pass without distortion. However, we must beware of choosing q to be too large, since if trend function is not linear, the filtered process, although smooth, will not be a good estimate of trend component. By clever choice of the weights $\{a_j\}$ it is possible to design a filter that will not only be effective in attenuating noise in the data, but that will also allow a larger class of trend functions (for example all polynomials of degree less than or equal to 3) to pass through without distortion. The **Spencer 15-point** moving average is a filter that passes **polynomials of degree 3** without distortion.

Exponential Smoothing:

We now introduced one-sided moving average where only past datapoints influence the values. **Exponential smoothing** is a technique used to smooth time series data and generate forecasts by assigning **exponentially decreasing weights** to past observations. We first consider a **smoothing parameter** $\alpha \in (0, 1)$ value between **0 and 1** that controls the weight given to recent observations. We initialize for $t = 1$, $\hat{m}_1 = X_1$ and then define recursive formula for $t \geq 2$:

$$\hat{m}_t = \alpha X_t + (1-\alpha)\hat{m}_{t-1}, \quad t = 2, 3, \cdots$$

We can see the exponential weighted structure as the smoothed value $\hat{m}_t$ is a weighted average of all past observations:

$$\hat{m}_t = \sum_{j=0}^{t-2} \alpha(1-\alpha)^j X_{t-j} + (1-\alpha)^{t-1} X_1, \quad t \geq 2$$

Here, weights are decaying exponentially and the oldest term $(1-\alpha)^{j-1} X_1$ becomes negligible as t grows. The weights form a **geometric series**, decreasing exponentially with each lag.

Polynomial Fitting:

Trend estimation by polynomial fitting involves modeling the underlying trend m_t of a time series as a polynomial function of time,

$$m_t = a_0 + a_1 t + a_2 t^2 + \cdots + a_p t^p,$$

where p is the polynomial order and $(a_0, a_1, \ldots, a_p)$ are coefficients estimated via least squares minimization. The coefficients are chosen to minimize the sum of squared residuals:

$$\sum_{i=1}^{n} \left(x_i - \left(a_0 + a_1 t + a_2 t^2 + \cdots + a_p t^p\right)\right)^2,$$

ensuring the polynomial closely follows the observed data ($\{x_1, \ldots, x_n\}$). For example, a quadratic trend ($p = 2$) is modeled as

$$m_t = a_0 + a_1 t + a_2 t^2,$$

capturing curvature in the data. Higher-order polynomials ($p \leq 10$) allow flexibility for complex trends, while generalized least squares (available in softwares) adjusts for autocorrelated residuals. Though interpretable and useful for extrapolation, high p risks overfitting noise, and polynomials may diverge unrealistically outside the observed range. This method balances simplicity and mathematical rigor, ideal for data with clear polynomial trends when validated carefully.

Trend Elimination by Differencing

Trend Elimination by Differencing is a method used to remove polynomial trends from a time series by applying the **differencing operator** ∇. We define lag-1 differencing operator as:

$$\nabla X_t = X_t - X_{t-1} = (1 - B)X_t$$

Here, B_t is the backward shift operator $BX_t = X_{t-1}$ and so $B^j X_t = X_{t-j}$. Moreover, $\nabla^j(X_t) = \nabla(\nabla^{j-1}(X_t)), \quad j \geq 1$ with $\nabla^0(X_t) = X_t$.

If our $\{X_t\}$ has some linear trend $X_t = c_0 + c_1 t + Y_t$, then the first order difference:

$$\nabla X_t = X_t - X_{t-1} = c_1 + Y_t - Y_{t-1} = c_1 + \nabla Y_t$$

leaves us with a stationary process with mean c_1.

More generally, for a polynomial trend of degree k, applying k-th times differencing:

$$\nabla^k X_t = k!c_k + \nabla^k Y_t$$

will leave us with a stationary process $\nabla^k Y_t$ with mean $k!c_k$.

We can just visualize this with a time-series X_t having cubic trend:

$$X_t = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + Y_t$$

Applying first difference:

$$\nabla X_t = X_t - X_{t-1} = (c_0 + c_1 t + c_2 t^2 + c_3 t^3 + Y_t) - (c_0 + c_1(t-1) + c_2(t-1)^2 + c_3(t-1)^3 + Y_{t-1})$$

$$\nabla X_t = (c_1 - c_2 + c_3) + (2c_2 - 3c_3)t + 3c_3 t^2 + (Y_t - Y_{t-1})$$

Applying second difference:

$$\nabla^2(X_t) = \nabla(\nabla X_t) = ((c_1 - c_2 + c_3) + (2c_2 - 3c_3)t + 3c_3 t^2 + (Y_t - Y_{t-1}))$$

$$-((c_1 - c_2 + c_3) + (2c_2 - 3c_3)(t-1) + 3c_3(t-1)^2 + (Y_{t-1} - Y_{t-2}))$$

$$\nabla^2 X_t = (2c_2 - 6c_3) + 6c_3 t + \nabla Y_t - \nabla Y_{t-1} = (2c_2 - 6c_3) + 6c_3 t + \nabla^2 Y_t$$

Applying third difference:

$$\nabla^3 X_t = \nabla^2 X_t - \nabla^2 X_{t-1} = 6c_3 + \nabla^2 Y_t - \nabla^2 Y_{t-1}$$

We can see:

$$\nabla^3 X_t = 3!c_3 + \nabla^3 Y_t$$

Therefore, we say this process eliminates the cubic trend, leaving a stationary process $\nabla^3 Y_t$ with mean $3!c_3$.

To determine differencing order k, start with $k = 1$. If residuals still show trends, increase k. Differencing reduces the series length by k. For example, a series of length n becomes $n - k$ after differencing. Excessive differencing ($k > 2$) can introduce artificial correlations or invertible patterns. Differencing eliminates polynomial trends by transforming the series into a stationary process through repeated application of the ∇ operator.

Estimation and Elimination of Both Trend and Seasonality

When you have both a trend m_t and a seasonal component s_t in your series, we define our **classical decomposition model** as:

$$X_t = m_t + s_t + Y_t, \quad E[Y_t] = 0, \quad s_{t+d} = s_t, \quad \sum_{j=1}^{d} s_j = 0$$

the symbol d denotes the seasonal period, i.e. the number of observations in one full cycle of seasonality.

Estimation of Trend and Seasonal components

To estimate the trend and seasonal components in a time series using the classical decomposition method, one can follow these steps:

We estimate the trend $\widehat{m}_t$ using a moving average filter, its purpose is to remove seasonality and noise to isolate the trend.

$$\widehat{m}_t = \begin{cases} \frac{0.5x_{t-q} + \sum_{j=q-1}^{q+1} x_{t-j} + 0.5x_{t+q}}{d}, & q < t \leq n-q \text{ if d is even } d = 2q \\ \frac{1}{d}\sum_{j=-q}^{j=q} x_{t+j}, & q+1 < t \leq n-q \text{ if d is odd } d = 2q+1 \end{cases}$$

The first q and last q datapoints will have no trend estimates.

We then compute the deviations from the trend:

$$\delta_t = x_t - \widehat{m}_t$$

The edge points will have no trend estimates here also. We can use interpolation for visualtion but we keep them **NA** in initial computation.

Computing average deviations for each seasonal position k, for each $k = 1, 2, \cdots, d$

$$w_k = \frac{1}{N_k} \sum_j \delta_{k+jd}$$

N_k is the number of observations at poisition k. We now adjust seasonal indices to sum to zero and estimate seasonal component as:

$$\widehat{s}_k = \begin{cases} w_k - \frac{1}{d}\sum_{i=1}^{d} w_i, & k = 1, 2, \cdots, d \\ \widehat{s}_{k-d}, & k > d \end{cases}$$

We now remove this seasonal component to restimate the trend and this will now have trend values assigned for edge points as well.

$$d_t = x_t - \hat{s}_k, \quad t = 1, 2, \cdots, n$$

We reestimate the trend parametrically by fitting polynomial to the deseasonalized data $\{d_t\}$:

$$\hat{m}_t = \sum_{j=0}^{p} a_p t^p$$

In the end, we compute final residuals by $Y_t = x_t - \hat{m}_t - \hat{s}_t$ and our final model is now defined as:

$$X_t = \underbrace{a_0 + a_1 t + \cdots + a_p t^p}_{\textbf{Reestimated Trend}} + \underbrace{\hat{s}_t}_{\textbf{Seasonality}} + \underbrace{\hat{Y}_t}_{\textbf{Noise}}$$

The initial moving average trend is non-parametric and cannot be extrapolated. A parametric polynomial allows forecasting. Polynomial trends may overfit or diverge outside observed range. This method isolates interpretable components for analysis, forecasting, and anomaly detection.

Elimination of Trend and Seasonal Components by Differencing

To eliminate both trend and seasonal components from a time series using differencing, we will first remove the seasonality and then we go simply like we use differencing described earlier.

We first apply seasonal differencing by using the lag-d differencing operator to remove the seasonal component s_t. This removes seasonality by differencing observations separated by the seasonal period d.

$$\nabla_d X_t = X_t - X_{t-d} = (1 - B^d) X_t$$

After seasonal differencing, the series will become:

$$\nabla_d X_t = (m_t - m_{t-d}) + (Y_t - Y_{t-d})$$

It will have trend component as $m_t - m_{t-d}$ and have noise $Y_t - Y_{t-d}$.

If a trend remains in $\nabla_d X_t$, we can now eliminate trend using method described earlier. We apply the first-difference operator as:

$$\nabla(\nabla_d X_t) = \nabla_d X_t - \nabla_d X_{t-1}$$

$$= (m_t - m_{t-d} - m_{t-1} + m_{t-d-1}) + (Y_t - Y_{t-d} - Y_{t-1} + Y_{t-d-1})$$

$$= \nabla m_t - \nabla m_{t-d} + \nabla Y_t - \nabla Y_{t-d}$$

So, we can also write:

$$\nabla(\nabla_d X_t) = \nabla_d(\nabla m_t) + \nabla_d(\nabla Y_t) = \nabla_d(\nabla m_t + \nabla Y_t)$$

Therefore, we have:

$$\nabla(\nabla_d X_t) = \nabla_d(\nabla X_t)$$

This first-order differencing eliminates residual trend (e.g., linear trends become stationary after one differencing). Even for complex trends (e.g., quadratic), we can apply ∇^k until it leads us to stationary process with constant mean. This method transforms the original series into a stationary process by systematically removing trend and seasonality.

In time series analysis, the elimination of trend and seasonal components can be approached through two primary methods: classical decomposition and differencing. Classical decomposition involves estimating and subtracting explicit trend (m_t) and seasonal (s_t) components from the data, leaving a residual noise sequence (Y_t). This method is particularly useful when interpretable estimates of these components are required, such as identifying long-term trends or periodic patterns. Alternatively, differencing employs operators like ∇_d (seasonal differencing) and ∇ (regular differencing) to systematically remove trends and seasonality by transforming the series into a stationary process. For example, $\nabla_d X_t = X_t - X_{t-d}$ eliminates fixed-period seasonality, while subsequent differencing (∇) addresses residual trends. The choice between methods depends on factors such as the need for component estimates, the constancy of seasonal patterns, and the goal of forecasting. After applying either technique, it is critical to validate the resulting noise sequence for independence (e.g., via autocorrelation checks or hypothesis tests like Ljung-Box). If significant serial dependence persists, the noise can be modeled using stationary processes (e.g., ARIMA) to leverage its structure for improved forecasting.

Testing the Estimated Noise Sequence

After transforming a time series to remove trends and seasonality, the resulting residuals must be rigorously evaluated to determine if they exhibit properties of independent and identically distributed (iid) random variables. If the residuals show no significant dependence, they can be treated as unstructured noise, requiring only estimation of their mean and variance. However, if dependence is detected, further modeling of the residual structure such as autoregressive or moving average components, becomes necessary to leverage historical patterns for improved forecasting. Statistical tests for this purpose include analyzing autocorrelation patterns to identify serial dependence, evaluating fluctuations via **turning point counts** to detect excess volatility or rigidity, and applying trend-specific tests like **difference-sign or rank tests** to uncover monotonic or linear trends. **Normality checks**, such as quantile-quantile plots or squared correlation tests, assess whether residuals conform to a Gaussian distribution. If these tests collectively reject the iid hypothesis, the residuals warrant modeling as a stationary process to account for their inherent dependence, enabling more accurate predictions. Conversely, confirmation of iid behavior simplifies the analysis, indicating no further temporal structure to exploit.

Basic Properties of Stationary Processes

Non-Negative Definite Functions

A function $\kappa : \mathbb{Z} \to \mathbb{R}$ is nonnegative definite if for every positive integer n and every real vector $a = (a_1, \cdots, a_n)'$, the quadratic form satisfies:

$$\sum_{i,j=1}^{n} a_i \kappa(i-j) a_j \geq 0$$

This ensures that the matrix $[\kappa(i-j)]_{i,j=1}^{n}$ is positive semi-definite for all n.

Autocovariance Characterization

A real-valued function $\gamma : \mathbb{Z} \to \mathbb{R}$ is the autocovariance function of a stationary time series if and only if it satisfies:

1) Evenness: $\gamma(h) = \gamma(-h) \forall h \in \mathbb{Z}$.

2) Nonnegative Definiteness: γ is nonnegative definite.

Remarks:

1) $\rho(\cdot)$ is the ACF of a stationary process if and only if it is a normalized ACVF (i.e., $\rho(\cdot)$ is even, nonnegative definite, and $\rho(0) = 1$.

2) To verify that a given function is nonnegative definite it is often simpler to find a stationary process that has the given function as its ACVF.

Properties of Strictly Stationary Series $\{X_t\}$

1) The random variables X_t are identically distributed.

2) For all integers t and h, the joint distribution satisfies:

$$(X_t, X_{t+h}) \stackrel{d}{=} (X_1, X_{1+h})$$

3) **Weak Stationarity Under Finite Variance:** If $E(X_t^2) < \infty$ for all t, the series is weakly stationary.

4) Weak stationarity does not imply strict stationarity.

5) An independent and identically distributed **(iid)** sequence is strictly stationary.

Weak stationarity preserves means and covariances under time shifts while **strict stationarity** preserves entire joint distributions under time shifts.

Constructing Strictly Stationary Time Series via Filtering:

To build a strictly stationary time series $\{X_t\}$, one can apply a filter to an iid sequence $\{Z_t\}$. Define:

$$X_t = g(Z_t, Z_{t-1}, \cdots, Z_{t-q})$$

where g is a fixed real-valued function. Since $\{Z_t\}$ is iid (and thus strictly stationary), shifting the time indices by any h preserves the distribution of $\{X_t\}$, ensuring strict stationarity. We can consider following properties as well:

1) $\{X_t\}$ is q-Dependence, when observations X_s and X_t are independent if $\mid t-s \mid > q$. As then both of the X_t and X_s won't come in the definition of each other defined through $g(\cdot)$.

2) A stationary series is q-correlated if its autocovariance $\gamma(h) = 0$ for $\mid h \mid > q$.

3) Every q-correlated process can be represented as a **moving-average process of order** q **(MA(q))**.

A white noise is 0-correlated.

The MA(q) Process

A **moving-average process** of order q (MA(q)) is defined as:

$$X_t = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ (white noise with mean 0 and variance σ^2) and $\theta_1, \cdots, \theta_q$ are constants. This model expresses X_t as a weighted sum of the current and past q noise terms. **An MA(q) process is a linear combination of current and past white noise terms.**

Proposition 1:

If a stationary time series $\{X_t\}$ has mean zero and is q-correlated (i.e., its autocovariance $\gamma(h) = 0$ for all $\mid h \mid > q$), then it can be represented as an MA(q) process.

Every stationary q-correlated series with mean zero is structurally equivalent to an MA(q) model. This allows modeling such series using the MA(q) framework, simplifying forecasting and analysis.

Linear Processes

A time series $\{X_t\}$ is termed a linear process if it can be expressed as a doubly infinite weighted sum of white noise terms:

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j}$$

where $\{\psi_j\}$ are coefficients satisfying $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$.

Properties of Linear Processes

1.) Using the operator B (where $B^k Z_t = Z_{t-k}$), the process is compactly written as:

$$X_t = \psi(B) Z_t, \quad \textbf{with } \psi(B) = \sum_{j=-\infty}^{\infty} \psi_j B^j$$

This operator $\psi(B)$ can be thought of as a **linear filter** as well.

2.) If $\psi_j = 0$ for all $j < 0$, the process becomes a moving average of infinite order (MA(∞)):

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j}$$

Since, $\{Z_t\}$ comes from white noise, we can see the convergence conditions fulfilled for absolute summability and square summability. $\sum |\psi_j| < \infty$ and $\sum \psi_j^2 < \infty$ ensures the series converges almost surely and in mean square (as $E|Z_t| < \sigma \implies E|X_t| < \infty$).

Proposition 2

Let $\{Y_t\}$ be a stationary time series with mean 0 and autocovariance function γ_Y. If the coefficients $\{\psi_j\}$ satisfy $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$, then the filtered series:

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Y_{t-j} = \psi(B) Y_t$$

is also stationary with mean 0 and autocovariance function:

$$\gamma_X(h) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \psi_j \psi_k \gamma_Y(h + k - j)$$

Special Case (Linear Process): If $\{Y_t\}$ is white noise ($\{Z_t\} \sim WN(0, \sigma^2)$, the autocovariance simplifies to:

$$\gamma_X(h) = \sigma^2 \sum_{j=-\infty}^{\infty} \psi_j \psi_{j+h}$$

Proof:

It is given to us that $\{Y_t\}$ is stationary, so $E[Y_t] \leq \sqrt{\gamma_Y(0)}$ and also the coefficients satisfy: $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$ so

$$E|X_t| = E|\sum_{j=-\infty}^{\infty} \psi_j Y_{t-j}| \leq \sum_{j=-\infty}^{\infty} |\psi_j| |E[Y_{t-j}| \leq \left(\sum_{j=-\infty}^{\infty} |\psi_j| \right) \sqrt{\gamma_Y(0)} < \infty$$

So, this ensures that the infinite sum in X_t converges with probability 1. Since $\sum_{j=-\infty}^{\infty} \psi_j^2 < \infty$ will also hold, this will ensure that the series converges in mean

square. We can also say that X_t is infact the mean square limit of the partial sums $\sum_{j=-\infty}^{\infty} \psi_j Y_{t-j}$.

Since $E[Y_t] = 0$, so

$$E[X_t] = E[\sum_{j=-\infty}^{\infty} \psi_j Y_{t-j}] = \sum_{j=-\infty}^{\infty} \psi_j E[Y_{t-j}] = 0$$

Since $E[X_{t+h}] = E[X_t] = 0$, the autocovariance $\gamma_X(h) = Cov(X_{t+h}, X_t)$ is computed as:

$$\gamma_X(h) = E\left[\left(\sum_{j=-\infty}^{\infty} \psi_j Y_{t+h-j}\right), \left(\sum_{k=-\infty}^{\infty} \psi_k Y_{t-k}\right)\right]$$

$$= \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \psi_j \psi_k E(Y_{t+h-j}, Y_{t-k})$$

Since autocovariance of $\{Y_t\}$ is γ_Y, $E(Y_{t+h-j}, Y_{t-k}) = \gamma_Y(h - j + k)$ and so:

$$\gamma_X(h) = \sum_{j=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} \psi_j \psi_k \gamma_Y(h - j + k)$$

The autocovariance of $\{X_t\}$ depends only on the lag h, confirming stationarity.

Now, for **special case**:

If $\{Y_t\}$ is white noise ($\{Y_t\} \sim WN(0, \sigma^2)$, then $\gamma_Y(h - j + k) = \sigma^2$, if $k = j - h$ and 0 otherwise.

So by symmetry of the summation over all the integers:

$$\gamma_X(h) = \sigma^2 \sum_{j=-\infty}^{\infty} \psi_j \psi_{j+h}$$

Filtering a stationary series with an absolutely summable linear operator preserves stationarity.The autocovariance of the output is a convolution of the filter weights and input autocovariance.

Remark:

When applying linear filters to a stationary time series $\{Y_t\}$, the combined effect of two filters can be expressed as a single equivalent filter. Let$\alpha(B) = \sum_{j=-\infty}^{\infty} \alpha_j B^j$ and $\beta(B) = \sum_{j=-\infty}^{\infty} \beta_j B^j$ be linear filters where $\sum |\alpha_j| < \infty$ and

$\sum |\beta_j| < \infty$. Applying $\alpha(B)$ and $\beta(B)$ successively to $\{Y_t\}$ generates a new stationary series:

$$W_t = \sum_{j=-\infty}^{\infty} \psi_j Y_{t-j}$$

where the coefficients ψ_j are the convolution of α_j and β_j:

$$\psi_j = \sum_{k=-\infty}^{\infty} \alpha_k \beta_{j-k} = \sum_{k=-\infty}^{\infty} \beta_k \alpha_{j-k}$$

The combined filter is equivalently written as:

$$\psi(B) = \alpha(B)\beta(B) = \beta(B)\alpha(B)$$

Combining linear filters via convolution preserves stationarity,and the order of filtering is irrelevant due to commutativity.

AR(1) Process

We define AR(1) process as a solution of:

$$X_t - \phi X_{t-1} = Z_t, \quad \{Z_t\} \sim WN(0, \sigma^2) \tag{1}$$

We will now see for what value of ϕ, we can expect a stationary solution of Equation 1.

Case $|\phi| < 1$: On iterating the AR(1) equation backward will yield:

$$X_t = Z_t + \phi Z_{t-1} + \phi^2 Z_{t-2} + \cdots = \sum_{j=0}^{\infty} \phi^j Z_{t-j}$$

The coefficients ϕ^j decay geometrically (since $|\phi| < 1$), ensuring the series converges absolutely and in mean square. Using proposition 2, $\{X_t\}$ is stationary with mean 0 and autocovariance:

$$\gamma_X(h) = \sum_{j=0}^{\infty} \phi^j \phi^{j+h} \sigma^2 = \frac{\sigma^2 \phi^{|h|}}{1 - \phi^2}, \quad h \in \mathbb{Z}$$

Let $\{Y_t\}$ be any stationary solution, then:

$$Y_t = \phi Y_{t-1} + Z_t = Z_t + \phi Z_{t-1} + \phi^2 Y_{t-2}$$

$$= Z_t + \phi Z_{t-1} + \phi^2 Z_{t-2} + \phi^3 Y_{t-3}$$

$$= \sum_{j=0}^{k} \phi^j Z_{t-j} + \phi^{k+1} Y_{t-k-1}$$

$$Y_t - \sum_{j=0}^{k} \phi^j Z_{t-j} = \phi^k Y_{t-k-1}$$

$$E[Y_t - \sum_{j=0}^{k} \phi^j Z_{t-j}]^2 = E[\phi^k Y_{t-k-1}]^2$$

Since Y_t is stationary, hence $E[Y]^2$ is finite and independent of t, so

$$E[Y_t - \sum_{j=0}^{k} \phi^j Z_{t-j}]^2 = E[\phi^k Y_{t-k-1}]^2 \to 0 \ \textbf{as} \ k \to 0$$

Hence, Y_t is equal to the mean square limit $\sum_{j=0}^{\infty} \phi^j Z_{t-j}$ and hence $X_t \equiv Y_t$ and so, the unique stationary solution of the equation Equation 1 for $|\phi| < 1$ is given by:

$$X_t = \sum_{j=0}^{\infty} \phi^j Z_{t-j}$$

Case $|\phi| > 1$:

Rewriting the equation Equation 1 as:

$$X_t = \phi^{-1} Z_{t+1} + \phi^{-1} X_{t+1}$$

Iterating forward will give:

$$X_t = -\sum_{j=1}^{\infty} \phi^{-j} Z_{t+j}$$

Coefficients ϕ^{-j} decay as $j \to \infty$ (since $|\phi| > 1$), ensuring convergence. The solution is stationary but not physically meaningful for forecasting or modeling because X_t defined here depends on the future values of noise. It is customary therefore in modeling stationary time series to restrict attention to AR(1) processes with $|\phi| < 1$.

Case $\phi = \pm 1$:

Equation Equation 1 will become:

$$X_t - (\pm 1)X_{t-1} = Z_t$$

Iterating the equation will lead to cumulative noise terms:

$$X_t = X_0 + \sum_{j=1}^{t} Z_j$$

resulting in unbounded variance as $t \to \infty$.

AR(1) processes are stationary and causal only if $|\phi| < 1$. For $|\phi| > 1$, solutions exist but are non-causal and impractical. No stationary solution exists if $|\phi| = 1$.

Remark

We can also derive the stationary solution using operator algebra:

For $|\phi| < 1$ and backward shift operator B:

$$\sum_{j=0}^{\infty} \phi^j B^j = \frac{1}{1 - \phi B}$$

So, if we set: \$ (B) = 1- B and $\pi(B) = \sum_{j=0}^{\infty} \phi^j B^j$, then

$$\phi(B)\pi(B) = 1$$

Now, using backward shift operator B in Equation 1 ,

$$\phi(B)X_t = (1 - \phi B)X_t = Z_t$$

applying $\pi(B)$ both sides:

$$\pi(B)\phi(B)X_t = \pi(B)Z_t$$

$$X_t = \pi(B)Z_t = \sum_{j=0}^{\infty} \phi^j B^j Z_t = \sum_{j=0}^{\infty} \phi^j Z_{t-j}$$

This method **leverages the invertibility** of $\phi(B)$ (guaranteed when $|\phi| < 1$) to directly express X_t as a causal linear process of past and present white noise terms.

ARMA Processes

Definition of ARMA(1,1) process

A time series $\{X_t\}$ is an ARMA$(1,1)$ process if it is stationary and satisfies:

$$X_t - \phi X_{t-1} = Z_t + \theta Z_{t-1}, \quad Z_t \sim WN(0, \sigma^2)$$

where ϕ (autoregressive coefficient) and θ (moving average coefficient) are constants while keeping $\phi \neq \theta$ to avoid redundancy in the model.

Using the **backshift operator** $B(BX_t = X_{t-1})$, this becomes:

$$(1 - \phi B)X_t = (1 + \theta B)Z_t \quad \textbf{or} \quad \phi(B)X_t = \theta(B)Z_t$$

where $\phi(B) = 1 - \phi B$ and $\theta(B) = 1 + \theta B$.

Case $|\phi| < 1$:

The operator $\phi(B)$ is invertible. Its inverse is:

$$\chi(B) = \frac{1}{1 - \phi B} = \sum_{j=0}^{\infty} \phi^j B^j$$

which converges absolutely for $|\phi| < 1$. On applying $\chi(B)$ to both sides of $\phi(B)X_t = \theta(B)Z_t$, we get :

$$X_t = \chi(B)\theta(B)Z_t = \left(\sum_{j=0}^{\infty} \phi^j B^j\right)(1 + \theta B)Z_t$$

Expanding this will give us:

$$X_t = Z_0 + (\phi + \theta)\sum_{j=1}^{\infty} \phi^{j-1} Z_{t-j}$$

Using the same approach that we used in AR(1) process, we can see that this is unique stationary solution. The solution is **casual** as X_t depends only on current and past noise terms $\{Z_t - j, j \geq 0\}$.

Case $|\phi| > 1$:

Now, we will rewrite $\phi(B)^{-1}$ using forward shifts (negative powers of B):

$$\frac{1}{\phi(B)} = -\sum_{j=1}^{\infty} \phi^{-j} B^{-j}$$

On applying this operator to $\theta(B)Z_t$, we get:

$$X_t = -\theta\phi^{-1}Z_t - (\theta+\phi)\sum_{j=1}^{\infty}\phi^{-j-1}Z_{t+j}$$

We can use the same logic like the previous case as here $|\phi^{-1}| < 1$ and so stationary solution exists but it is **non-casual** as X_t depends on future noise terms $\{Z_{t+j}, j \geq 1\}$, making it impractical for real-world applications.

Case $\phi = \pm 1$: Following the same approach, our model will lead to unbounded variance over time and hence it won't be stationary.

ARMA$(1,1)$ **has a unique stationary solution if** $|\phi| \neq 1$:

$$\textbf{For } |\phi| < 1: \quad X_t = Z_0 + (\phi+\theta)\sum_{j=1}^{\infty}\phi^{j-1}Z_{t-j} \textbf{ (casual)}$$

$$\textbf{For } |\phi| > 1: \quad X_t = -\theta\phi^{-1}Z_t - (\theta+\phi)\sum_{j=1}^{\infty}\phi^{-j-1}Z_{t+j} \textbf{ (non-casual)}$$

$$\textbf{For } |\phi| = 1: \textbf{ No stationary solution exists}$$

Invertibility:

A process is invertible if the white noise term Z_t can be expressed as a function of current and past observations $X_s, s \leq t$. This allows recovering the noise sequence from observed data. Invertibility depends on the moving average coefficient θ, analogous to how causality depends on the autoregressive coefficient ϕ. Just like we do calculation of casuality in the case of ϕ, we can do the same for θ and it is straight forward to show that:

ARMA$(1,1)$ **is invertible if and only if** $|\theta| < 1$ **(noise depends on past** X_t**). If** $|\ \theta\ | > 1$**, it is non-invertible (noise depends on future** X_t**). Invertibility ensures recoverability of noise terms for practical modeling.**

Noncausal/noninvertible ARMA$(1,1)$ **processes can be reparameterized using a new white noise sequence** $\{W_t\}$ **to become causal and invertible without losing statistical properties (mean, variance, autocovariance). Thus, focusing on causal/invertible models suffices for analysis, even for higher-order ARMA models.**

Properties of the sample mean and autocorrelation function

Estimation of Sample mean μ

Given a stationary process $\{X_t\}$, we can compute the sample mean as:

$$\bar{X_n} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

It will be an unbiased estimator of μ since

$$E[\bar{X_n}] = \frac{1}{n}(E(X_1) + \cdots + E(X_n)) = \mu$$

The variance (or mean squared error, as the estimator is unbiased) is:

$$Var(\bar{X}_n\ ) = E[(\bar{X}_n\ - \mu)^2] = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} Cov(X_i, X_j)$$

For a stationary process, $Cov(X_i, X_j) = \gamma(i-j)$, where $\gamma(h)$ is the autocovariance at lag h. Let $h = i - j$. The double sum becomes:

$$Var(\bar{X}_n) = \frac{1}{n^2}\sum_{h=-n+1}^{n-1}(n - |h|)\gamma(|h|)$$

For each lag h, there are $n - |h|$ pairs (i, j) such that $i - j = h$. We now adjust the summation limits by extending the sum to include $h = \pm n$ (terms vanish because $n - |h| = 0$):

$$Var(\bar{X}_n) = \frac{1}{n^2}\sum_{h=-n}^{n}(n - |h|)\gamma(|h|)$$

Therefore:

$$Var(\bar{X}_n) = \frac{1}{n}\sum_{h=-n}^{n}(1 - \frac{|h|}{n})\gamma(|h|)$$

The term $(1 - \frac{|h|}{n})$ acts as a weight for each lag h, decaying linearly with $|h|$. Lags closer to 0 contribute more due to more observation pairs, while distant lags contribute less.

Proposition 3

If $\{X_t\}$ is a stationary time series with mean μ and autocovariance function $\gamma(\cdot)$, then as $n \to \infty$,

$$Var(\bar{X}_n) = \mathbb{E}(\bar{X}_n - \mu)^2 \longrightarrow 0 \quad \text{if } \gamma(n) \to 0$$

$$n\mathbb{E}(\bar{X}_n - \mu)^2 \longrightarrow \sum_{h=-\infty}^{\infty} \gamma(h) \quad \text{if } \sum_{h=-\infty}^{\infty} |\gamma(h)| < \infty$$

For large n, under Gaussianity or linearity (e.g., ARMA models), the Central Limit Theorem for dependent data applies:

$$\sqrt{n}\,(\bar{X}_n - \mu) \xrightarrow{d} \mathcal{N}(0, v)$$

where

$$v = \sum_{h=-\infty}^{\infty} \gamma(h)$$

is the long-run variance, capturing the cumulative effect of autocorrelations. This leads to:

$$\bar{X}_n \approx \mathcal{N}(\mu, \frac{v}{n})$$

An approximate 95% confidence interval for μ is:

$$\left(\bar{X}_n - 1.96\sqrt{\frac{\hat{v}}{n}}, \bar{X}_n + 1.96\sqrt{\frac{\hat{v}}{n}}\right)$$

where $\hat{v}$ estimates v. The estimator $\hat{v}$ accounts for autocovariances up to lag $\sqrt{n}$ with **Bartlett weights**:

$$\hat{v} = \sum_{|h|<\sqrt{n}} \left(1 - \frac{|h|}{n}\right) \hat{\gamma}(h)$$

Truncation at $\sqrt{n}$ balances bias (including sufficient lags) and variance (excluding noisy estimates at large lags). These weights $(1 - \frac{|h|}{n})$ reduces the influence of higher lags, akin to the Bartlett kernel in HAC estimators.

Estimation of Autocovariance $\gamma(\cdot)$ and Autocorrelation $\rho(\cdot)$ Functions

We will compute ACVF $\hat{\gamma}(h)$ and ACF $\hat{\rho}(h)$ as:

$$\hat{\gamma}(h) = n^{-1} \sum_{i=1}^{n-|h|} (X_{i+|h|} - \bar{X}_n)(X_i - \bar{X}_n)$$

and

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$$

these estimators quantify the covariance and correlation between observations at lag h. While $\hat{\gamma}(h)$ and $\hat{\rho}(h)$ are biased (even if normalized by $n-|h|$ instead of n), they become nearly unbiased for large n. A critical property of $\hat{\gamma}(h)$ is that the k-dimensional sample covariance matrix:

$$\hat{\Gamma}_k = \begin{bmatrix} \hat{\gamma}(0) & \hat{\gamma}(1) & \cdots & \hat{\gamma}(k-1) \\ \hat{\gamma}(1) & \hat{\gamma}(0) & \cdots & \hat{\gamma}(k-2) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{\gamma}(k-1) & \hat{\gamma}(k-2) & \cdots & \hat{\gamma}(0) \end{bmatrix}$$

is **nonnegative definite**. This ensures valid covariance structures, as negative eigenvalues would imply implausible negative variances.

The matrix $\hat{\Gamma}_k$ is constructed using a transformation matrix T of size $k \times 2k$, where each row contains lagged deviations $Y_i = X_i - \bar{X}_n$ (with $Y_i = 0$ for $i > n$). For any real vector a:

$$a'\hat{\Gamma}_k a = n^{-1}(a'T)(T'a) \geq 0$$

which guarantees nonnegative definiteness.

Using n^{-1} (instead of $(n-|h|)^{-1}$) preserves nonnegative definiteness but introduces bias. For example, the divisor n ensures structural validity of $\hat{\Gamma}_k$, while $(n-|h|)^{-1}$ adjusts for reduced pairs at larger lags but risks invalid covariance matrices.

The normalized matrix (Sample Autocorrelation Matrix) :

$$\hat{R}_k = \frac{\hat{\Gamma}_k}{\hat{\gamma}(0)}$$

The estimators $\hat{\gamma}(h)$ and $\hat{\rho}(h)$ balance bias and structural integrity. While slightly biased, their design ensures covariance/correlation matrices remain valid, enabling reliable statistical inference in time series analysis.

Sample ACVF and ACF estimators prioritize nonnegative definiteness over bias reduction.Using n^{-1} ensures valid covariance matrices, critical for model identification and inference.

The matrices $\hat{\Gamma}_k$ and $\hat{R}_k$ are in fact nonsingular if there is at least one nonzero Y_i, or equivalently if $\hat{\gamma}(0) > 0$.

The standard methods suggest n should be atleast 50 and $h \leq n/4$.

For systematic inference concerning $\rho(h)$, understanding the sampling distribution of the sample ACF estimator $\hat{\rho}(h)$ is essential. While exact distributions are complex, asymptotic theory provides a practical approximation: for large sample sizes n, the vector $\hat{\rho}_k = (r\hat{h}o(1), \cdots, \hat{\rho}(k))'$ follows a multivariate normal distribution. Specifically, under linear model assumptions (e.g., ARMA models),

$$\hat{\rho}_k \approx N(\rho_k, n^{-1}W)$$

where $\rho_k = (\rho(1), \cdots, \rho(k))'$ contains the true autocorrelations, and W is the covariance matrix defined by Bartlett's formula. The (i, j)-th element of W is given by:

$$w_{ij} = \sum_{k=-\infty}^{\infty} \{\rho(k+i)\rho(k+j) + \rho(k-i)\rho(k+j) + 2\rho(i)\rho(j)\rho^2(k) - 2\rho(i)\rho(k)\rho(k+j) - 2\rho(j)\rho(k)\rho(k+i)\}$$

This expression simplifies computationally to a finite sum:

$$w_{ij} = \sum_{k=1}^{\infty} \{\rho(k+i) + \rho(k-i) - 2\rho(i)\rho(k)\} \{\rho(k+j) + \rho(k-j) - 2\rho(j)\rho(k)\}$$

The covariance structure W captures dependencies between autocorrelation estimates at different lags, which is vital for constructing confidence intervals and testing hypotheses (e.g., whether multiple autocorrelations jointly differ from zero).

Example 1: IID Noise

Consider a time series $\{X_t\} \sim IID(0, \sigma^2)$, where each observation has mean 0, variance σ^2, and no serial dependence. For such a process, the true autocorrelation function (ACF) satisfies:

$$\rho(h) = \begin{cases} 1 & \textbf{if} \quad h = 0 \\ 0 & \textbf{otherwise} \end{cases}$$

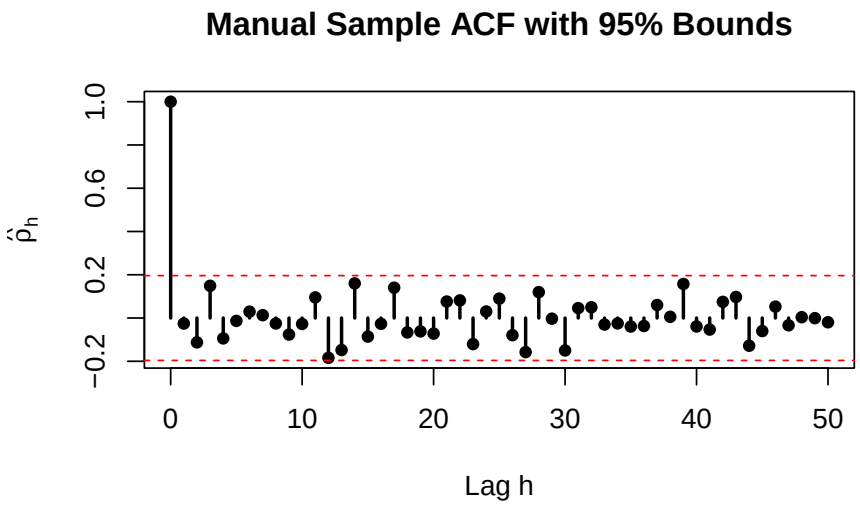
Since $\rho(k) = 0$ for all $k \neq 0$, every term in the sum vanishes except when $i = j$, leading to:

$$w_{ij} = \begin{cases} 1 & \textbf{if} \quad i = j \\ 0 & \textbf{otherwise} \end{cases}$$

Thus, W reduces to the identity matrix.

We take lags $h = 1, 2, \cdots, k$ on X-axis and values of $\hat{\rho}(h)$ on Y-axis. We draw two dashed lines at $y = \pm 1.96/n$. We now plot $\hat{\rho}(h)$ as vertical bars at each lag h.

```r
set.seed(123)
n     <- 100      # sample size
sigma <- 1        # noise sd
nlags <- 50       # number of lags to compute
X <- rnorm(n, mean = 0, sd = sigma)
Xbar  <- mean(X)
denom <- sum((X - Xbar)^2)
rho <- numeric(nlags + 1)
for (h in 0:nlags) {
  # numerator: covariance at lag h
  num <- sum((X[(1+h):n] - Xbar) * (X[1:(n-h)] - Xbar))
  rho[h+1] <- num / denom
}
bound <- qnorm(0.975) / sqrt(n)   # same as 1.96/sqrt(n)
plot(0:nlags, rho,
     type = "h",          # vertical lines
     lwd  = 2,
     xlab = "Lag h",
     ylab = expression(hat(rho)[h]),
     main = "Manual Sample ACF with 95% Bounds")
points(0:nlags, rho, pch = 16)    # add dots
abline(h = c(+bound, -bound),
       col = "red", lty = 2)
```

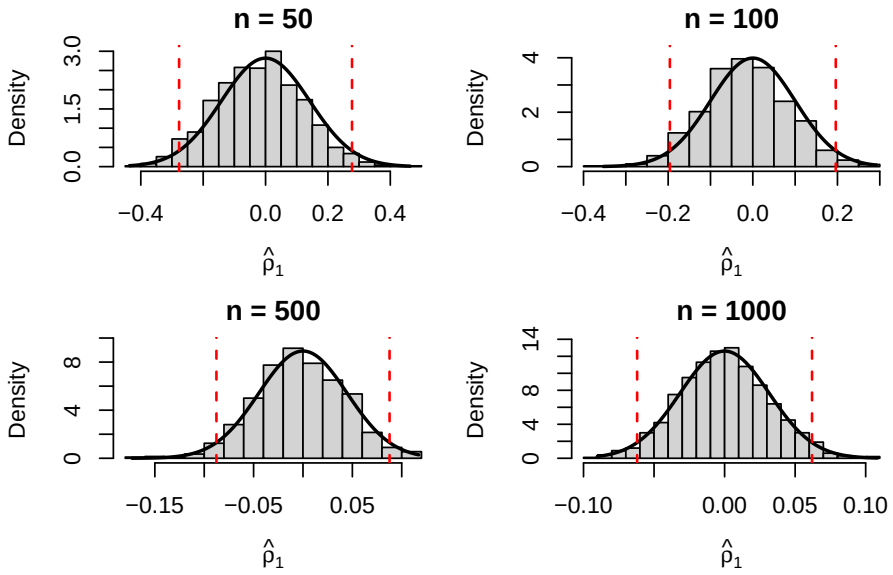


Approximately 95% of $\hat{\rho}(h)$ values lie within the confidence bounds. No systematic pattern (e.g., decaying trends or spikes) appeared.

We now see the verification that the sample ACF converges to the theoretical normal distribution as the number of points n in the time series increases.

```r
set.seed(123)
n_vec <- c(50, 100, 500, 1000)  # different series lengths
M     <- 1000                   # number of simulations per n
h     <- 1                      # lag to examine
old_par <- par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

for (n in n_vec) {
  # 3a. Simulate M series and compute rho_hat(h) manually
  rho_vals <- replicate(M, {
    x    <- rnorm(n)           # iid N(0,1)
    xbar <- mean(x)
    num  <- sum((x[(1+h):n] - xbar) * (x[1:(n-h)] - xbar))
    den  <- sum((x - xbar)^2)
    num / den
  })

  # 3b. Plot histogram of empirical rho
  hist(rho_vals,
       breaks = 15,
       freq   = FALSE,
       main   = paste0("n = ", n),
```

```
       xlab   = expression(hat(rho)[1]),
       ylim   = c(0, max(density(rho_vals)$y)*1.1))

  # 3c. Overlay theoretical N(0, 1/n) density
  curve(dnorm(x, mean = 0, sd = 1/sqrt(n)),
        from = min(rho_vals),
        to   = max(rho_vals),
        add  = TRUE,
        lwd  = 2)

  # 3d. Add ±1.96/√n bounds
  abline(v = c(-1.96/sqrt(n), 1.96/sqrt(n)),
         col = "red", lty = 2, lwd = 1.5)
}
```




```
# 4. Reset plotting parameters
par(old_par)
```

As $n \uparrow, \hat{\rho}(h)$ clusters closer to 0. The distribution becomes Gaussian. The spread of $\hat{\rho}(h)$ diminishes proportionally to $1/n$.

Example 2: An MA(1) Process

For an MA(1) process, the true autocorrelation function (ACF) satisfies:

$$\rho(h) = \begin{cases} 1 & \textbf{if } h = 0 \\ \frac{\theta}{1+\theta^2} & \textbf{if } h = \pm 1 \\ 0 & \textbf{if } |h| > 1 \end{cases}$$

GARCH Process

Let $\{r_t\}_{t\in\mathbb{Z}}$ be a (zero-mean) return series, or demeaned log-returns of an asset. We write

$$r_t = \epsilon_t$$

where the innovations ϵ_t decompose as:

$$\epsilon_t = \sigma_t z_t$$

with $z_t \sim IID(0,1)$ (often gaussian or student-t), $\sigma_t^2 = Var(\epsilon_t|\mathcal{F}_{t-1})$ is the conditional variance, and $\mathcal{F}_{t-1}$ is the $\sigma-$ algebra generated by $\{r_{t-1}, r_{t-2}, \cdots\}$.

Definition: GARCH(p,q)

A **GARCH** (p,q)model posits that

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2$$

subject to

$$\alpha_0 > 0, \;\; \alpha_i \geq 0, \;\; \beta_j \geq 0, \;\; \sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$$

α_i measure the immediate "ARCH" effects (past shocks), β_j capture the "GARCH" effects (persistence of past variances) and the stationary condition $\sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$ ensures a finite unconditional variance.

LSTM Neural Network

An **LSTM (Long Short-Term Memory**) is a type of recurrent neural network tailored for sequences, especially time series, by using a memory cell and three gates (**forget, input, output**) to control information flow. This architecture lets it learn long-range dependencies without suffering from vanishing or exploding gradients. In practice, we first:

- Window our series into input–target pairs and normalize them. Then,
- Stack one or more LSTM layers (possibly bidirectional) and add dense output layers. And finally, we
- Train (e.g., with Adam) on a regression loss (like MSE) for forecasting.

Their strength lies in capturing both short-term fluctuations and long-term trends, making LSTMs a go-to for time-series prediction.

Comparing Forecasting through ARMA-GARCH with LSTM on Time-Series Data

Financial time series, such as daily closing prices of publicly traded stocks, exhibit complex patterns including autocorrelation, volatility clustering, and non-stationarity. Classical approaches like ARMA (AutoRegressive Moving Average) augmented with a GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) error model are well-established for capturing linear dependencies and time-varying volatility. More recently, deep learning models, particularly Long Short-Term Memory (LSTM) networks, have shown promise in learning nonlinear patterns and long-range dependencies directly from raw data.

In this section, we compare the forecasting performance of an LSTM network against an ARMA(1,1)–GARCH(1,1) pipeline on daily closing prices of Yahoo Inc. from November 23, 2015 to November 20, 2020. Our objectives are to:

1. Assess whether the LSTM's ability to model complex, non-linear dynamics translates into more accurate short-term forecasts.
2. Evaluate how well a classical ARMA-GARCH approach handles linear trends and volatility clusters compared to the LSTM.
3. Understand the trade-offs between model interpretability, computational cost, and predictive accuracy.

We begin by outlining our data preprocessing pipeline then detail the architecture and training regimen for our LSTM model. Next, we describe the selection

of ARMA and the fitting of a GARCH error structure to capture conditional heteroskedasticity. Forecasts from both models are evaluated over an out-of-sample test set using standard error metrics (MSE, MAE, RMSE, and MAPE). Finally, we discuss the empirical results and their implications for financial forecasting practice.

LSTM Model

Data Preprocessing

To prepare the daily Yahoo stock data for our forecasting comparison, we first imported the raw CSV into a time-indexed DataFrame and focused on five core series—High, Low, Open, Volume, and Close. In the first approach, we experimented with different look-back windows (time steps of 5, 10, 20, 40, and 80 days) to assess how varying historical context affects predictive performance. After converting the Date column to a `DateTimeIndex`, we extracted the feature matrix and target vector (the next-day Close price). All feature columns were scaled to the [0, 1] range using a Min-Max transformation to ensure uniformity across inputs. We then slid a window of length t (for each chosen time-step) over the normalized data, constructing sequences of t consecutive feature vectors and pairing each sequence with the corresponding next-day Close value. Finally, the resulting dataset was split chronologically into an 80 percent training set and a 20 percent hold-out test set.

```
{python}
time_step = [5, 10, 20, 40, 80]
# Load data ONCE outside the loop
df = pd.read_csv('yahoo_stock.csv')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
features = df[['High', 'Low', 'Open', 'Volume', 'Close']]
target = df['Close'].values.reshape(-1, 1)

# Normalization
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(features)

# Create sequences
X, y = [], []
for i in range(len(scaled_data) - time_steps):
  X.append(scaled_data[i:i+time_steps])
  y.append(scaled_data[i+time_steps, -1])

X, y = np.array(X), np.array(y)

# Train-test split
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

In the second preprocessing scheme, we fixed the look-back window at 5 days and instead varied the richness of input information by incrementally adding new technical indicators. Starting again with the basic five features, we computed a suite of momentum and volatility measures: a 20-day simple moving

average, 12- and 26-day exponential moving averages (for MACD), a 14-day Relative Strength Index, upper and lower Bollinger Bands (±2 standard deviations around the 20-day SMA), and a five-day momentum series. We dropped any rows containing NaNs arising from indicator calculations to maintain data integrity. From this enhanced feature set, we selected subsets of increasing cardinality (8, 9, 11, and 14 features) to investigate how additional derived variables influence model accuracy. Each subset was normalized via Min-Max scaling, then converted into five-day input sequences paired with the next-day Close price. As before, we partitioned the sequences into 80 percent training and 20 percent test splits.

```
{python}
# Calculate technical indicators
df['SMA_20'] = df['Close'].rolling(20).mean()
df['EMA_12'] = df['Close'].ewm(span=12, adjust=False).mean()
df['EMA_26'] = df['Close'].ewm(span=26, adjust=False).mean()
# RSI Calculation
delta = df['Close'].diff()
gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)
avg_gain = gain.rolling(14).mean()
avg_loss = loss.rolling(14).mean()
rs = avg_gain / avg_loss
df['RSI'] = 100 - (100 / (1 + rs))
df['MACD'] = df['EMA_12'] - df['EMA_26']
df['Signal_Line'] = df['MACD'].ewm(span=9, adjust=False).mean()
df['Upper_BB'] = df['SMA_20'] + 2*df['Close'].rolling(20).std()
df['Lower_BB'] = df['SMA_20'] - 2*df['Close'].rolling(20).std()
# Momentum
df['Momentum_5'] = df['Close'] - df['Close'].shift(5)
df = df.dropna()
totalfeatures = df[['High', 'Low', 'Open', 'Volume', 'Close', 'SMA_20','EMA_12',
'EMA_26', 'RSI', 'MACD', 'Signal_Line','Upper_BB', 'Lower_BB','Momentum_5']]
time_steps = 5
selected_columns = totalfeatures.columns.tolist()[:num_features] #num_features will
be features we want to use
features = totalfeatures[selected_columns]
close_idx = selected_columns.index('Close')
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(features)
X, y = [], []
for j in range(len(scaled_data) - time_steps):
  X.append(scaled_data[j:j + time_steps])
  y.append(scaled_data[j + time_steps, close_idx])  # Use correct Close index
X, y = np.array(X), np.array(y)
split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

This two-pronged preprocessing strategy allows us both to gauge the effect of memory length and to evaluate the incremental value of technical indicators in forecasting performance.

Figure 2: Yahoo Stock data

Model Training and Forecasting

For both preprocessing schemes, we employed an identical LSTM architecture to ensure a fair comparison. The network begins with a 50-unit LSTM layer that returns the full sequence of hidden states, allowing the model to capture temporal patterns across the entire look-back window. A dropout layer with a 20 percent rate follows to reduce overfitting by randomly omitting a fraction of the LSTM's outputs during training. A second 50-unit LSTM layer then distills the sequence into a single vector representation, which again passes through a 20 percent dropout. Finally, a dense layer with a single neuron produces the one-step-ahead forecast for the closing price. We optimized the model using the Adam algorithm and minimized mean squared error (MSE) as our loss function.

Training proceeded for up to 200 epochs with a batch size of 32, but we incorporated early stopping based on validation loss: if the validation MSE did not improve for ten consecutive epochs, training halted and the model reverted to the weights that achieved the lowest validation error. We reserved 20 percent of the training data for validation at each epoch, enabling us to monitor generalization performance and guard against overfitting. Throughout training, we recorded the loss curves for both training and validation sets to visualize convergence behavior and to confirm that neither underfitting nor overfitting dominated.

Once training completed, we generated out-of-sample forecasts by feeding the test sequences into the trained LSTM and obtaining predicted scaled closing values. To interpret these predictions on the original price scale, we applied the inverse of the Min-Max transformation using the same scaler fitted during preprocessing to both the predicted values and the actual test targets. We

then plotted the reconstructed forecasted and true closing prices over the test period. Finally, we overlaid the training and validation loss curves to provide a clear picture of the model's learning trajectory and to support our discussion of forecasting accuracy in the subsequent results section.

```{python}
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(time_steps, num_features)))
model.add(Dropout(0.2))
model.add(LSTM(50))
model.add(Dropout(0.2))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights
=True)
history = model.fit(
    X_train, y_train,
    epochs=200,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=0
)
y_pred_actual = inverse_scale(scaler, X_test, model.predict(X_test))
y_test_actual = inverse_scale(scaler, X_test, y_test.reshape(-1, 1))
```

Visual Analysis of Actual vs Predicted (Forecasted) Closing Prices

The forecasting performance of the LSTM model was visually assessed across varying look-back windows (5, 10, 20, and 40 days) by plotting actual and predicted closing prices over the test period.

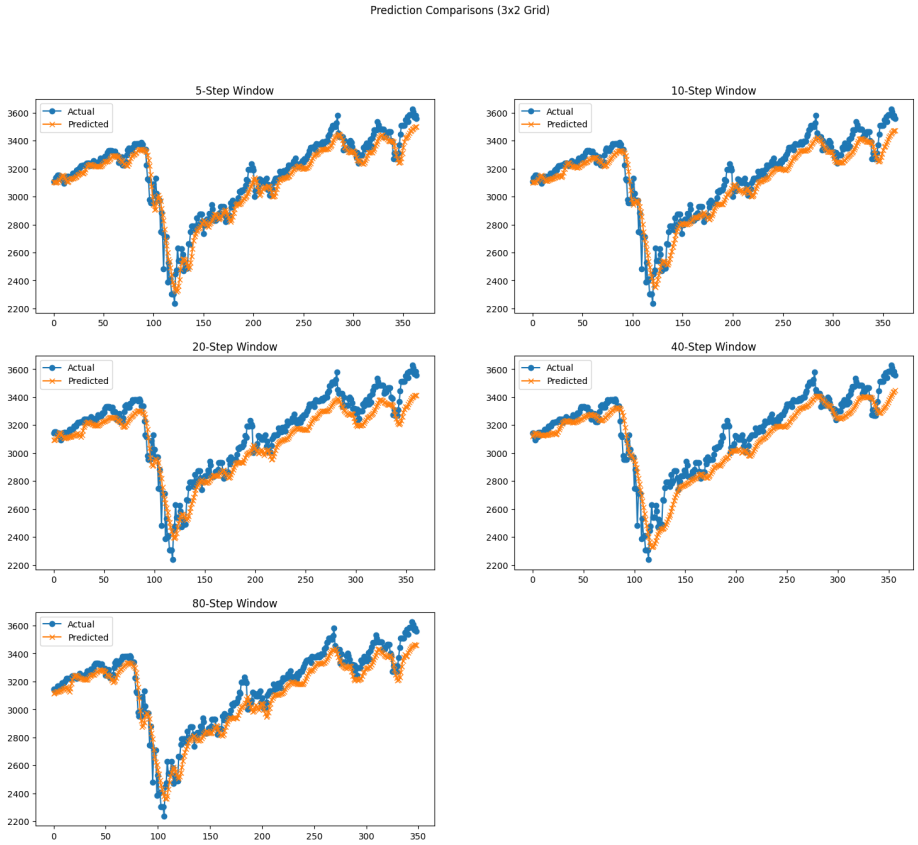


Figure 3: Plot: Actual Close vs Predicted Close by LSTM for different time step window

We can see if we translate the curve of predicted closing prices, it could actually approximate the curve of actual closing price. It means there is a lag, but when we zoom-in these plot, we can see that our model is not really predicting nicely for immediate gains / loss:

Figure 4: Plot: Actual Close vs Predicted Close by LSTM for different time step window (zoomed in)

Evaluation Metrics:

Time-step Window	RMSE	MAE
5	82.07	64.22
10	95.00	75.95
20	107.57	91.17
40	116.92	93.59
80	90.82	75.09

However, our model is performing well if we just look at the performance on the basis of training and validation loss:

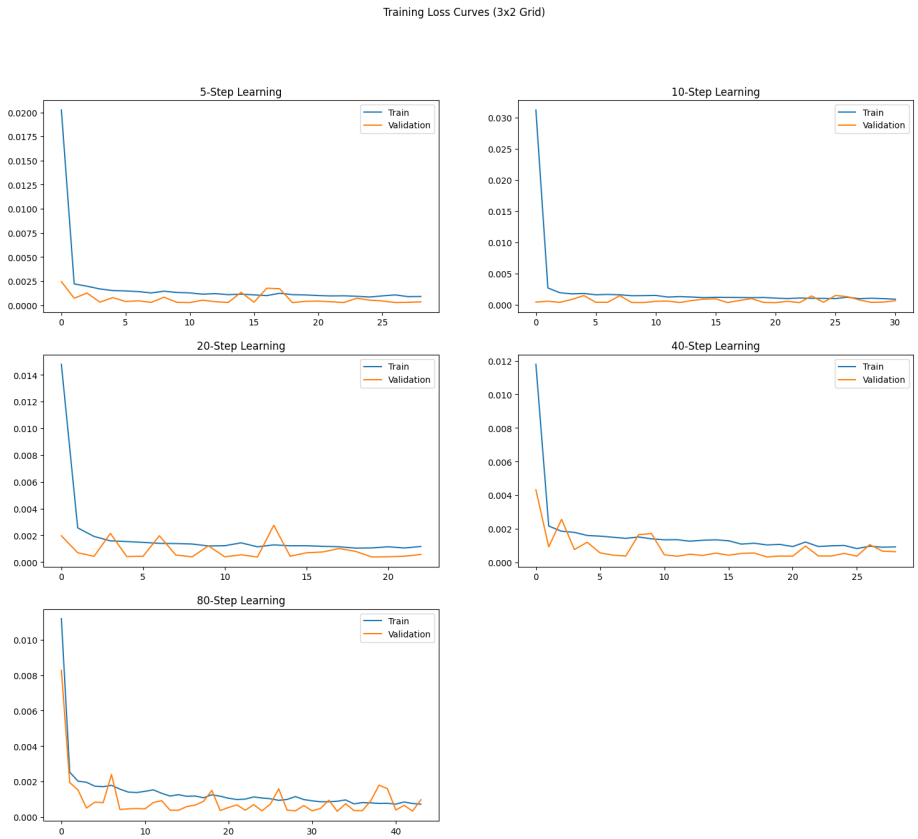


Figure 5: Plot: Training Loss and Validation Loss while Training for different time step window

While the training and validation loss curves (e.g., for 5-, 10-, 20-, and 40-step windows) suggest successful convergence, the visual misalignment between predicted and actual closing prices reveals a critical limitation. The loss functions (MSE, MAE) measure average error magnitudes but are insensitive to systematic timing errors, such as lag. For instance, the LSTM may learn to forecast smoothed approximations of the true series, prioritizing overall trend adherence over precise temporal alignment. This is particularly evident in volatile regimes, where the model's predictions trail abrupt market reversals or spikes, as seen in the attached plots.

The lag arises because financial time series often exhibit stochastic trends and noise that are challenging to disentangle. While the model minimizes average error effectively, it inherently averages out high-frequency fluctuations, resulting in delayed responses to turning points. Thus, the loss curves, while indicative of technical training success, mask the model's struggle to resolve the precise

timing of market movements, a limitation inherent to both the data's complexity and the loss metrics' design.

We will plot the similar results for different number of features we can take into account in our model:

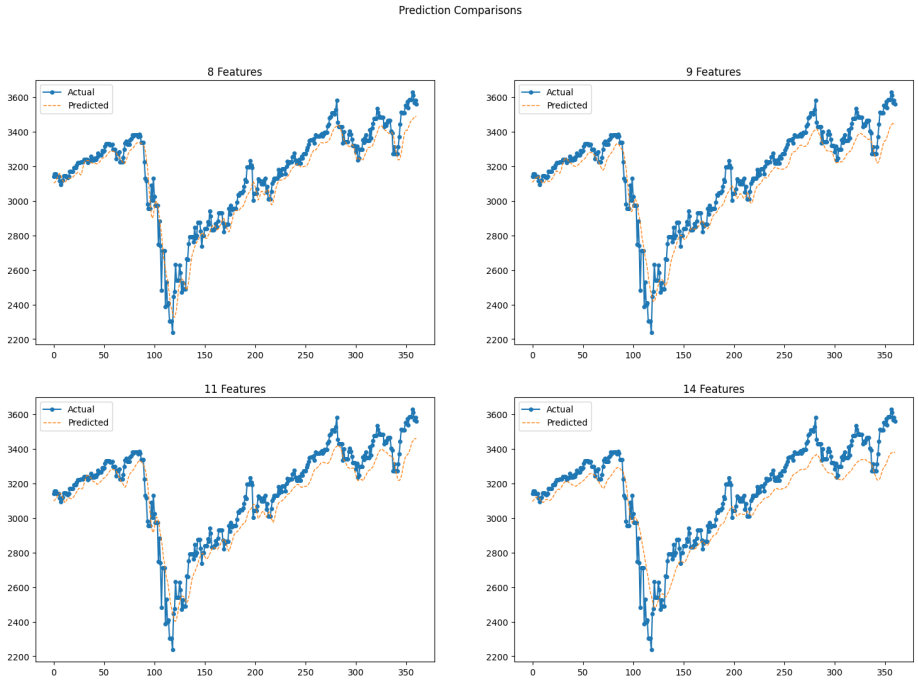


Figure 6: Plot: Actual Close vs Predicted Close by LSTM on different set of features

On zooming it, we have:

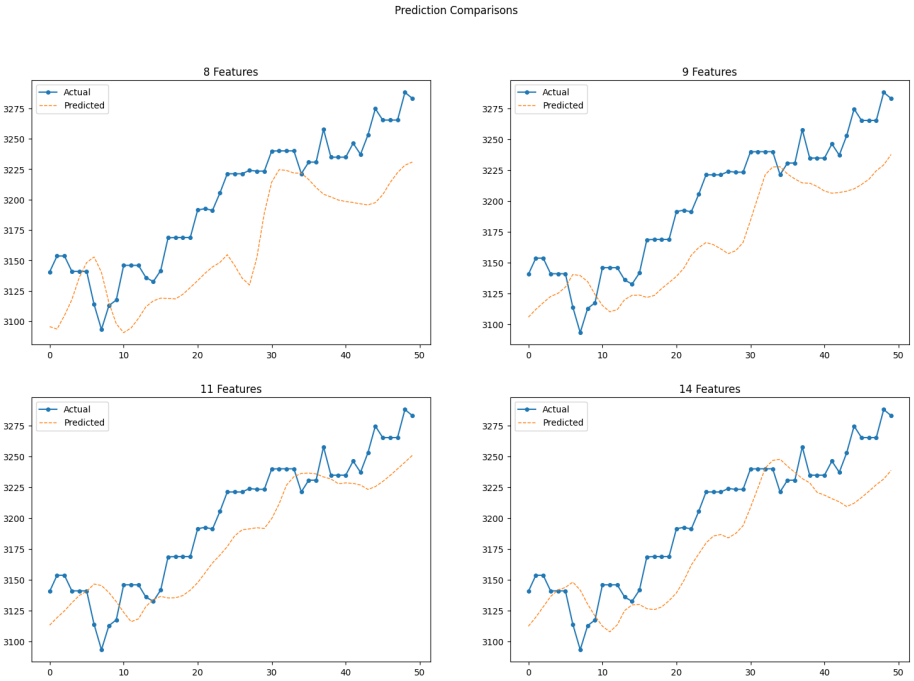


Figure 7: Plot: Actual Close vs Predicted Close by LSTM on different set of features (zoomed-in)

And the model performance plot we got:

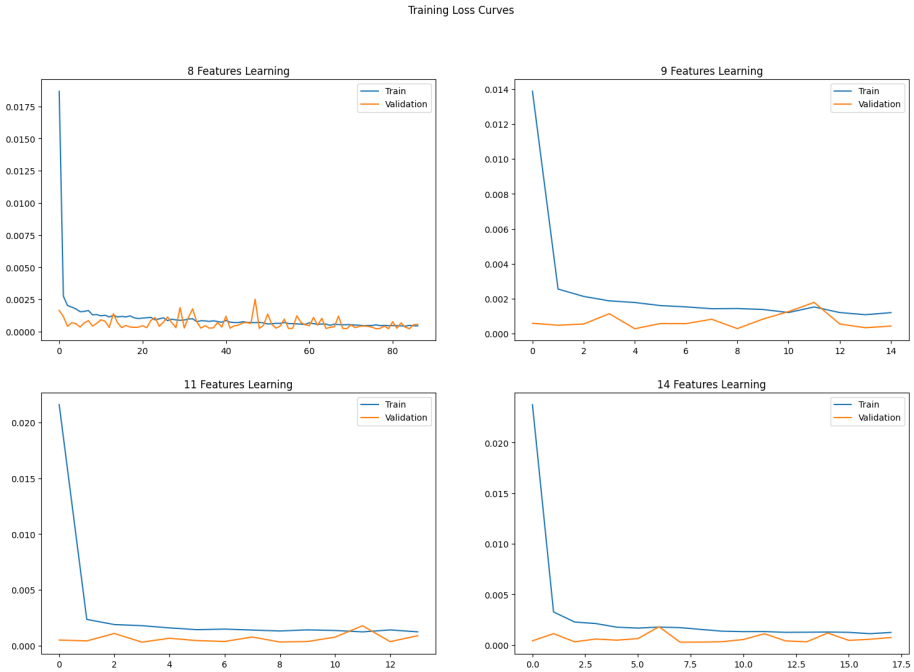
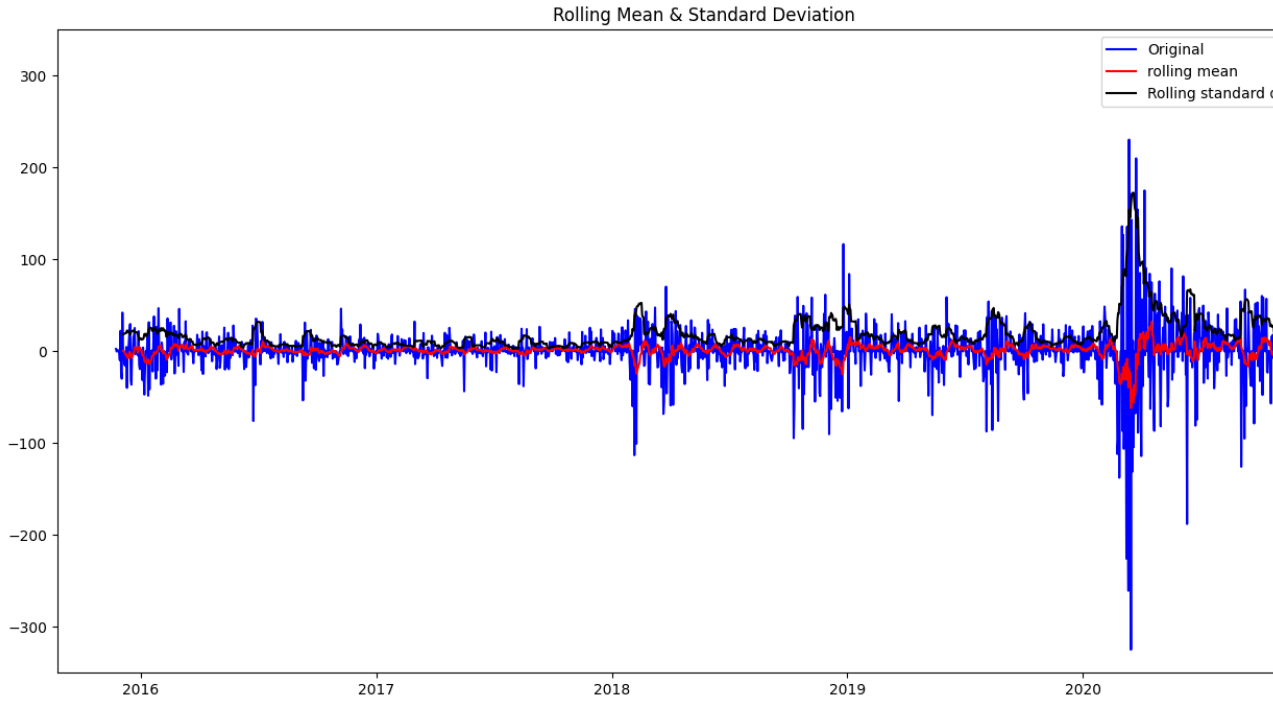


Figure 8: Plot: Training Loss and Validation Loss on different set of features

ARMA Method

To forecast Yahoo stock prices using the ARMA approach, we first ensured stationarity by differencing the log-transformed closing prices. The Dickey-Fuller test (test statistic: -8.52, p-value: 1.09×10^{-13}) and stable rolling statistics confirm stationarity in the differenced series, a prerequisite for ARMA modeling. The absence of trends in the rolling mean and bounded volatility (rolling std) validate consistent statistical properties.

```python
{python}
def test_stationarity(timeseries):
    rolmean = pd.Series.rolling(timeseries,window=12).mean()
    rolstd = pd.Series.rolling(timeseries, window=12).std()
    fig = plt.figure(figsize=(16,8))
    fig.add_subplot()
    orig = plt.plot(timeseries, color = 'blue',label='Original')
    mean = plt.plot(rolmean , color = 'red',label = 'rolling mean')
    std = plt.plot(rolstd, color = 'black', label= 'Rolling standard deviation')
    plt.ylim([-350,350])
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    print( 'Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries,autolag = 'AIC')
    dfoutput = pd.Series(dftest[0:4],index = ['Test Statistic','p-value','#Lags
Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical value (%s)' %key] = value
    print(dfoutput)
ts_log = df['Close']
ts_log_diff = ts_log - ts_log.shift()
ts_log_diff.dropna(inplace=True)
test_stationarity(ts_log_diff)
```



Results of Dickey-Fuller Test: Test Statistic -8.522188e+00 p-value 1.093086e-13 #Lags Used 2.200000e+01 Number of Observations Used 1.801000e+03 Critical value (1%) -3.433986e+00 Critical value (5%) -2.863146e+00 Critical value (10%) -2.567625e+00 dtype: float64

```python
{python}
lag_acf = acf(ts_log_diff, nlags=20)
lag_pacf = pacf(ts_log_diff, nlags=20, method='ols')
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Autocorrelation Function')
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle='--',color='gray')
plt.axhline(y=-1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.axhline(y=1.96/np.sqrt(len(ts_log_diff)),linestyle='--',color='gray')
plt.title('Partial Autocorrelation Function')
plt.show()
```

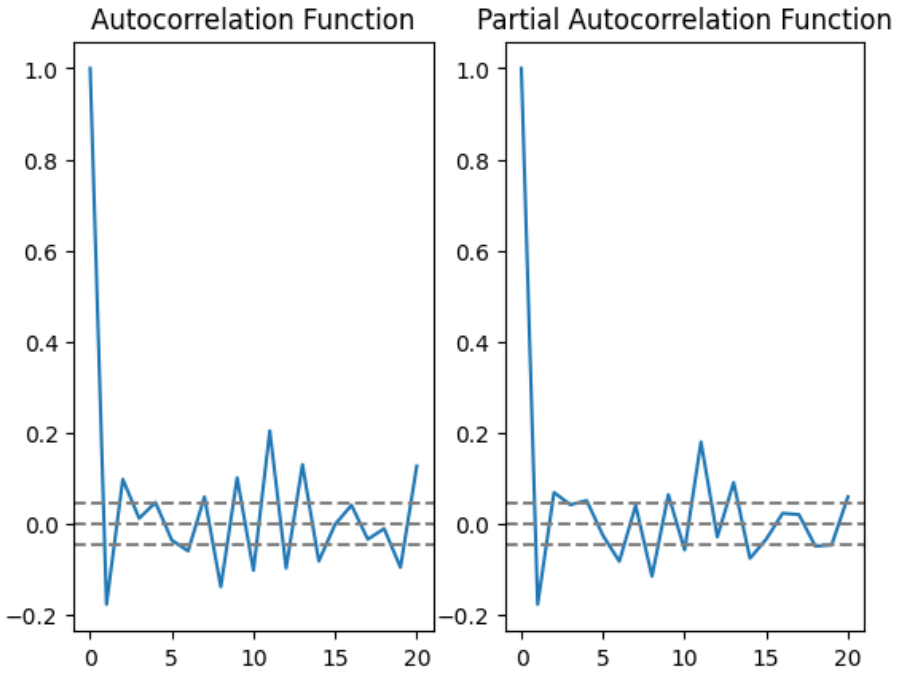


Figure 9: Plot: ACF and PACF

This stationary behavior, coupled with decaying ACF/PACF patterns (not sharp cutoffs), aligns with an ARMA(1,1) structure, which captures short-term autocorrelation via both autoregressive (AR) and moving average (MA) terms. The model's simplicity and alignment with the data's linear dependencies make it a pragmatic choice for forecasting the transformed series. We then implemented an ARIMA(1,0,1) model (equivalent to ARMA(1,1) on the differenced series) and performed walk-forward validation on the test set. At each step, the model was re-fitted to the expanding training window, generating one-step-ahead forecasts.

```python
{python}
train_arma = train_data['Close']
test_arma = test_data['Close']
history = [x for x in train_arma]
y = test_arma
predictions = list()
model_fit = ARIMA(history, order=(1,0,1)).fit()
yhat = model_fit.forecast()[0]
predictions.append(yhat)
history.append(y[0])
for i in range(1, len(y)):
    model = ARIMA(history, order=(1,0,1))
    model_fit = model.fit()
    yhat = model_fit.forecast()[0]
    predictions.append(yhat)
    obs = y[i]
    history.append(obs)
plt.figure(figsize=(14,8))
plt.plot(df.index, df['Open'], color='green', label = 'Train Stock Price')
plt.plot(test_data.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Stock
Price')
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(14,8))
plt.plot(df.index[-100:], df['Open'].tail(100), color='green', label = 'Train Stock
Price')
plt.plot(test_data.index, y, color = 'red', label = 'Real Stock Price')
plt.plot(test_data.index, predictions, color = 'blue', label = 'Predicted Stock
Price')
plt.legend()
plt.grid(True)
plt.show()
print('MSE: '+str(mean_squared_error(y, predictions)))
print('MAE: '+str(mean_absolute_error(y, predictions)))
print('RMSE: '+str(sqrt(mean_squared_error(y, predictions))))
```

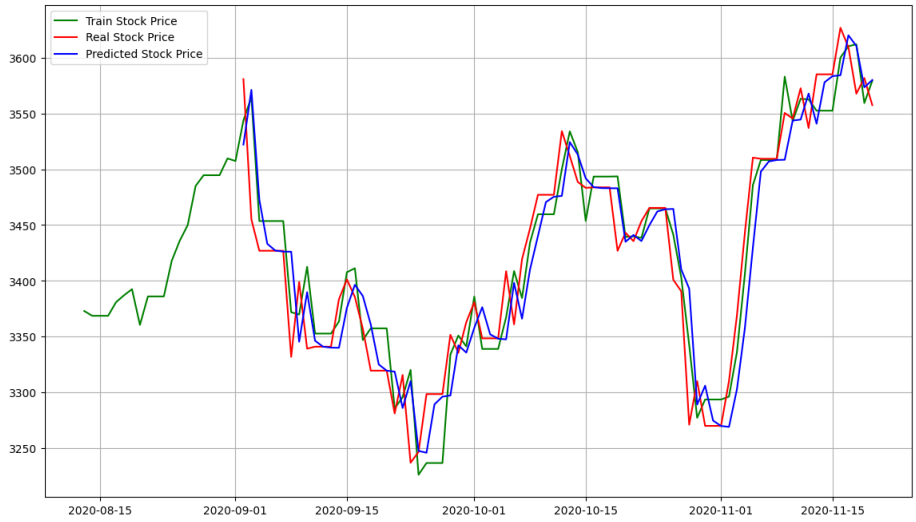


Figure 10: Plot: Actual Close vs Predicted Close vs Trained after Prediction using ARMA (zoomed in)

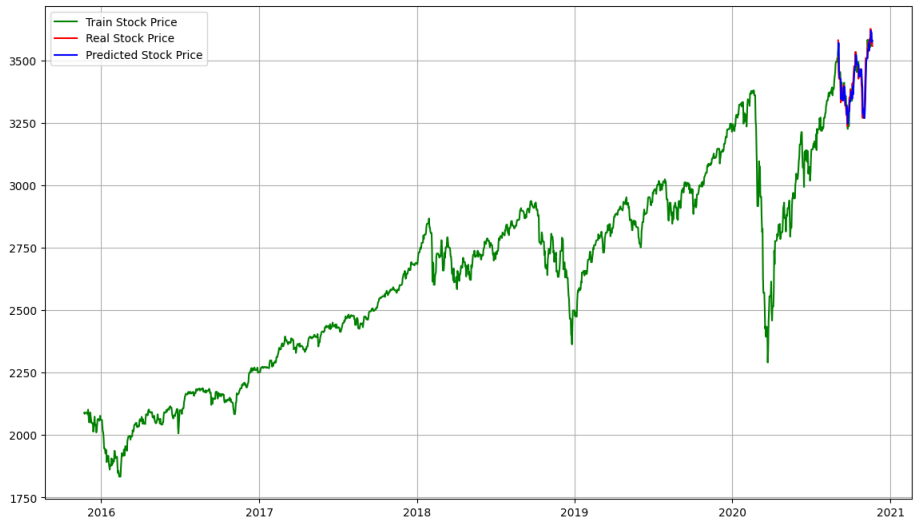


Figure 11: Plot: Actual Close vs Predicted Close vs Trained using ARMA

MSE: 1548.6888350240904 MAE: 27.768974448514733 RMSE: 39.35338403522739

The predicted prices (blue line) in the resulting plots closely followed the actual test series (red line) during stable market periods, demonstrating the model's ability to capture linear dependencies. However, during volatile phases, the ARMA forecasts exhibited noticeable lag, particularly in reacting to sudden price spikes or drops. This aligns with the model's inherent limitation of relying on linear historical relationships, which struggle to adapt to abrupt nonlinear shifts. The iterative retraining approach ensured adaptability to recent trends but introduced computational overhead compared to the LSTM's batch forecasting.

Overall, the ARMA model provided interpretable and stable predictions but lagged in capturing rapid market dynamics, highlighting the trade-off between classical and machine learning methods.

Volatility Prediction

While closing price forecasts provide directional insights, financial markets are inherently noisy and dominated by stochastic shocks, rendering precise point predictions unreliable for risk-sensitive decisions. **Volatility**, the variability of price movements, serves as a critical measure of market uncertainty and risk. Unlike deterministic price trends, volatility exhibits clustering (e.g., periods of high instability followed by calm) and persistence, making it more systematically predictable. By modeling volatility, we shift focus from predicting exact

price levels to quantifying the *magnitude* of expected fluctuations, which is indispensable for portfolio optimization, derivative pricing, and Value-at-Risk (VaR) calculations.

This transition aligns with the limitations observed in our earlier LSTM and ARMA forecasts: both models struggled to anticipate abrupt price swings, as their loss functions prioritized average error reduction over volatility dynamics. Integrating a **GARCH** framework with ARMA explicitly addresses this gap by modeling time-varying variance, where volatility depends on past squared residuals and lagged variances. This approach not only complements price forecasts but also quantifies the confidence intervals around predictions, offering a more holistic view of market behavior. Thus, volatility prediction becomes a pragmatic pivot, bridging the gap between directional accuracy and actionable risk assessment.

Volatility Prediction through ARMA-GARCH:

To model time-varying volatility in Yahoo stock returns, we first computed daily percentage returns from adjusted closing prices, which exhibited characteristic volatility clustering. An ARMA(1,1) model was fitted to the returns to capture linear dependencies in the mean equation, with residuals extracted to isolate the stochastic noise component. The PACF plot of squared residuals confirmed persistent autocorrelation in volatility, justifying the need for a GARCH structure. We then integrated a GARCH(1,2) model to the ARMA residuals, where volatility was modeled as a function of one lagged conditional variance (GARCH term) and two lagged squared residuals (ARCH terms).

```{python}
df = pd.read_csv('yahoo_stock.csv',
                 parse_dates=['Date'],    # parse “Date” into Timestamps
                 index_col='Date')
returns = 100 * df['Close'].pct_change().dropna()
plt.figure(figsize=(10,4))
plt.plot(returns.index, returns)    # x = dates, y = returns
plt.ylabel('Pct Return', fontsize=16)
plt.title('Yahoo Stock Returns', fontsize=20)
plt.tight_layout()
plt.show()
```

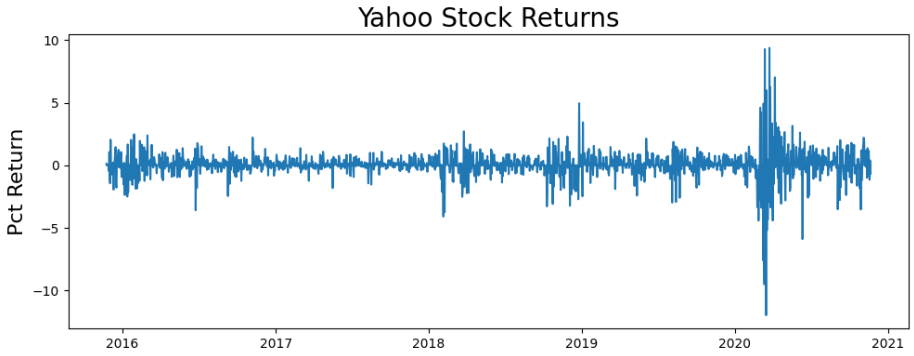


Figure 12: Plot: Percentage Return

```{python}
plot_pacf(returns**2)
plt.show()
```

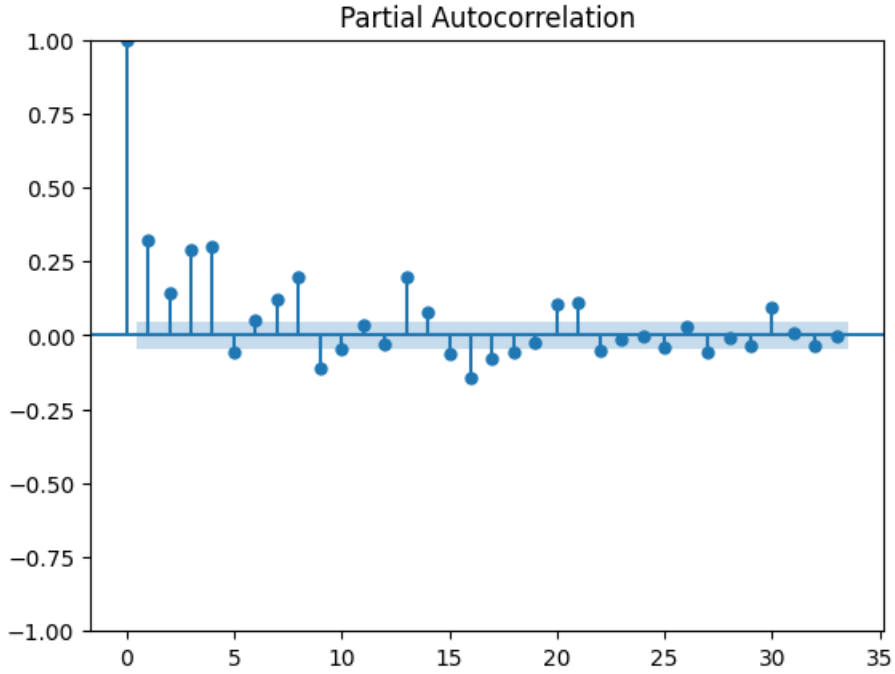


Figure 13: Plot: PACF for Pcnt Change

From the above figure, we can assume that ARMA(4,4) may work and so we compute p-value of coefficients, we realised that choosing p=1 and q=1 will be

sufficient. Similarly, we found using p=1, q=2 for GARCH would have meaning as one coefficient was insignificant when q=3.

```{python}
arma_model = ARIMA(returns, order=(1, 0, 1))
arma_fit = arma_model.fit()
print(arma_fit.summary())
```

self._init_dates(dates, freq) SARIMAX Results
==========
Dep. Variable: Close No. Observations: 1824 Model: ARIMA(1, 0, 1) Log Likelihood -2574.094 Date: Tue, 20 May 2025 AIC 5156.188 Time: 06:09:12 BIC 5178.223 Sample: 11-24-2015 HQIC 5164.316 - 11-20-2020
Covariance Type: opg
==========
coef std err z P>|z| [0.025 0.975] ———— const 0.0344 0.022 1.562 0.118 -0.009 0.078 ar.L1 -0.4325 0.031 -13.857 0.000 -0.494 -0.371 ma.L1 0.2537 0.033 7.594 0.000 0.188 0.319 sigma2 0.9847 0.008 118.005 0.000 0.968 1.001
==========
Ljung-Box (L1) (Q): 0.00 Jarque-Bera (JB): 72120.30 Prob(Q): 0.95 Prob(JB): 0.00 Heteroskedasticity (H): 5.33 Skew: -1.34 Prob(H) (two-sided): 0.00 Kurtosis: 33.69 ==========

```{python}
resid = arma_fit.resid
model = arch_model(resid, p=1, q=2)
model_fit = model.fit()
model_fit.summary()
```

Constant Mean - GARCH Model Results

Dep. Variable:	None	R-squared:	0.000
Mean Model:	Constant Mean	Adj. R-squared:	0.000
Vol Model:	GARCH	Log-Likelihood:	-1881.80
Distribution:	Normal	AIC:	3773.61
Method:	Maximum Likelihood	BIC:	3801.15
		No. Observations:	1824
Date:	Tue, May 20 2025	Df Residuals:	1823
Time:	06:13:40	Df Model:	1

Mean Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
mu	0.0401	1.445e-02	2.775	5.521e-03	[1.177e-02,6.840e-02]

Volatility Model

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	0.0234	7.236e-03	3.228	1.248e-03	[9.174e-03,3.754e-02]
alpha[1]	0.2265	4.628e-02	4.895	9.849e-07	[0.136, 0.317]
beta[1]	0.2025	7.475e-02	2.709	6.755e-03	[5.597e-02, 0.349]
beta[2]	0.5553	6.531e-02	8.503	1.850e-17	[0.427, 0.683]

For robustness, we performed a rolling 1-step volatility forecast over the final 365 days: at each iteration, the ARMA-GARCH pipeline was re-estimated on an expanding window, and the conditional variance was forecasted. This adaptive approach ensured the model dynamically incorporated recent market shocks.

```python
{python}
arma = ARIMA(returns, order=(1,0,1)).fit()
resid = arma.resid  # get the ARMA residuals
# Fit GARCH(1,2) to the residuals
garch = arch_model(resid, vol='GARCH', p=1, q=2, dist='normal')
garch_fit = garch.fit(disp='off')
# -- Rolling 1-step volatility forecast over last 365 days ---
rolling_vol = []
test_size = 365

for i in range(test_size):
    # rolling window of ARMA
    train_ret = returns[:-(test_size - i)]
    arma_i = ARIMA(train_ret, order=(1,0,1)).fit()
    resid_i = arma_i.resid

    # fit GARCH to that window
    garch_i = arch_model(resid_i, vol='GARCH', p=1, q=2)
    garch_fit_i = garch_i.fit(disp='off')

    fcast = garch_fit_i.forecast(horizon=1)
    var1  = fcast.variance.values[-1, 0]
    rolling_vol.append(np.sqrt(var1))

rolling_vol = pd.Series(rolling_vol, index=returns.index[-test_size:])
plt.figure(figsize=(10,4))
plt.plot(returns[-test_size:], label='True Returns')
plt.plot(rolling_vol, label='Predicted Volatility')
plt.title('Rolling Forecast: ARMA(1,1)-GARCH(1,2)')
plt.legend()
plt.tight_layout()
plt.show()
```

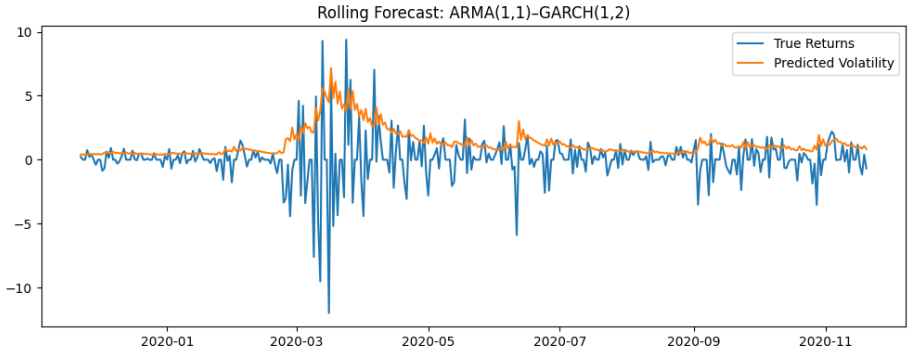
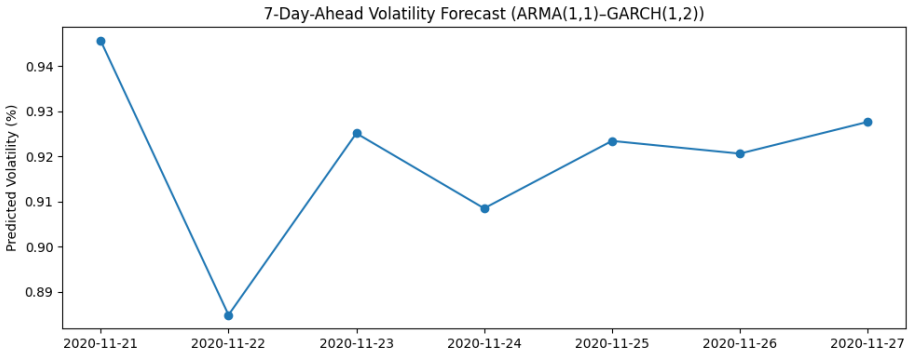


Figure 14: Plot: True Returns vs Volatility using ARMA(1,1)-GARCH(1,2)

Additionally, we generate 7-day-ahead volatility forecast from the full-sample ARMA-GARCH fit, projecting future uncertainty. The results revealed that volatility spikes during market downturns were systematically captured, with the GARCH(1,2) structure effectively reflecting the "memory" of past turbulence.

```python
{python}
arma_full = ARIMA(returns, order=(1,0,1)).fit()
resid_full = arma_full.resid

garch_full = arch_model(resid_full, vol='GARCH', p=1, q=2)
garch_full_fit = garch_full.fit(disp='off')

fcast7 = garch_full_fit.forecast(horizon=7)
var7 = fcast7.variance.values[-1, :]
vol7 = np.sqrt(var7)

future_dates = [returns.index[-1] + timedelta(days=i) for i in range(1,8)]
vol7_series = pd.Series(vol7, index=future_dates)

plt.figure(figsize=(10,4))
plt.plot(vol7_series, marker='o')
plt.title('7-Day-Ahead Volatility Forecast (ARMA(1,1)-GARCH(1,2))')
plt.ylabel('Predicted Volatility (%)')
plt.tight_layout()
plt.show()
```



By decoupling mean (ARMA) and variance (GARCH) dynamics, this hybrid approach quantifies both expected returns and their confidence intervals, addressing a critical limitation of standalone price prediction models.

Volatility Prediction through LSTM-GARCH

To forecast volatility via LSTM, we first derived daily volatility from Yahoo stock returns using a 5-day rolling standard deviation (annualized by scaling with $\sqrt{252}$). The LSTM model incorporated five features: High, Low, Open, Volume, and lagged Volatility—normalized to [0,1] to ensure uniform scaling. Sequences of varying historical windows (5, 10, 20, 40, and 80 days) were constructed to evaluate how temporal context influences predictive accuracy. We need to pay attention that this set of windows (5, 10, 20, 40, 80) is the time step used while training the model. Each sequence paired a window of scaled features with the subsequent volatility value, enabling the LSTM to learn temporal dependencies.

The architecture included two LSTM layers (50 units each) with 20% dropout to mitigate overfitting, followed by a dense output layer. Training employed early

stopping (patience=10) on validation loss, with an 80-20 train-test split. Predictions were inverse-transformed to the original volatility scale by reconstructing the feature matrix, ensuring interpretable outputs.

```
{python}
df['Return'] = df['Close'].pct_change()
window = 5
df['Volatility'] = df['Return'].rolling(window).std() * np.sqrt(252)
df.dropna(inplace=True)
features = df[['High','Low','Open','Volume','Volatility']].values
target = df['Volatility'].values.reshape(-1,1)
scaler = MinMaxScaler()
scaled = scaler.fit_transform(features)

X, y = [], []
for i in range(len(scaled) - time_steps):
  X.append(scaled[i:i+time_steps])
  y.append(scaled[i+time_steps, -1])
X, y = np.array(X), np.array(y)

split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(time_steps, X.shape[2])),
    Dropout(0.2),
    LSTM(50),
    Dropout(0.2),
    Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights
=True)
history = model.fit(X_train, y_train,
                    epochs=200, batch_size=32,
                    validation_split=0.2,
                    callbacks=[early_stop], verbose=1)
y_pred = model.predict(X_test)
y_test_vol = inverse_vol(scaler, X_test, y_test.reshape(-1,1))
y_pred_vol = inverse_vol(scaler, X_test, y_pred)
```

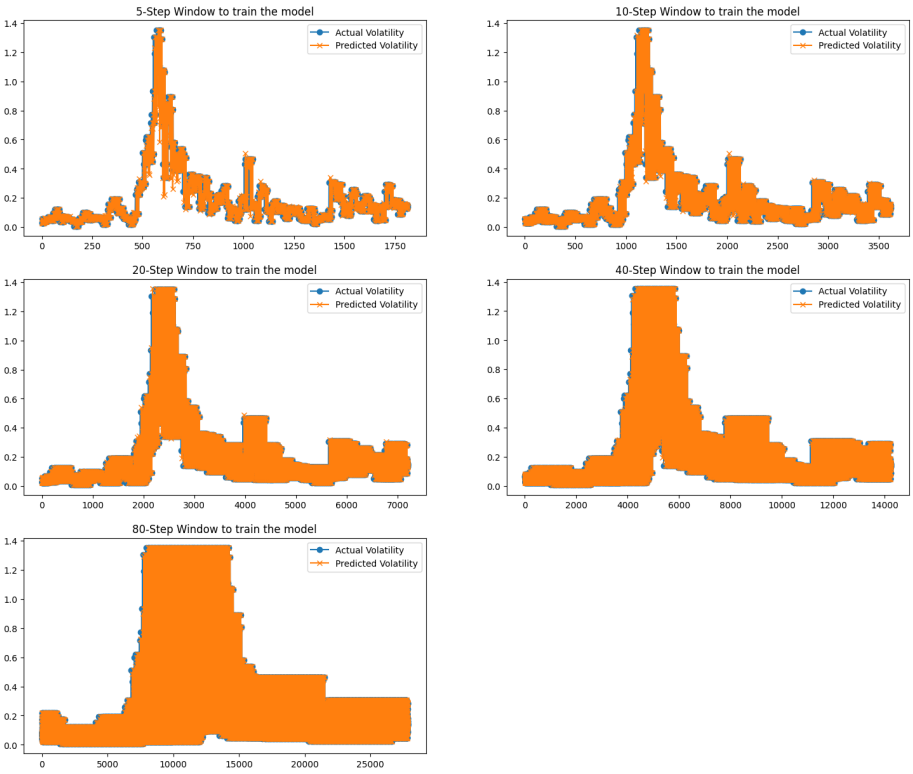


Figure 15: Plot: Rolling forecast using LSTM

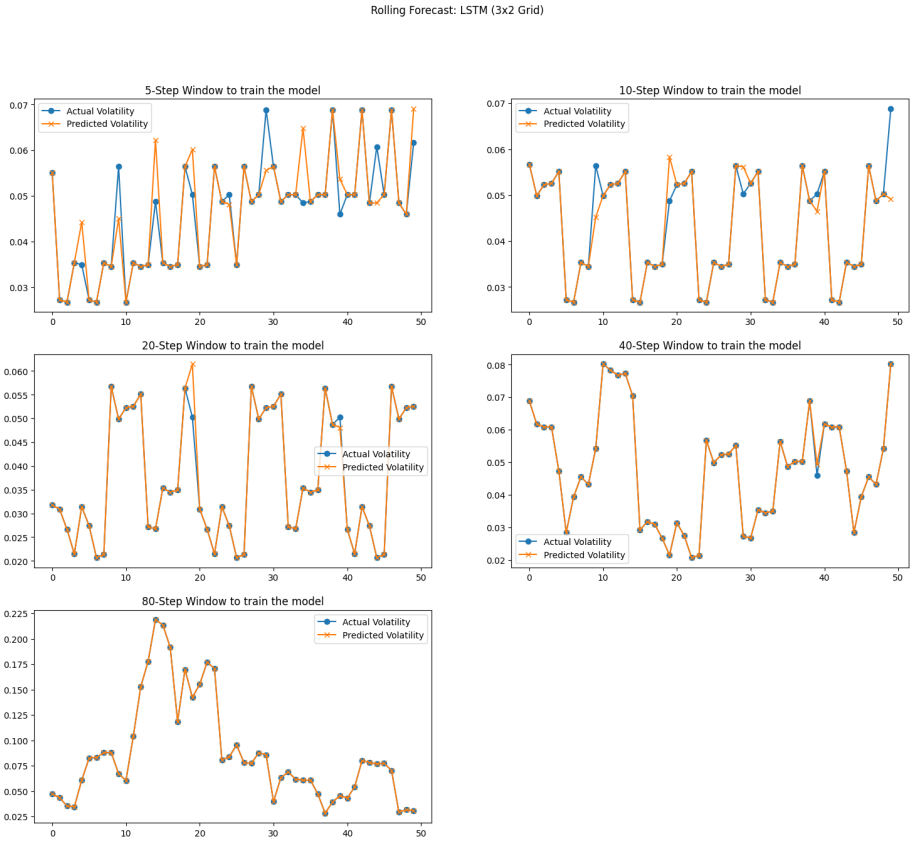


Figure 16: Plot: Rolling forecast using LSTM (zoomed in)

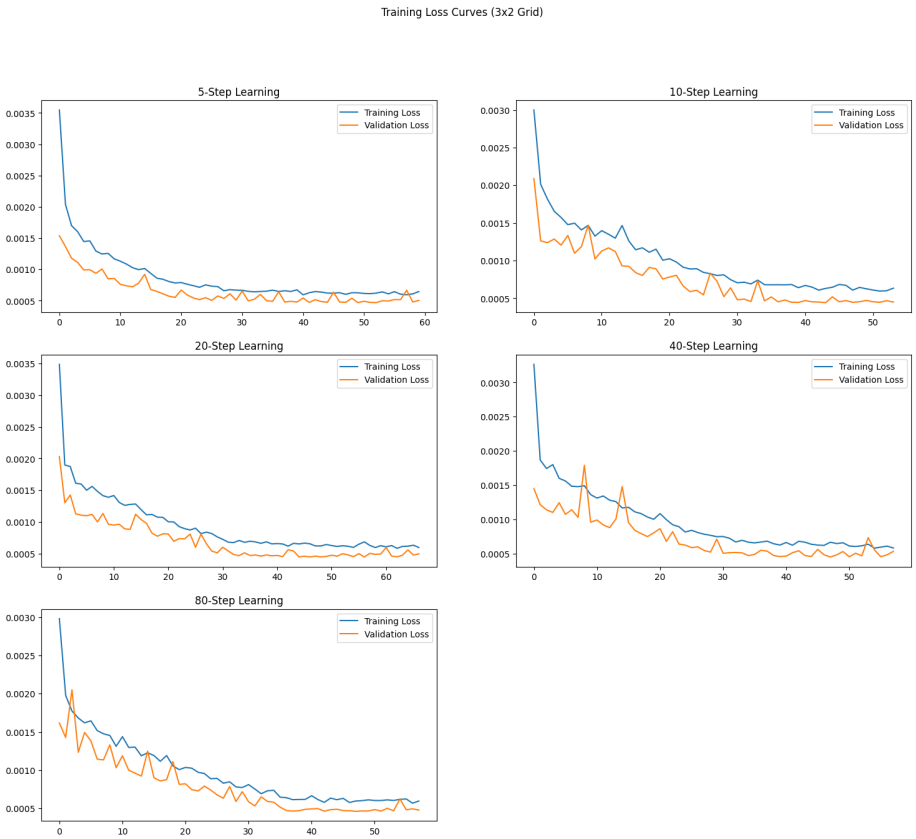


Figure 17: Plot: Training vs Validation Loss in LSTM

Time-Step Window	Volatility RMSE	MAE
5	0.0395	0.0089
10	0.0245	0.0042
20	0.0163	0.0021
40	0.0125	0.0011
80	0.0089	0.0005

Performance was quantified via RMSE and MAE, while prediction plots compared actual vs. forecasted volatility for the test points. We can conclude as we increase the window size while training, the volatility will have spikes at nearly all date. Training and validation loss curves were visualized in a 3×2 grid to assess convergence across window sizes.

Conclusion: Methodological Synergy and Divergence

In this study, we employed two distinct paradigms: **deep learning** and **classical econometrics**, to model and forecast financial time series. The **LSTM network** leveraged its ability to learn nonlinear temporal dependencies directly from raw data, utilizing sequences of historical features (e.g., prices, volume, and volatility) to predict future volatility. Its architecture, designed to retain long-term memory through recurrent layers and dropout regularization, allowed it to adaptively capture complex patterns in market behavior.

Conversely, the **ARMA-GARCH framework** adopted a modular approach: the ARMA component modeled linear dependencies in the mean equation, while the GARCH extension explicitly parameterized time-varying volatility through lagged residuals and conditional variances. This method's strength lies in its interpretability, statistical rigor, and adherence to stylized facts of financial markets (e.g., volatility clustering).

While the LSTM's flexibility accommodates unstructured, high-dimensional inputs, the ARMA-GARCH pipeline operates under well-defined assumptions (e.g., stationarity) and offers transparent parameter estimates. The former excels in learning latent patterns without prior constraints, whereas the latter provides a parsimonious representation of market dynamics grounded in economic theory. Together, these approaches underscore the complementary roles of data-driven machine learning and theory-guided econometrics in financial forecasting, each addressing the challenges of uncertainty and nonlinearity through its unique lens.