Application of Linear Algebra in Quantitative Finance

Samrat Acharya Moon, 2203320

30.09.2024

Linear algebra plays a crucial role in quantitative finance, providing powerful tools for modeling and solving complex financial problems. In this project, we explore its application in optimizing financial portfolios, analyzing risk, and making data-driven decisions. With the rise of data science and machine learning in finance, linear algebra enables efficient computation of large datasets and matrices, facilitating faster and more accurate financial predictions.

This report will specifically focus on the integration of linear algebra with time series forecasting, primarily using the ARIMA (AutoRegressive Integrated Moving Average) model. ARIMA is widely used in finance for modeling and predicting stock prices, volatility, and other economic indicators. We aim to combine these techniques with linear algebra to enhance prediction accuracy and optimize financial strategies.

The source file can be accessed using the following github link: Source file

1 Introduction

The Auto-Regressive Integrated Moving Average (ARIMA) model is a class of statistical models for analyzing and forecasting time series data. The ARIMA model combines three components:

- **Auto-regressive (AR)**: Refers to the use of past values in the regression equation for the series.
- **Integrated (I)**: Refers to differencing the raw observations to make the time series stationary.
- **Moving Average (MA)**: Refers to the dependency between an observation and a residual error from a moving average model applied to lagged observations.

A nonseasonal ARIMA model is classified as an **ARIMA**(p, d, q) model, where:

- p is the number of autoregressive terms,
- d is the number of nonseasonal differences needed for stationarity, and
- q is the number of lagged forecast errors in the prediction equation.

It is one of the most generalized models in statistics.ARIMA models are widely used in finance, economics, and various other domains for forecasting trends and seasonal patterns in time series data.

2 Assumptions

The Data used for this model should be stationary. Stationary, here, means variance, mean and autocorrelations should be constant w.r.t time.

Various methods are used to Stationarize data. In this model, logarithm is used to make variance constant over time. Whereas, Differencing is a part of the model.

3 Components of Model

3.1 Differencing in Time Series

Differencing is a technique used in time series analysis to transform a non-stationary series into a stationary one by subtracting the current observation from a previous observation. A stationary series has constant mean, variance, and autocorrelation over time, which is often required for many time series models, such as ARIMA.

The first difference of a time series $\{y_t\}$ is defined as:

$$\Delta y_t = y_t - y_{t-1}$$

where:

- y_t is the value of the series at time t,
- y_{t-1} is the value of the series at time $t-1$,
- Δy_t is the differenced series (the change between consecutive observations).

The first differencing can remove linear trends from the series, making it stationary. If the series is still non-stationary after first differencing, higher-order differences can be applied. The second difference is defined as:

$$\Delta^2 y_t = \Delta y_t - \Delta y_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2})$$

In general, the d-th difference is given by:

$$\Delta^d y_t = \Delta^{d-1} y_t - \Delta^{d-1} y_{t-1}$$

3.2 Solving the Linear Regression Model using Ordinary Least Squares (OLS)

The linear regression model can be written in matrix form as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where:

- $\mathbf{y}$ is the $n \times 1$ vector of observed values (dependent variable),
- $\mathbf{X}$ is the $n \times p$ design matrix of independent variables (with a column of ones for the intercept),
- $\boldsymbol{\beta}$ is the $p \times 1$ vector of unknown coefficients (parameters),
- $\boldsymbol{\epsilon}$ is the $n \times 1$ vector of errors (residuals).

Step 1: Residual Sum of Squares (RSS)

To estimate the coefficients $\hat{\boldsymbol{\beta}}$, we minimize the Residual Sum of Squares (RSS), which is given by:

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^T (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})$$

Step 2: Deriving the Normal Equations

To find the minimum RSS, we take the derivative of the RSS with respect to the coefficients $\boldsymbol{\beta}$ and set it equal to zero:

$$\frac{\partial}{\partial \boldsymbol{\beta}} \left[(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right] = 0$$

Expanding and simplifying the derivative gives the **normal equations**:

$$\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y}$$

Step 3: Solving for the Coefficients

The normal equations can be solved by multiplying both sides by the inverse of $\mathbf{X}^T\mathbf{X}$ (assuming it is invertible), resulting in the solution for the estimated coefficients $\hat{\boldsymbol{\beta}}$:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Step 4: Computing Predicted Values

Once the coefficients $\hat{\boldsymbol{\beta}}$ are estimated, the predicted values $\hat{\mathbf{y}}$ can be computed as:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$$

Step 5: Calculating Residuals

The residuals, which are the differences between the observed values $\mathbf{y}$ and the predicted values $\hat{\mathbf{y}}$, can be calculated as:

$$\boldsymbol{\epsilon} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$$

Step 6: Assumptions for Solution

For the solution to exist and be unique, the matrix $\mathbf{X}^T\mathbf{X}$ must be invertible. This implies that:

- The independent variables in $\mathbf{X}$ must be linearly independent.
- The number of observations n must be greater than the number of predictors p.

3.3 Autoregressive (AR) Model

The Autoregressive (AR) model is a time series model where the current value of the series is based on its own previous values. In an AR model of order p, denoted as AR(p), the value at time t, y_t, is expressed as a linear combination of the previous p values of the time series, along with a random error term ϵ_t (white noise). The general form of an AR(p) model is:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t$$

where:

- y_t is the value of the time series at time t,
- c is a constant term (or intercept),
- $\phi_1, \phi_2, \ldots, \phi_p$ are the autoregressive coefficients,
- $y_{t-1}, y_{t-2}, \ldots, y_{t-p}$ are the previous values of the time series (lags),
- ϵ_t is the error term (white noise), assumed to be uncorrelated with a mean of zero and constant variance.

The AR model relies on the assumption that the past values of the series have a linear influence on the current value. It is commonly used for forecasting when there is autocorrelation in the time series, meaning the current value is dependent on previous observations.

An AR(1) model, for example, looks like this:

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t$$

This represents a first-order autoregressive process, where the current value y_t depends only on the immediately preceding value y_{t-1} and a random error term ϵ_t.

Below is an example of Python code to implement the AR model using the `statsmodels` library.

Listing 1: Python code for AR model

```
def AR(p,df):
  df_temp = df

  #Generating the lagged p terms
  for i in range(1,p+1):
    df_temp['Shifted_values_%d' % i ] = df_temp['Value'].shift(i)

  train_size = (int)(0.8 * df_temp.shape[0])

  #Breaking data set into test and training
  df_train = pd.DataFrame(df_temp[0:train_size])
  df_test = pd.DataFrame(df_temp[train_size:df.shape[0]])

  df_train_2 = df_train.dropna()
  #X contains the lagged values ,hence we skip the first column
  X_train = df_train_2.iloc[:,1:].values.reshape(-1,p)
  #Y contains the value,it is the first column
  y_train = df_train_2.iloc[:,0].values.reshape(-1,1)

  #Running linear regression to generate the coefficents of lagged terms
  from sklearn.linear_model import LinearRegression
  lr = LinearRegression()
  lr.fit(X_train,y_train)

  theta  = lr.coef_.T
  intercept = lr.intercept_
  df_train_2['Predicted_Values'] = X_train.dot(lr.coef_.T) + lr.intercept_
  # df_train_2[['Value','Predicted_Values']].plot()

  X_test = df_test.iloc[:,1:].values.reshape(-1,p)
  df_test['Predicted_Values'] = X_test.dot(lr.coef_.T) + lr.intercept_
  # df_test[['Value','Predicted_Values']].plot()

  RMSE = np.sqrt(mean_squared_error(df_test['Value'], df_test['
      Predicted_Values']))

  print("The RMSE is :", RMSE,", Value of p :",p)
  return [df_train_2,df_test,theta,intercept,RMSE]
```

In the function defined above, We will later concatenate the predicted values and original Data and later plot it to show forcasting.

3.4 Moving Average (MA) Model

The Moving Average (MA) model is a time series model where the current value of the series depends on past forecast errors (residuals). In an MA model of order q, denoted as MA(q), the value at time t, y_t, is expressed as a linear combination of the past q forecast errors and a mean term. The general form of an MA(q) model is:

$$y_t = \mu + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}$$

where:

- y_t is the value of the time series at time t,

- μ is the mean of the series,
- ϵ_t is the error term (white noise) at time t, assumed to have zero mean and constant variance,
- $\theta_1, \theta_2, \ldots, \theta_q$ are the moving average coefficients,
- $\epsilon_{t-1}, \epsilon_{t-2}, \ldots, \epsilon_{t-q}$ are the past forecast errors (lags of the residuals).

Unlike the Autoregressive (AR) model, which uses past values of the series, the MA model uses past errors or shocks to explain the current value. The error terms ϵ_t represent random noise, and they are crucial in capturing short-term dependencies in the data.

An MA(1) model, for example, looks like this:

$$y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1}$$

This represents a first-order moving average process, where the current value y_t depends on the current error term ϵ_t and the immediately preceding error term ϵ_{t-1}.

Below is an example of Python code to implement the MA model using the `statsmodels` library.

Listing 2: Python code for MA model

```
def MA(q,res):

  for i in range(1,q+1):
    res['Shifted_values_%d' % i ] = res['Residuals'].shift(i)

  train_size = (int)(0.8 * res.shape[0])

  res_train = pd.DataFrame(res[0:train_size])
  res_test = pd.DataFrame(res[train_size:res.shape[0]])

  res_train_2 = res_train.dropna()
  X_train = res_train_2.iloc[:,1:].values.reshape(-1,q)
  y_train = res_train_2.iloc[:,0].values.reshape(-1,1)

  from sklearn.linear_model import LinearRegression
  lr = LinearRegression()
  lr.fit(X_train, y_train)

  theta  = lr.coef_.T
  intercept = lr.intercept_
  res_train_2['Predicted_Values'] = X_train.dot(lr.coef_.T) + lr.intercept_
  # res_train_2[['Residuals','Predicted_Values']].plot()

  X_test = res_test.iloc[:,1:].values.reshape(-1,q)
  res_test['Predicted_Values'] = X_test.dot(lr.coef_.T) + lr.intercept_
  res_test[['Residuals','Predicted_Values']].plot()

  from sklearn.metrics import mean_squared_error
  RMSE = np.sqrt(mean_squared_error(res_test['Residuals'], res_test['
      Predicted_Values']))

  print("The RMSE is :", RMSE," Value of q :",q)
  return [res_train_2,res_test,theta,intercept,RMSE]
```

In the function defined above, We will later concatenate the predicted values and original Data and later plot it to show forcasting

4 Working of ARIMA model

The data used in this project is of AAPL(apple stocks). Training and testing portions have been concatenated, considering 80% of data is for training and remaining is for testing.

Figure 1, shows the format of Data we are using. The code snippet below is used to convert the data of each day to data of each month for the years(2000-2024). And naming it Values as the column is unnamed.

```
Date
2010-01-04    7.643214
2010-01-05    7.656429
2010-01-06    7.534643
2010-01-07    7.520714
2010-01-08    7.570714
Name: Close, dtype: float64
```

Figure 1: Stocks data

Listing 3: Grouping the data by mean values

```
df.index = pd.to_datetime(df.index)
df = pd.DataFrame(df.groupby(df.index.strftime('%Y-%m')).mean())
df.columns=['Value']
```

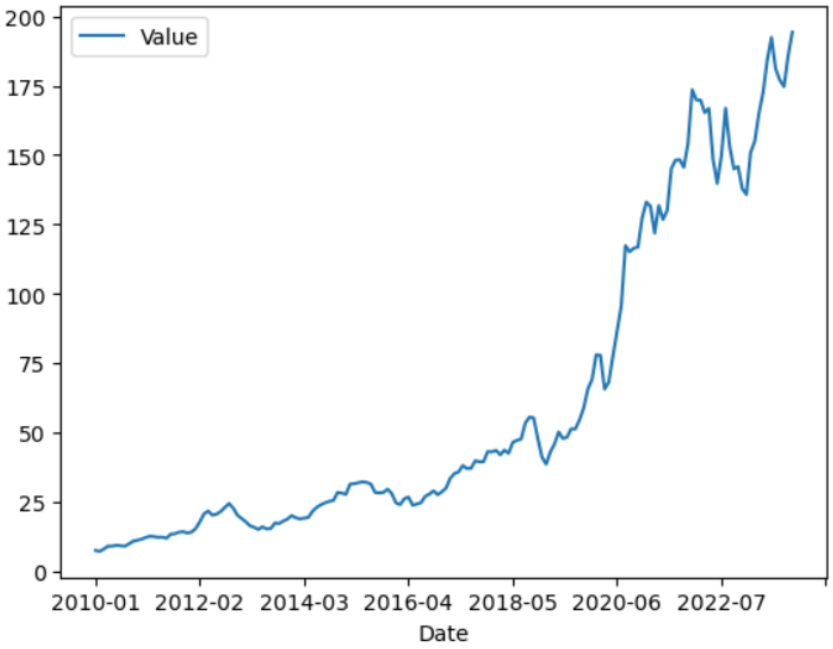

Figure 2: Plotting the data

After getting these values, we shall move the following steps:

4.1 Making the Data Stationary

Using Log and Differencing on each value, the data of Apple stocks is tried to made stationary.

Listing 4: Differencing the data

```
df_testing = pd.DataFrame(np.log(df.Value).diff().diff(12))
adf_check(df_testing.Value.dropna())
```

```
# print(df_testing.to_string())
df_testing.plot()

ACF = plot_acf(df_testing.dropna(),lags=50)
PACF = plot_pacf(df_testing.dropna(),lags=50)
```

Note : ACF and PACF are used for guessing of P factor for AR model.

4.2 Fitting AR model

Since our data set is small, we will find the value of p factor through brute force.

Listing 5: Finding p factor and getting the AR model for it

```
best_RMSE=100000000000
best_p = -1

for i in range(1,21):
    [df_train,df_test,theta,intercept,RMSE] = AR(i,pd.DataFrame(df_testing.iloc
        [:, 0]))
    if(RMSE<best_RMSE):
        best_RMSE = RMSE
        best_p = i

print(best_p)

[df_train,df_test,theta,intercept,RMSE] = AR(best_p,pd.DataFrame(df_testing.
    Value))
```

Then get the AR model with optimum factor p.

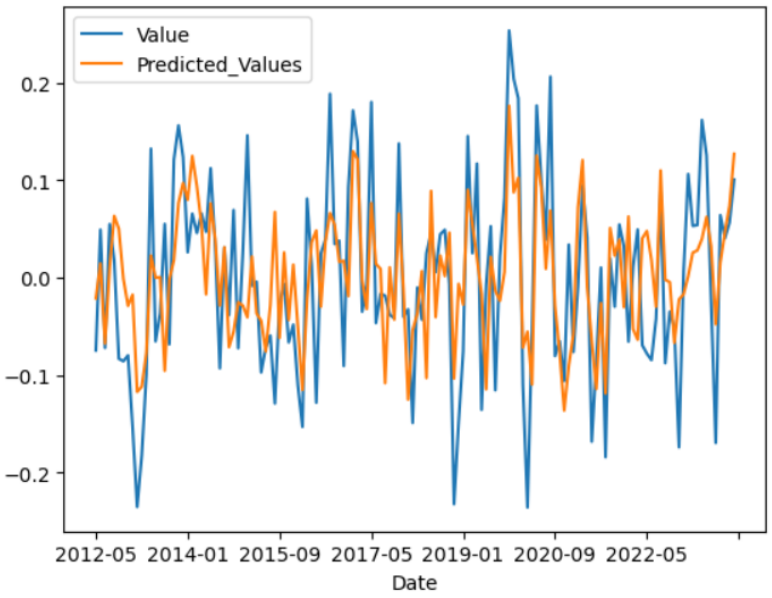

Figure 3: Forecasted Values using AR Model

For the MA model we need to get the error/residual values for Fitting in the model. the following snippet is used to achieve that. We have already considered this while making our AR model.

Listing 6: Getting residuals

```
res = pd.DataFrame()
res['Residuals'] = df_c.Value - df_c.Predicted_Values

res.plot(kind='kde')
```

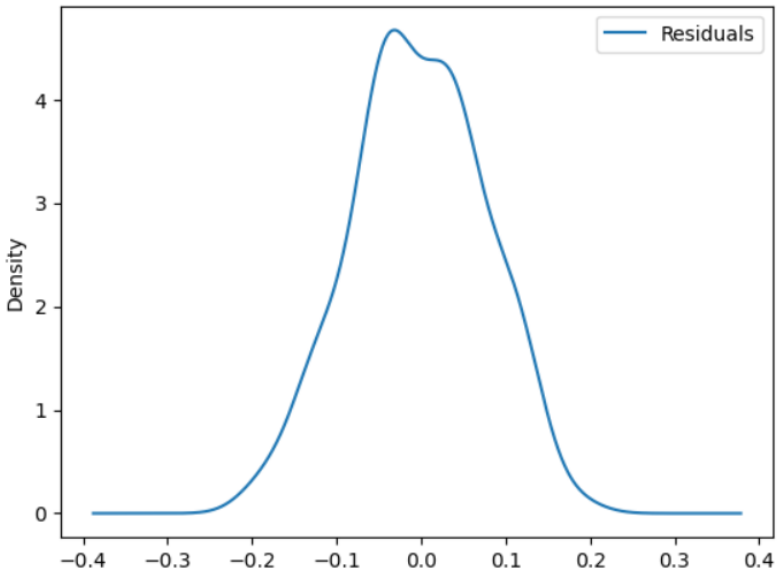

Figure 4: Residual Graph

4.3 Fitting MA model

Now, we shall start with the MA model part. Just enter the residual values in the model. Again for MA model we need an optimum factor q. With such a small dataset, again brute force wont be an issue.

Listing 7: Findind q factor and getting the MA model for it

```
best_RMSE=100000000000
best_q = -1

for i in range(1,20):
    [res_train,res_test,theta,intercept,RMSE] = MA(i,pd.DataFrame(res.Residuals)
        )
    if(RMSE<best_RMSE):
        best_RMSE = RMSE
        best_q = i

print(best_q)

[res_train,res_test,theta,intercept,RMSE] = MA(best_q,pd.DataFrame(res.
    Residuals))
```

Using the best value of q to train MA model, We get the following graph.

Now, we simply concatenate our training and testing set for both residual and Predicted values.

Also adding residuals to the data points for error correction.

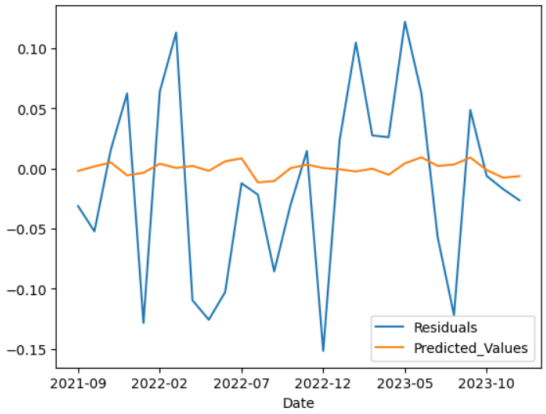

Figure 5: Forecasted Values using MA Model

Listing 8: Concatenation

```
df_c = pd.concat([df_train, df_test])
res_c = pd.concat([res_train, res_test])

df_c.Predicted_Values += res_c.Predicted_Values
```

This Final plot for the Stationary data will be:

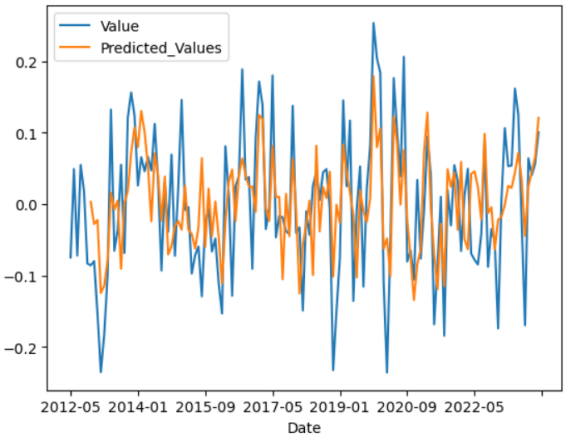

Figure 6: Stationary predicted value plot

4.4 Getting Original Data back

The above plotted graph represents the Stationary data after transformations that we applied in the beginning. To get the actual plot, just reversing the Transformation should work to get the actual values:

The Code snippet below does the same:

Listing 9: Python code for ARIMA model

```
df_c.Value += np.log(df).shift(1).Value
df_c.Value += np.log(df).diff().shift(12).Value
df_c.Predicted_Values += np.log(df).shift(1).Value
df_c.Predicted_Values += np.log(df).diff().shift(12).Value
```

```
df_c.Value = np.exp(df_c.Value)
df_c.Predicted_Values = np.exp(df_c.Predicted_Values)
```

5 Graphs and Outputs

The Final plot with predicted values we achieve after the tranformations is shown in fig 7:

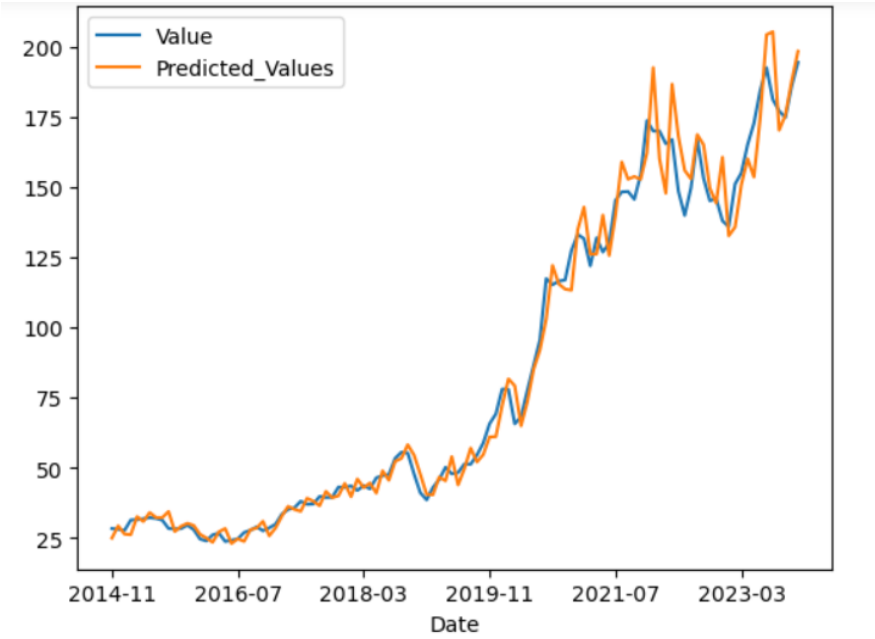

Figure 7: Forecasted Values using ARIMA Model

6 Conclusion:

The ARIMA model can act as a powerful tool for forecasting in any statistical scenario. As the model only depends on previous values for predictions, it may not consider various other factors such as market sentiment.

But ARIMA is the most General Statistical model. With Different values of p,d and q, other interpretations can be made.

Fun Facts

The following is a list of different ARIMA models and their corresponding interpretations:

- **ARIMA(p, d, q)**: General ARIMA forecasting equation
- **ARIMA(1, 0, 0)**: First-order autoregressive model (AR(1))
- **ARIMA(0, 1, 0)**: Random walk
- **ARIMA(1, 1, 0)**: Differenced first-order autoregressive model
- **ARIMA(0, 1, 1) without constant**: Simple exponential smoothing
- **ARIMA(0, 1, 1) with constant**: Simple exponential smoothing with growth
- **ARIMA(0, 2, 1) or ARIMA(0, 2, 2) without constant**: Linear exponential smoothing
- **ARIMA(1, 1, 2) with constant**: Damped-trend linear exponential smoothing

7 References

References

[1] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1976.

[2] R. J. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018.

[3] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2nd edition, 2002.

[4] `Geeksforgeeks` Documentation, *Least Square Method*, Available at: https://www.geeksforgeeks.org/least-square-method/.

[5] `Geeksforgeeks` Documentation, *Linear Regression in Machine Learning*, Available at: https://www.geeksforgeeks.org/ml-linear-regression/.

[6] `people.duke.edu` Documentation, *Introduction to ARIMA: nonseasonal models*, Available at: https://people.duke.edu/~rnau/411arim.htm.