ARIMA (AutoRegressive Integrated Moving Average) & Box-Jenkins applied to *"Colse"* data

---

1 Model identification:

The first step in building an ARIMA model using the Box-Jenkins method is to identify the appropriate order of differencing, autoregression, and moving average terms. The goal is to transform the non-stationary time series into a stationary one through differencing, and then to fit an ARIMA model to the stationary series.

We started by visually inspecting the time series plot and checking for any trends or seasonal patterns. We also used the Augmented Dickey-Fuller (ADF)test to check for stationarity.

After performing the ADF test on the original time series, we found that the time series was non-stationary. To address this, we applied the logarithm function to the time series, as it is a commonly used technique to make non-stationary time series stationary. However, we found that the log-transformed time series was still non-stationary. Therefore, we applied difference to the time series, and we found that the difference time series was stationary. This is shown in the figures below.

```
% Load data from Shel.csv
data = readtable('Shel.csv', 'VariableNamingRule', 'preserve');

% Extract the dates and Close column
dates = data.Date;
close_values = data.Close;
% Perform augmented Dickey-Fuller (ADF) test
[h, pValue, stat, criticalValues] = adftest(close_values);
disp(['ADF test statistic: ',num2str(stat)])
disp(['p-value: ',num2str(pValue)])
disp(['Critical values: ',num2str(criticalValues')])

% Apply natural logarithm to Close values
log_close_values = log(close_values);
% Perform augmented Dickey-Fuller (ADF) test
[h, pValue, stat, criticalValues] = adftest(log_close_values);
disp(['ADF test statistic: ',num2str(stat)])
disp(['p-value: ',num2str(pValue)])
disp(['Critical values: ',num2str(criticalValues')])

% Apply Differencing  to Close values
Diff_close_values=[close_values(1);diff(close_values)];
% Perform augmented Dickey-Fuller (ADF) test
[h, pValue, stat, criticalValues] = adftest(Diff_close_values);
disp(['ADF test statistic: ',num2str(stat)])
disp(['p-value: ',num2str(pValue)])
disp(['Critical values: ',num2str(criticalValues')])
```

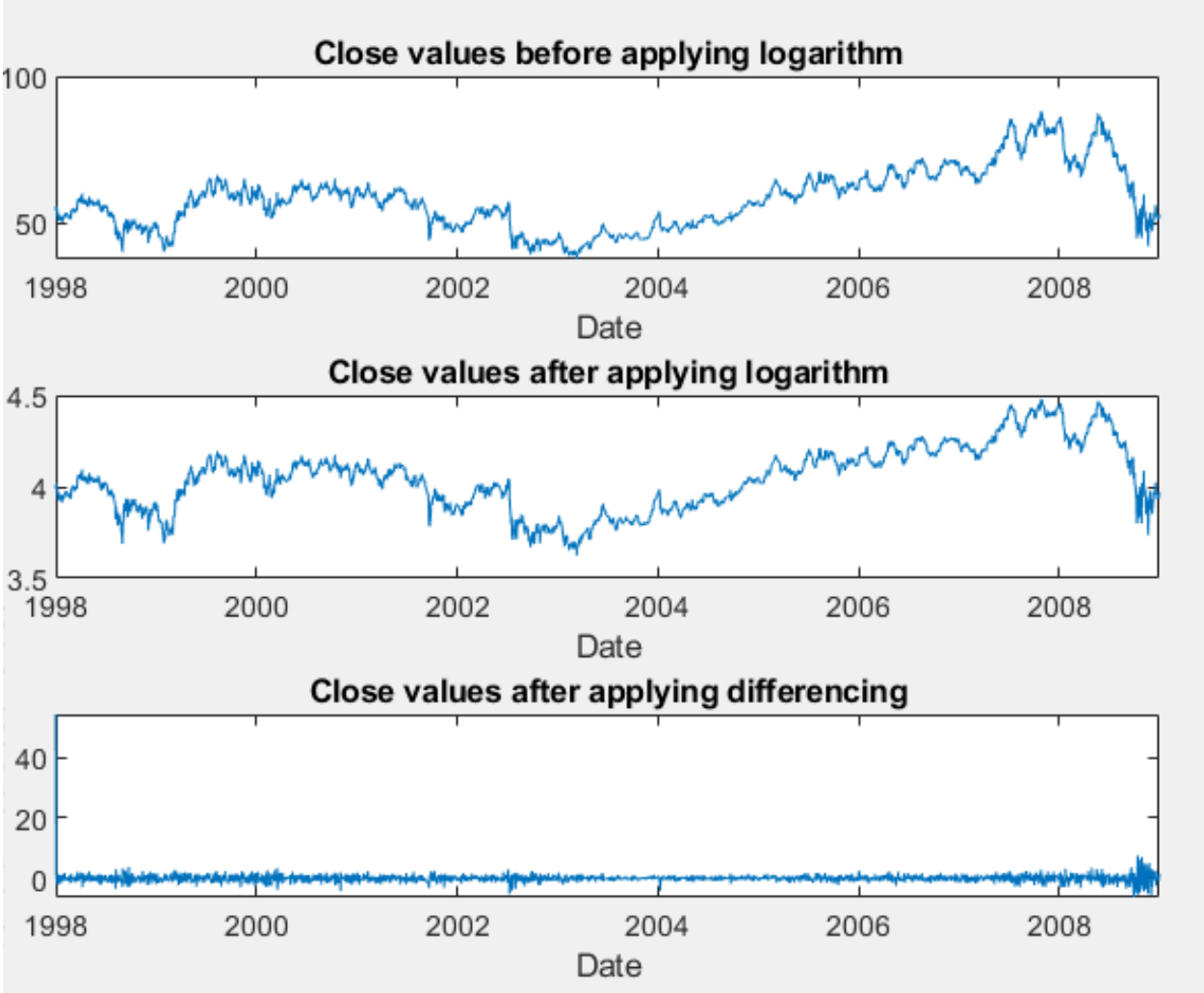


Figure 1:

We chose the value of d as 1 because the differencing parameter d in an ARIMA model is significant in determining the number of times that the time series needs to be differenced to achieve stationarity. Stationarity is a fundamental assumption of many time series analysis techniques, including ARIMA models. To determine the values of p and q for the ARIMA model, we used the Bayesian Information Criterion (BIC) and Akaike Information Criterion (AIC) to measure the goodness of fit of the model while taking into account the number of parameters used to estimate the model.

The BIC and AIC are information criteria that balance the fit of the model with the complexity of the model. The BIC penalizes models with more parameters more heavily than the AIC. The lower the BIC or AIC value, the better the model fit.

We tried different combinations of p and q values and calculated the corresponding BIC and AIC values for each model. We chose the values of p and q that resulted in the lowest BIC and AIC values, as these values indicate the best balance between model fit and complexity.

After applying this algorithm, we found that the optimal values for p and q were 17 and 116, respectively. These values were chosen based on the lowest BIC and AIC values obtained from fitting ARIMA models with different combinations of p and q values.

```
% Define range of possible values for p and q
p = 0:20;
q = 0:200;

% Initialize AIC and BIC matrices
AIC = zeros(length(p), length(q));
BIC = zeros(length(p), length(q));

% Estimate ARMA models and compute AIC and BIC values
for i = 1:length(p)
    for j = 1:length(q)
        try
            model = arima(p(i), 1, q(j));
            fit = estimate(model, close_values);

            % Compute log-likelihood
            resid = infer(fit, close_values);
            sigma2 = fit.Variance;
            loglik = -0.5 * (nobs * log(2*pi) + nobs * log(sigma2) + sum(resid.^2) /
                sigma2);
            % Compute the number of parametres
            numparams = size(fit.AR, 2) + size(fit.MA, 2) + 1;

            [AIC(i,j), BIC(i,j)] = aicbic(loglik, numparams, nobs);
        catch ME
            disp(['Error for p = ', num2str(p(i)), ', q = ', num2str(q(j)), ': ', ME.
                message]);
            AIC(i,j) = NaN;
            BIC(i,j) = NaN;
        end
    end
end

% Find p and q values with lowest AIC and BIC values
[minAIC, idxAIC] = min(AIC(:));
[minBIC, idxBIC] = min(BIC(:));
[bestpAIC, bestqAIC] = ind2sub(size(AIC), idxAIC);
[bestpBIC, bestqBIC] = ind2sub(size(BIC), idxBIC);

% Display results
disp(['Best p and q values based on AIC: ', num2str(bestpAIC-1), ', ', num2str(bestqAIC
    -1)]);



disp(['Best p and q values based on BIC: ', num2str(bestpBIC-1), ', ', num2str(bestqBIC
    -1)]);
```

Finally we have : $P = 17$; $D = 1$; $Q = 116$

1.1 Parameters estimation

To estimate the parameters of an ARIMA model in Matlab, we can use the built-in function estimate. This function takes the time series data and the values of p, d, and q as input and returns an object that represents the estimated ARIMA model.

```
% Define the ARIMA model parameters
p =17;      % AR order

d = 1;     % differencing order

q = 116;    % MA orderV

% Fit the ARIMA model to the training data
model = arima(p, d, q);
fit = estimate(model, train_data);
```

ARIMA(17,1,116) Model (Gaussian Distribution):

	Value	StandardError	TStatistic	PValue
Constant	8.2589e-05	0.00033616	0.24568	0.80593
AR{1}	0.049148	1.7194	0.028584	0.9772
AR{2}	-0.071841	0.73242	-0.098087	0.92186
AR{3}	0.18714	0.75892	0.24659	0.80523
AR{4}	0.1014	0.41058	0.24698	0.80492
AR{5}	-0.13235	0.35441	-0.37344	0.70882
AR{6}	-0.20946	0.35167	-0.59562	0.55143
AR{7}	0.16515	0.41969	0.39349	0.69396
AR{8}	-0.22952	0.41159	-0.55764	0.57709
AR{9}	0.10672	0.54757	0.1949	0.84547
AR{10}	0.13161	0.41233	0.31918	0.74959
AR{11}	0.13528	0.32108	0.42133	0.67351
AR{12}	-0.016882	0.45009	-0.037508	0.97008
AR{13}	0.11427	0.29452	0.38798	0.69803
AR{14}	0.2185	0.40424	0.54052	0.58884
AR{15}	-0.0026859	0.60783	-0.0044189	0.99647
AR{16}	0.13219	0.31714	0.41681	0.67682
AR{17}	-0.089969	0.47738	-0.18846	0.85051
MA{1}	-0.1277	1.7189	-0.07429	0.94078
MA{2}	0.0079505	0.6942	0.011453	0.99086
MA{3}	-0.14821	0.63431	-0.23366	0.81525
MA{4}	-0.1106	0.38801	-0.28506	0.7756
MA{5}	0.13053	0.3505	0.37241	0.70959
MA{6}	0.18068	0.34696	0.52075	0.60254

Figure 2:

2 Diagnostic checking

Once we have estimated the parameters of the ARIMA model, we need to check the goodness-of-fit of the model. One way to do this is to examine the residuals of the model and perform diagnostic checks to ensure that they are white noise.

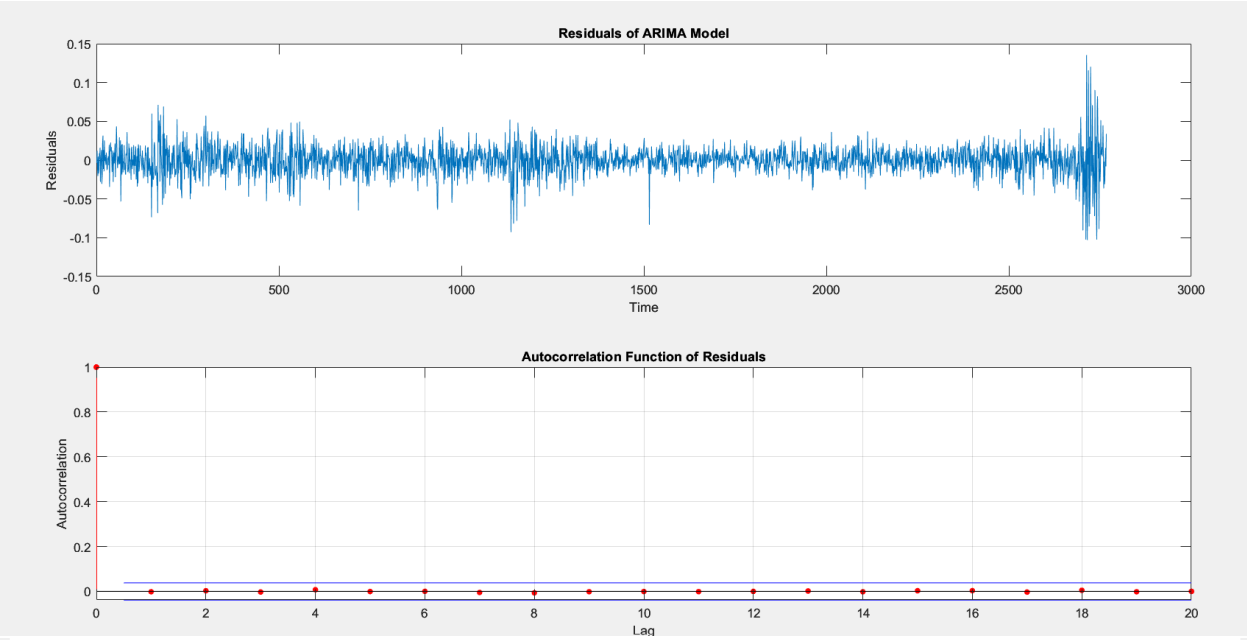
We perform a Ljung-Box test for residual autocorrelation using the lbqtest function. This test checks whether the residuals exhibit significant autocorrelation at different lags. If the p-value of the test is less than the significance level, we reject the null hypothesis of no autocorrelation, and conclude that the residuals exhibit significant autocorrelation.

If the Ljung-Box test result is "The residuals exhibit significant autocorrelation," we can conclude that the ARIMA model does not adequately capture the autocorrelation structure in the time series data. This means that the residuals exhibit some systematic patterns or structures that are not accounted for by the model.

If the Ljung-Box test result is "The residuals do not exhibit significant autocorrelation," we can conclude that the ARIMA model adequately captures the autocorrelation structure in the time series data. This means that the residuals are effectively random and do not exhibit any systematic patterns or structures.

```
% Diagnostic checking
residuals = infer(fit, train_data);
figure;
subplot(2,1,1);
plot(residuals);
title('Residuals of ARIMA Model');
xlabel('Time');
ylabel('Residuals');

subplot(2,1,2);
autocorr(residuals);
title('Autocorrelation Function of Residuals');
xlabel('Lag');
ylabel('Autocorrelation');

% Ljung-Box test for residual autocorrelation
[h, pValue, stat, criticalValues] = lbqtest(residuals, 'Lags', [10, 15, 20]);
disp(['Ljung-Box test statistic: ',num2str(stat)])
disp(['p-value: ',num2str(pValue)])
disp(['Critical values: ',num2str(criticalValues)])

if h==1
    disp("The residuals exhibit significant autocorrelation.");

else
    disp("The residuals do not exhibit significant autocorrelation.");
end
```

Our model produced a non-significant Ljung-Box test result, which suggests that the ARIMA model adequately captures the autocorrelation structure in the time series data. This means that the residuals are effectively random and do not exhibit any systematic patterns or structures. As a result, w***e can be more confident in the model's ability to provide reliable forecasts for future values.***




```
Ljung-Box test statistic: 0.40014     0.44419     0.57086
p-value: 1           1           1
Critical values: 18.307      24.9958      31.4104
The residuals do not exhibit significant autocorrelation.
```

Figure 3:

3 Forecasting

We then perform forecasting using the forecast function and specify the forecast horizon as 500 time units. The forecast function uses the estimated ARIMA model to generate a sequence of forecasted values for the specified forecast horizon.

Finally, we plot the forecasted values on top of the actual data using the plot function and add a legend to distinguish between the actual and forecasted values.

```
% Generate forecasts for the next 500 time units
[forecast_data, YMSE] = forecast(fit, 500, 'Y0', train_data);

% Convert the forecasted values back to the original scale
forecast_values = exp(forecast_data);
YMSE=exp(YMSE);

% Compute lower and upper 95% forecast intervals
lower = forecast_values - 1.96*sqrt(YMSE);
upper = forecast_values + 1.96*sqrt(YMSE);

% Plot the original data and the initial forecasted values
figure;
plot(dates, close_values);
hold on;
forecast_line = plot(dates(end)+1:dates(end)+500, forecast_values);
lower_line = plot(dates(end)+1:dates(end)+500, lower, 'g--');
upper_line = plot(dates(end)+1:dates(end)+500, upper, 'g--');
```

```
legend('Original data', 'Forecasted values', 'Lower 95% interval', 'Upper 95% interval');
title('ARIMA Model with 95% Forecast Intervals for Log-Transformed Close Data');
xlabel('Date');
ylabel('Close values');  disp("The residuals do not exhibit significant autocorrelation
    .");
end
```

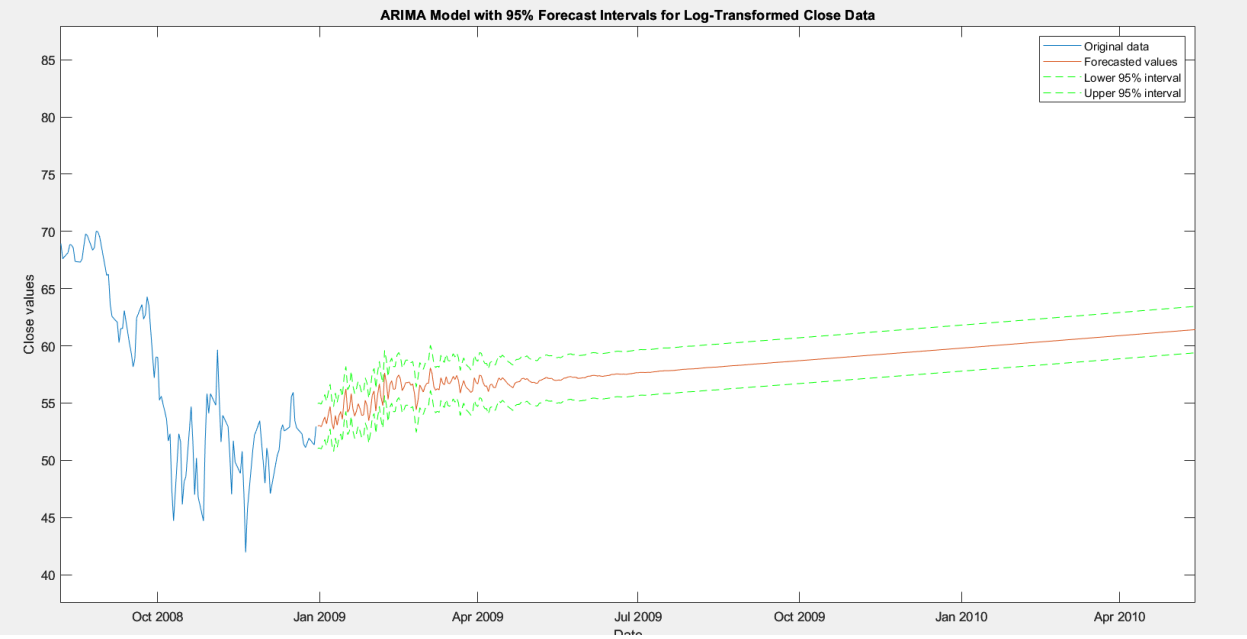


Figure 4:

By performing forecasting, we can use the estimated ARIMA model to generate predictions for future values of the time series. This can be useful for making informed decisions or planning based on expected future trends or patterns in the data.