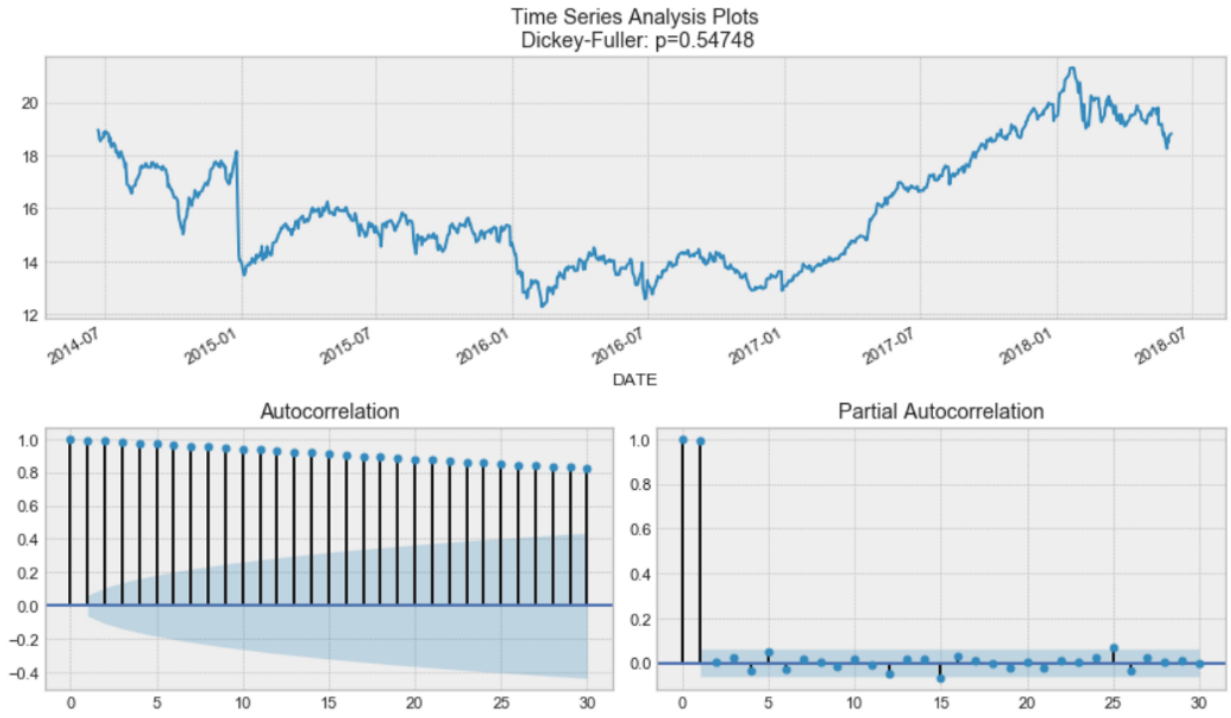
Time Series Analysis
Application based R Assignment

Kirtan Patel s2935848
University of Twente



[1]

Contents

1 Introduction

1.1 Background

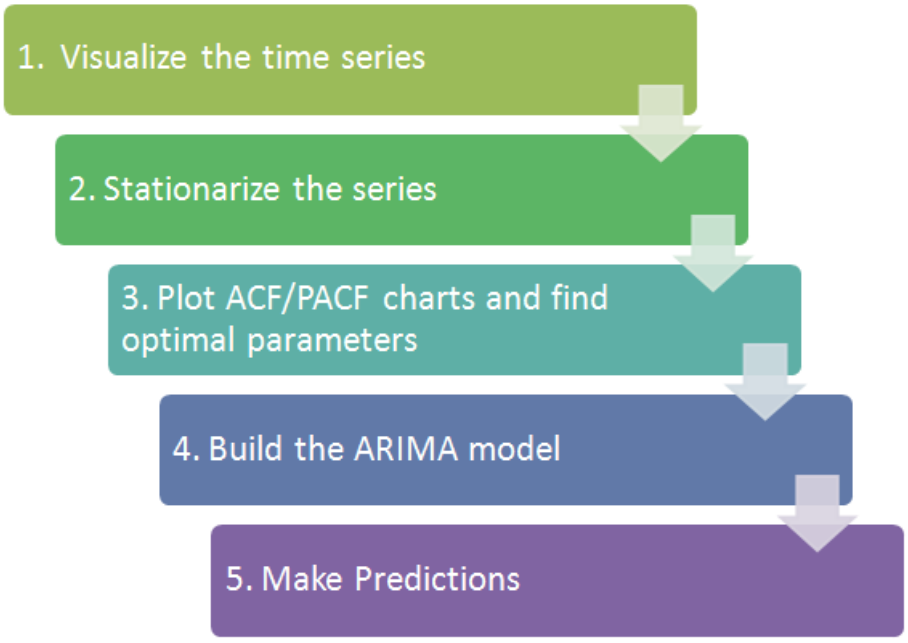
This report is encompasses practical Time Series Analysis carried out using R. The fundamentals of Time Series Analysis taught in the course number 191571090, Time Series Anaylsis, by Dr. Annika Betken.

The course covers important aspects of Time Series Anaylsis such as :

- Time Series Decomposition
- Stochastic Processes in Time Series
- Linear Processes and their Analysis
- ARMA Processes and their Analysis
- Estimation Theory
- Non-Parametric Time Series Analysis
- Estimation of ARMA Models

Time series analysis accounts for the fact that data points taken over time may have an internal structure (such as autocorrelation, trend or seasonal variation) that should be accounted for.

Understanding the Mathematics behind the underlying forces which model, govern and can be used to predict a series is an important part of complete analysis of a time series. The steps which constitute Time Series Anaylsis can be represented as follows:

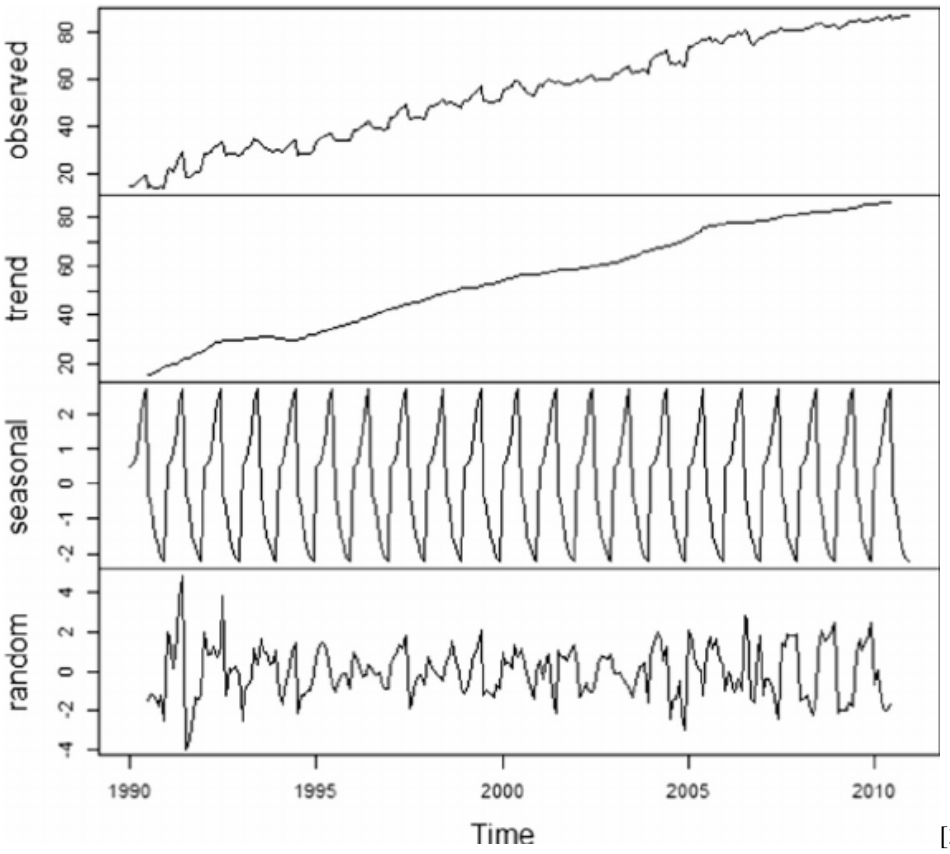


[2]

1.2 Time Series

A time series is a collection of observations of well-defined data items obtained through repeated measurements over time. The data collected for all practical purposes have an underlying force which governs it. To be able to predict the data at a future time, one must be able to model the data well.

To model a time series accurately, we decompose it into elements which we then model. A Time Series has the following constituents :



[3]

The Trend and Seasonal Component can be estimated with a large accuracy. The Random Component, also called the incidental component, needs to be modelling stochastically and can only be predicted with a certain probability.

Time Series Analysis majorly focuses on the analysis and modeling of this random component of a time series using Linear Processes, Stochastic Processes and White Noise.

1.3 Exercises

The Exercises solved in this report cover both the aspects of Time Series Analysis mentioned above.

Exercise 1 focuses on the decomposition of a given Time Series into its constituents. The method adopted to do so is explained in detail in the exercise.

Exercise 2 focuses on the analysis of 2 time series, beyond the decomposition. Mathematical Analysis has been carried out on the random component obtained after detrending the time series, and different ARMA models have been tested to fit the data.

1.4 Use of R

The R software for statistical computing and graphics is a common choice for data analysis and development of new statistical methods. R is available as Free Software under the terms of the Free Software Foundations's GNU General Public License in source code form. It compiles and runs on all common operating systems including Windows, MacOS X, and Linux.[4]

R is used highly owing to its powerful mathematical analytic capabilities and efficiency in dealing with numeric data. R provides different object classes to work with time series, and has a large number of in-built functions which are useful in the analysis of time series.

1.4.1 Object Class for Time Series in R

Before you start any time series analysis in R, a key decision is your choice of data representation (object class). This is especially critical in an object-oriented language such as R, because the choice affects more than how the data is stored; it also dictates which functions (methods) will be available for loading, processing, analyzing, printing, and plotting your data. When many people start using R they simply store time series data in vectors[5]. Vectors are easy to work with and manipulate for first time users. However, none of the advanced analytics for time series analysis work with simple vectors. Switching to using an object class intended for time series data makes the analysis easier and opens a gateway to valuable functions and analytics.

The base distribution of R includes a time series class called *ts*. This implementation itself is too limited and restrictive. However, the base distribution includes some important time series analytics that depend upon ts, such as the autocorrelation function (*acf*) and the cross-correlation function (*ccf*).

The book[5] recommends using the zoo or xts packages for representing time series data. They are quite general and meet the needs of most users. The *xts* implementation is a superset of *zoo*, so *xts* can do everything that zoo can do.

1.4.2 Spectral Analysis in R [6]

The periodogram is often hard to understand due to it poor distribution.Log-scaling often provides a better visualization and makes the spectrum more revealing. The log-scaling has some theoretical advantages, too. The periodogram values should be approximately normally distributed in the log scale. Sometimes a log scaling can be helpful in cleaning up low frequencies, too, because as log scaling will spread out the low frequencies and squish the high frequencies.

Smoothing the Periodogram with Moving Averages : Kernel Functions
There is a fundamental problem with the periodogram. Unlike most estimates you've encountered, such as the mean or a regression coefficient, which get more reliable as you collect more data, the periodogram does not get more reliable. As you collect more data, you add more periodogram points, but they are all just as noisy as before.

We are assuming that there is some underlying curve of spectral values, and that the periodogram estimates this. But the periodogram is noisy, and will always be noisy. We call this underlying curve the "spectral density function," or sometimes the "power spectrum."

The only way to get smooth estimates of the power spectrum is by taking moving averages of the periodogram. In essence, though, we want to give more weight to close frequencies, and little weight to far away frequencies. There are different ways to create weights. You could use a bell curve shape (or a triangle, or a rectangle) to give weights. These are called 'kernel functions.' There are many different kernal functions. Some of them are :

```
kernel("daniell", m = 10)  # A short moving average
kernel("daniell", m = 50)  # A long moving average
kernel("daniell", c(5, 5))  # m=5 moving average of a m=5 moving average
kernel("daniell", c(5, 5, 5))  # a m=5 moving average of that!
kernel("daniell", c(5, 5, 5, 5))  # a m=5 moving average of that!
kernel("daniell", c(3, 3, 21))
```

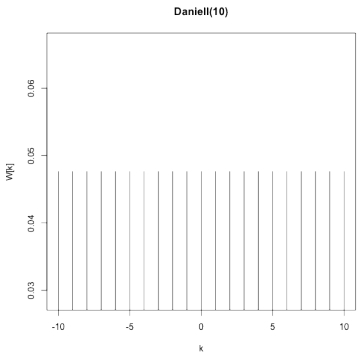


Figure 1: (“daniell ”, m = 10)

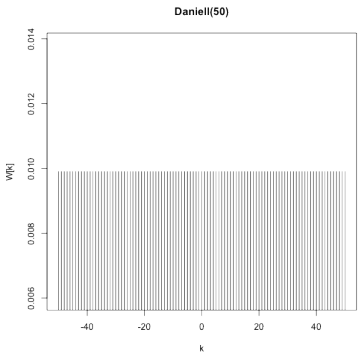


Figure 2: (“daniell ”, m = 50)

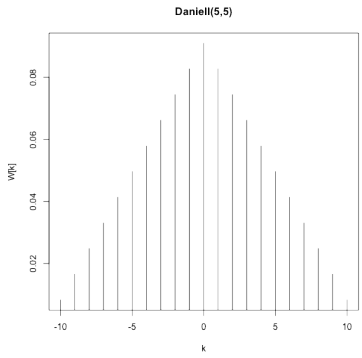


Figure 3: (“daniell ”, c(5,5))

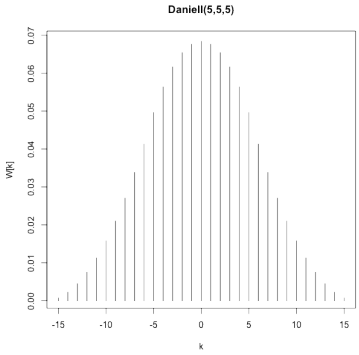


Figure 4: (“daniell ”, c(5,5,5))

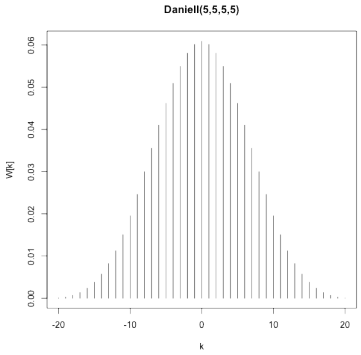


Figure 5: (“daniell ”, c(5,5,5,5))

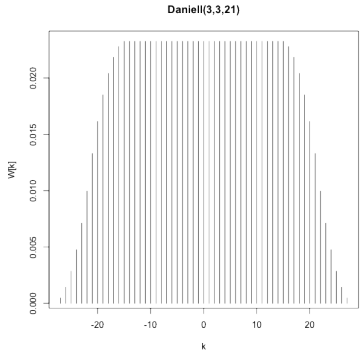


Figure 6: (“daniell ”, c(3,3,21))

Figure 7: Different Kernel Function Plots

Tapering

Besides windowing, there is one other ’trick’ commonly done when spectral estimating, called tapering. Before describing tapering, let’s discuss the problem.

When you estimate a periodogram, you are implicitly making the assumption that your time series is circular, i.e. that you could wrap the time series around and just keep time marching on until infinity. Obviously, this isn’t so. If you wrap the time series around, there will be a jump where the end meets the start again. This jump is spurious, but it will propagate itself through all the frequencies, contaminating them.

The solution is to downweight the beginning and end of the data. This way, when you calculate the periodogram, you’ll be giving more weight to the middle, and less weight to the ends. There is still the jump at the end, but it has very little weight, so it’s effect is diminished. This downweighting is called tapering.

In practice, a 5% (from each side) (corresponding to a taper of 0.05) often works pretty well. Tapering is less important the longer your time series is, but it can be very important in short series.

I’m not going to cover confidence intervals. The default plotting shows a confidence interval. But, in general, it is difficult to construct meaningful confidence intervals of spectral density estimates

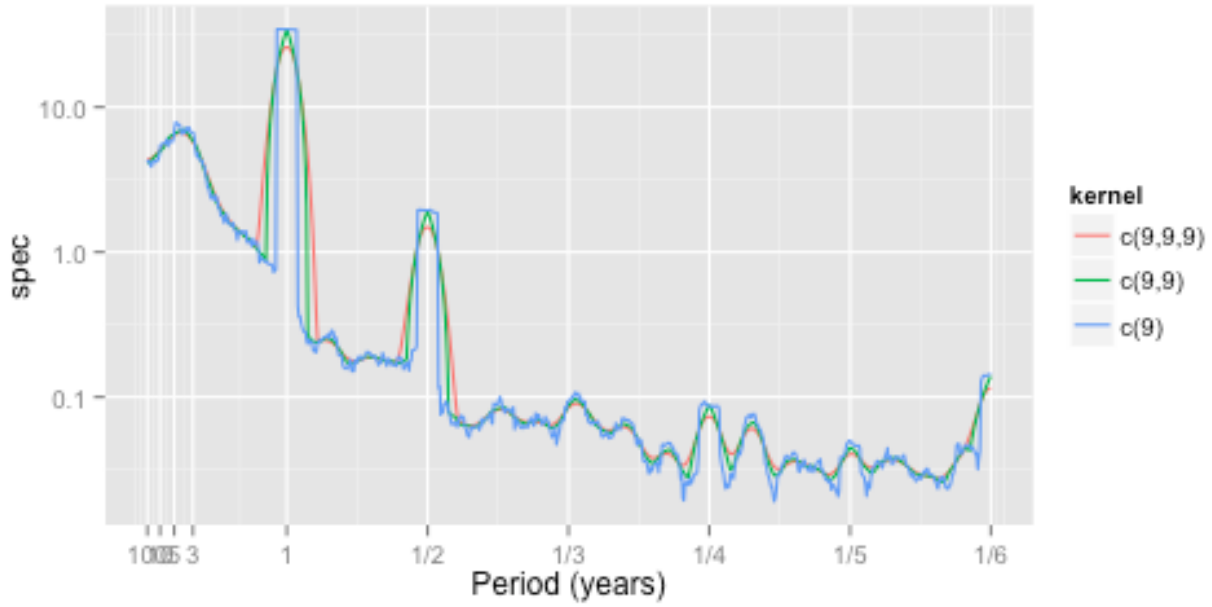


Figure 8: Effect of Moving Average Smoothing

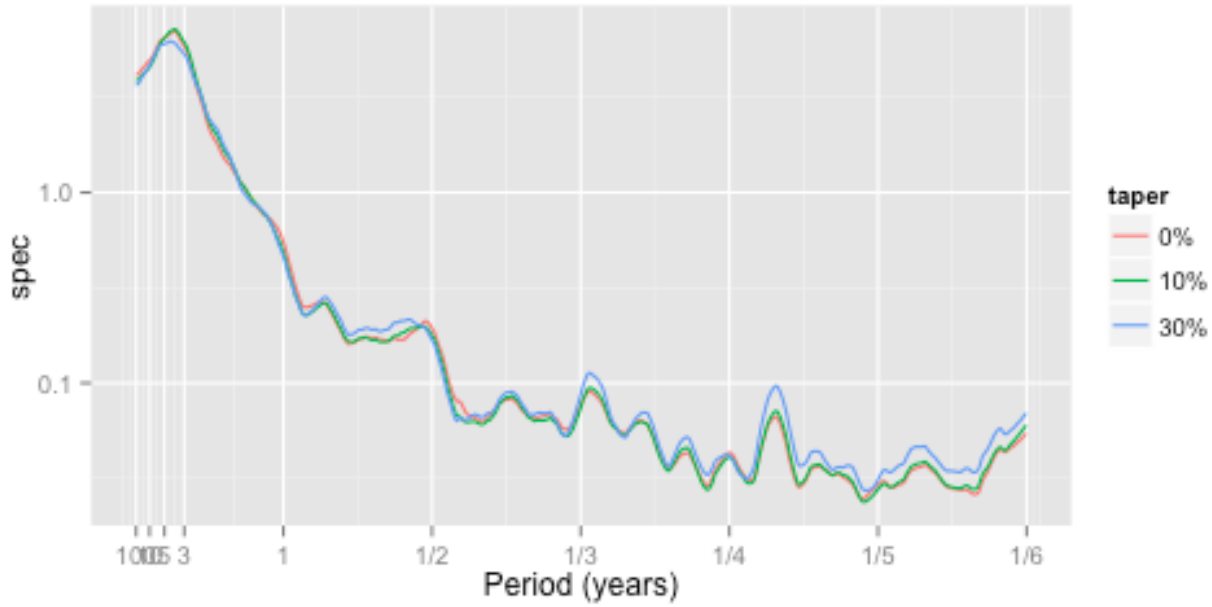


Figure 9: Effect of Tapering

The Analysis presented in this report exhibits the results obtained using Moving Average Smoothening. The attached code in the Appendix can be modified further to include the effects presented by tapering.

1.5 ARIMA Models

For models whose seasonality period is not known, we need to resort to differencing to elimate the seasonal component. A variant of the ARMA model called the ARIMA is useful when the time series is nonstationary and displays a stochastic trend. Therefore, we should first differentiate the time series before applying the ARMA model on the differentiated time series.

ARIMA stands for AutoRegressive Integrated Moving Average and is specified by three order parameters: (p, d, q) and is noted ARIMA(p,r,q), where p is the number of lags for the autoregressive part, q the number of lags of the Moving average part and r is the number of time we should differentiate in order to obtain a stationary ARMA model.

The R Programming Language provides and inbuilt function which calculates the constants (with a certain standard deviation) of an ARIMA Process for a process. This function requires manual selection of ARIMA Model. There exists another function (*auto.arima()*) which has automated optimal model selection. The function uses a variation of the Hyndman-Khandakar algorithm[7], which combines unit root tests, minimisation of the AICc and MLE to obtain an ARIMA model. The arguments to auto.arima() provide for many variations on the algorithm.

The steps to estimate an ARIMA Model manually are stated online[8]. When fitting an ARIMA model to a set of (non-seasonal) time series data, the following procedure provides a useful general approach.

1. Plot the data and identify any unusual observations.
2. If necessary, transform the data (using a Box-Cox transformation) to stabilise the variance.
3. If the data are non-stationary, take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an ARIMA(p, d, 0) or an ARIMA(0, d, q) model appropriate?
5. Try your chosen model(s), and use the AIC to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
7. Once the residuals look like white noise, calculate forecasts.

The process is summarised in a flowchart attached in the Appendix. The Hyndman-Khandakar algorithm only takes care of steps 3–5. So even if you use it, you will still need to take care of the other steps yourself.

1.6 Model Validation and Residual Diagnostics

After fitting the model, we should check whether the model is appropriate. As with standard non-linear least squares fitting, the primary tool for model diagnostic checking is residual analysis. The most basic residual plot is the plot of standardized residuals against time.

It is a general principle of time series models (including ARMA) that you would like to capture all systematic dynamics in the data. This means, every variation that can be explained by input variables, auto-regression, moving-average etc. should be explained.

Being pragmatic, the residuals of an ARMA process are by defintion white noise. If you fit your data to an ARMA model, and you see that residuals are white noise, then this indicates you that the model fits the data well. If there is correlation in the residuals for the ARIMA Model, the It seems that your model is not yet the best model.

Sometimes there may be seasonality in the data although the series is stationary. This can be tried to overcome by using *Seasonal ARIMA or SARIMA* Models. If seasonality is an issue in your data this method will help in obtaining a better model with white-noise error term. Seasonality can even be removed by differencing, but that uses an ARIMA Model, and hence this report will not use differencing. Else, it might be resolved by increasing ARIMA Order (since out assignment limits the ARMA orders to 5). Furthermore, for our report, we will find ARMA Model Fits, and hence the order of differncing will be kept as 0.

2 Air Passengers Data Model Decomposition

2.1 Introduction

The *AirPassengers* dataset present in the *Rdataset* library contains monthly totals of international airline passengers from 1949 to 1960. It is given to us that the time series follows a multiplicative model i.e $x_t = m_t s_t w_t$ with m_t denoting a trend, s_t a seasonal component, and w_t an incidental component i.e a stationary stochastic process.

This can even be perceived form the plot of the raw data as the fluctuations increase with time. This is indicative of a multiplicative model. The cause of increasing fluctuations then is the increasing trend factor m_t.

The seasonality is also quite evident from the plot of the raw data, and with the information given to us, we can identify that the data has a period of 12 *(P = 12)* and the number of seasons in the data is also 12 *(M = 12)*. This is cross-verified with the total number of data-points in the time series, being 144 *(N = P*M = 144)*.

2.2 Analysis

2.2.1 Observation

For Preliminary Observations, we plot the raw data for elementary inferences.

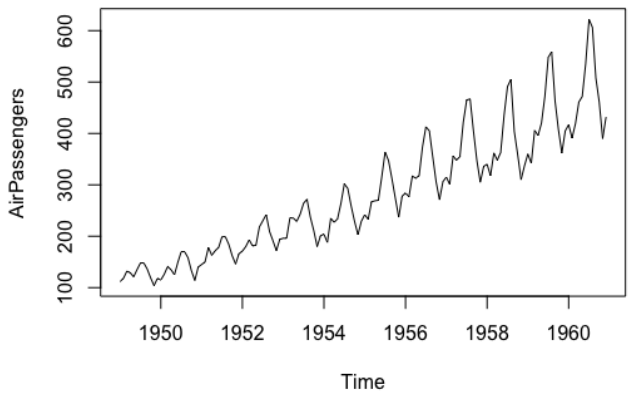


Figure 10: Air Passenger Data

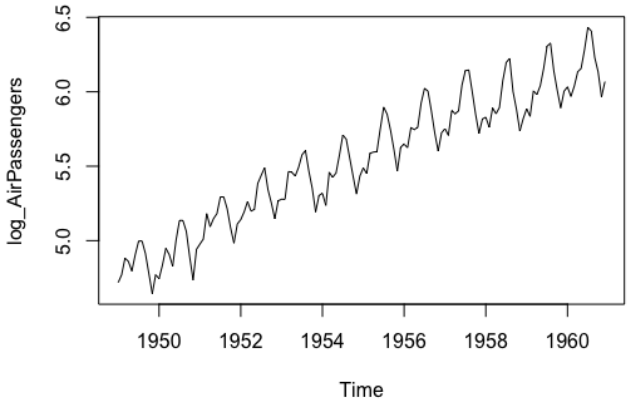


Figure 11: log(Air Passenger Data)

From Figure 13 we can conclude that the model is indeed multiplicative, and to further decompose the model into its constituents, we seperate them by taking log. Plotting the data after taking log, we obtain Figure 12.

Figure 12 clearly indicates the presence of a linear trend, a seasonal component and an incidental component. Expressing the original model of the raw data to be

$$\widetilde{x}_t = \widetilde{m}_t.\widetilde{s}_t.\widetilde{w}_t$$

the model after taking the log of the dataset becomes

$$log(\widetilde{x}_t) = log(\widetilde{m}_t) + log(\widetilde{s}_t) + log(\widetilde{w}_t)$$

which we will now assume to be our model, indicated as

$$x_t = m_t + s_t + w_t$$

2.2.2 Time Series Decomposition

The Time Series Decomposition is an algorithmic process which enables us to separate the additive components in a time series. For a dataset *x*, it consists of the following steps :

1. De-trending the dataset *x*

$$initial \;\; dataset \;\; x_t = m_t + s_t + w_t$$

$$modified \;\; dataset \;\; z_t = detrended(x_t) = s_t + w_t$$

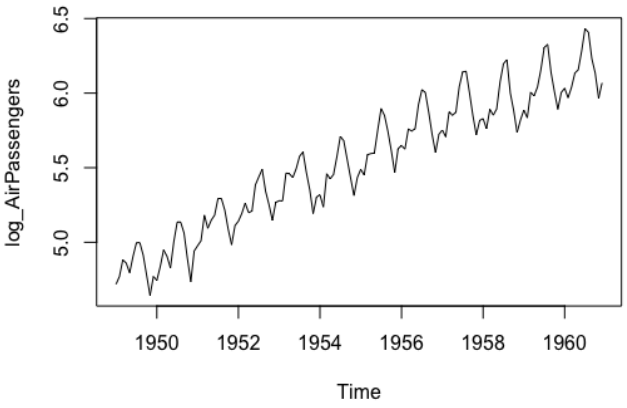


Figure 12: Initial Dataset

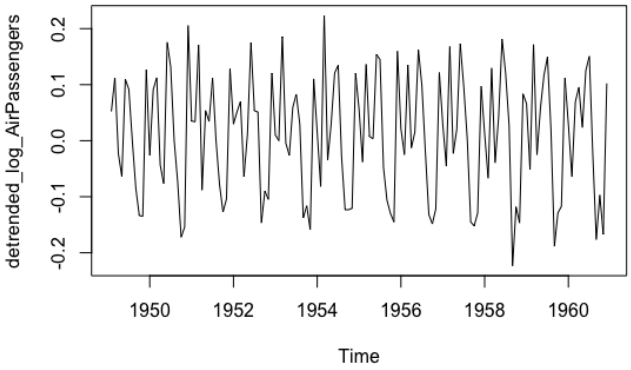


Figure 13: Detrended Dataset

2. Eliminating Non-Zero Mean
   If the detrended process has a non-zero mean, we subtract that form the modified dataset as well. This is because, we consider seasonal and incidental components to have zero mean, and the constant additive factor is considered to be a part of the trend.

$$\tilde{m} = mean(z)$$

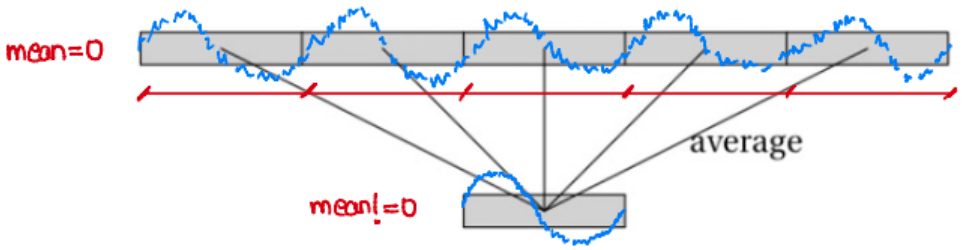
$$z_t - \tilde{m} = s_t + w_t$$

   where, both the seasonal component *s* and the incedental component *w* have zero mean.

   In the log(*AirPassengers*) data set, after detrending using the *detrend()* function, the mean is of the order of 1e-15, and hence can be approximated as 0.

3. Estimating Seasonal Component
   Assuming that the frequency of the incidental componenet is considerably greater than the frequency of the seasonal component, averaging across seasons almost removes all noise, and gives us a good approximation of the seasonal component.

   This is done but dividing the dataset into seasons, and taking the average value across all seasons, to give an average seasonal component spanning 1 season. This component might not have zero-mean and hence the mean is calculated and subtracted to obtain a zero-mean seasonal component spanning 1 season.



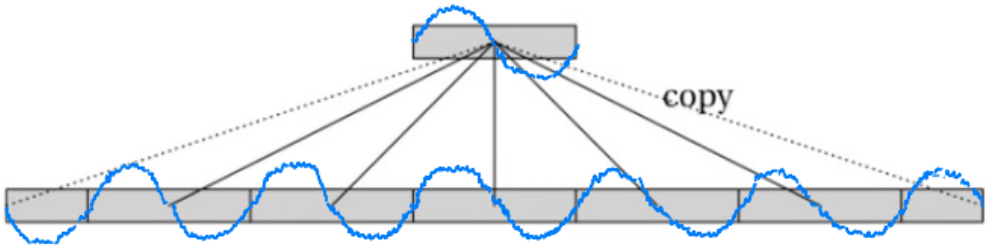
Let the non-zero mean unit-span seasonal process be denoted by s^0.

The zero-mean unit-span season process is them calculated as

$$s_{unit} = s^0 - mean(s^0)$$

In our data set, after averaging out the seasonal spans, the mean is of the order of 1e-16, and hence can be approximated as 0.

The seasonal component of the dataset is then obtained by replicating the zero-mean unit-span seasonal process equal to the number of seasons. This process is truly periodic, and is the periodic component s of out dataset. The following figures show the result of the process on our dataset.



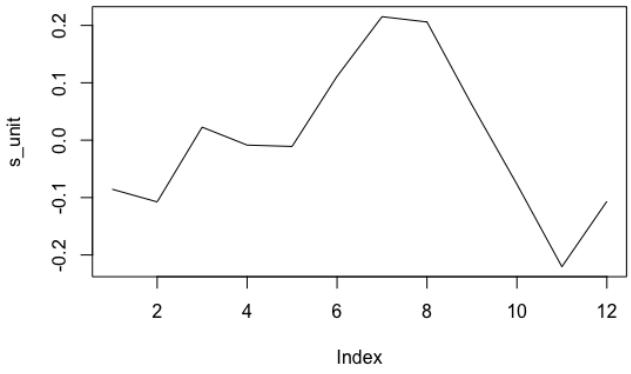


Figure 14: Zero-Mean Unit-Span Seasonal Process

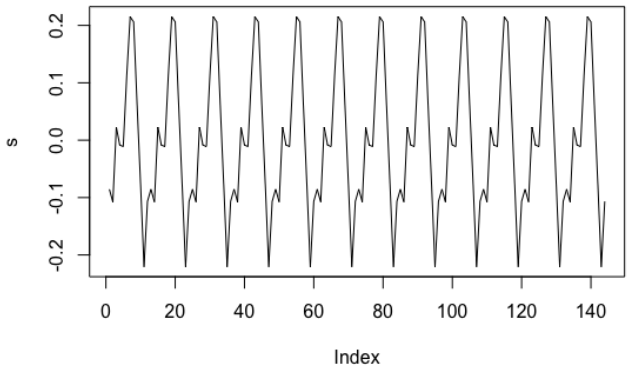


Figure 15: Periodic Component

4. Estimating Incidental Component
   For our dataset which we had asssumed to be of the model

$$x_t = m_t + s_t + w_t$$

   we know x_t and we have estimated s_t. We also know that the *detrend()* function eliminates m_t. Thus, we can obtain the incidental component by

$$detrend(x_t - s_t) = detrend(m_t + w_t) = w_t$$

   To ensure that the incidental component is zero-mean, we eliminate the mean from it

$$w_t = detrend(x_t - s_t) - mean(detrend(x_t - s_t))$$

5. Estimating Trend
   For our dataset which we had asssumed to be of the model

$$x_t = m_t + s_t + w_t$$

   we know x_t and we have estimated s_t and w_t. Thus, the trend m_t can be estimated by

$$x_t - s_t - w_t = m_t$$

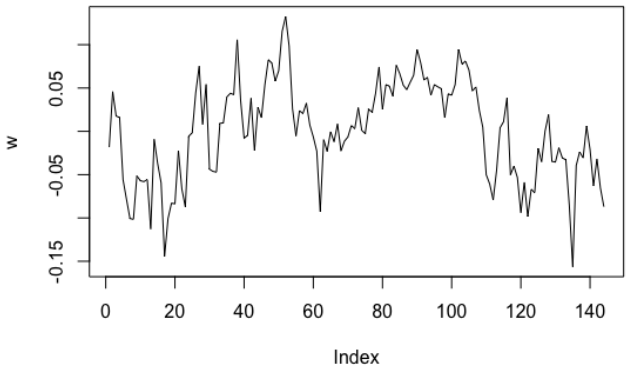


Figure 16: Incidental Component

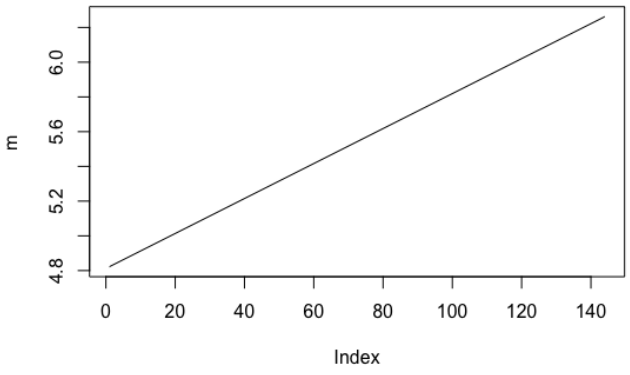


Figure 17: Trend

2.3 Results

We can see that after taking log of the *AirPassengers* dataset, the modified dataset conforms to the additive model of time series. It consists of a linear trend, a 12 period (annual) seasonal component and an incidental component.

Both, the seasonal and the incidental components have zero-mean and the mean is captured in the trend.

The process has been obtained from the TSA Coursebook[9] and results obtained can be reproduces using the code in the Appendix.

3 Dow Jones Index Analysis

3.1 Preliminary Observation

The data has been plot for preliminary observation. It can be seen that the fluctuations increase as down the time series (as seen in Fig.18), which suggests a multiplicative model. To convert it to an additive model, for easier analysis, the log of the data has been taken (as seen in Fig.19).

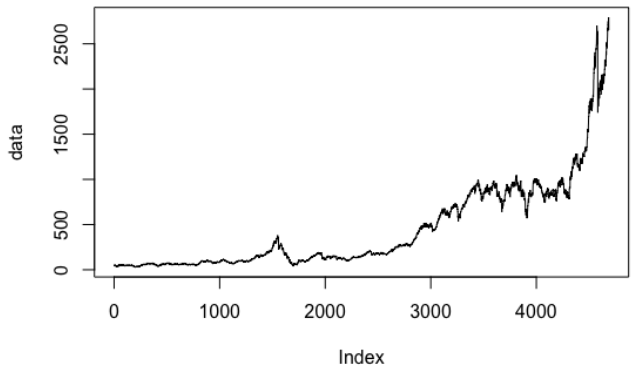


Figure 18: Raw Data Plot

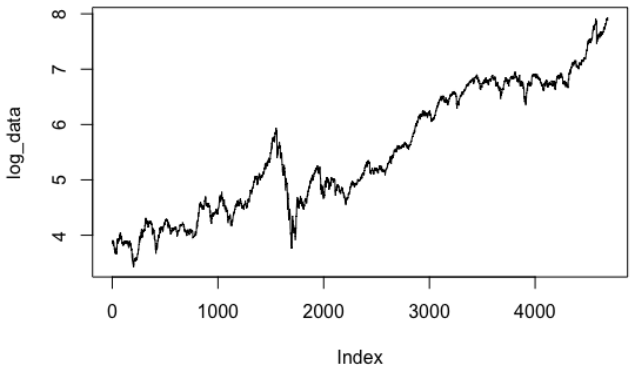


Figure 19: log(Raw Data) Plot

Figure 20: Plots for Preliminary Observation

Figure 19 shows a clear and trend and hence was detrended using the built-in *detrend()* function present in R. After being detrended, the mean was calculated and eliminated, for the given dataset, the mean for the log of the data was negligible and this can be seen in the plots below. There is a significant drop in the series which can be attributed to a rare event. Furthermore, a weak seasonality can be percieved, which can be removed using Seasonal ARIMA (*SARIMA*) modelling in R.

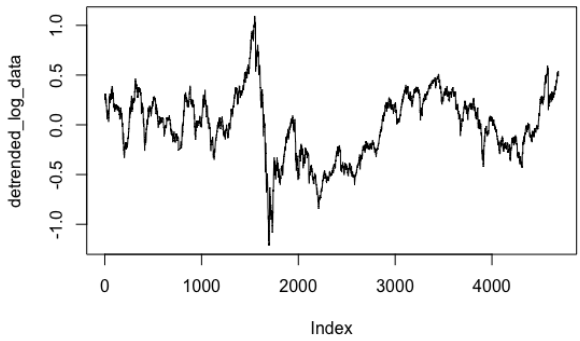


Figure 21: Detrended log(Raw Data)

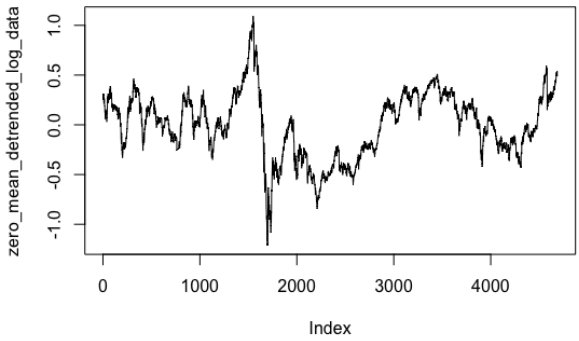


Figure 22: Zero-Mean Detrended log(Raw Data)

Figure 23: Plots for Preliminary Observation

3.2 Correlation Functions

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots help us analyse the data correlations and estimate the order of ARMA process.

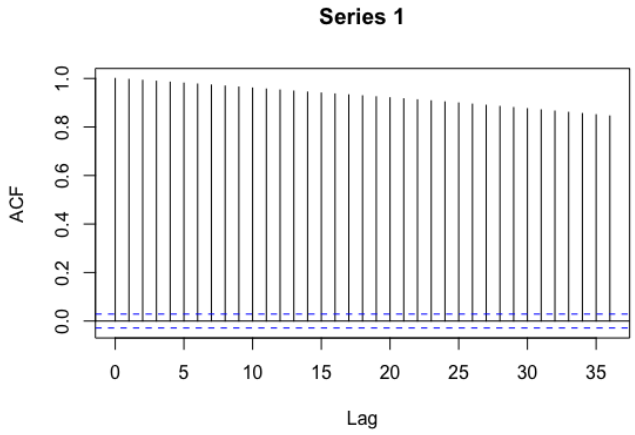


Figure 24: ACF Plot

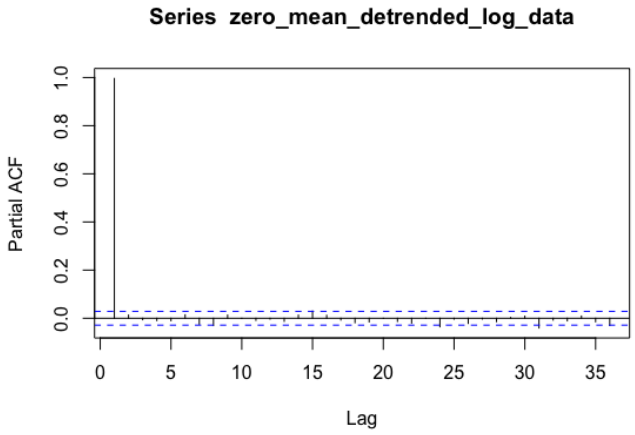


Figure 25: PACF Plot

Figure 26: Autocorrelation Functions Plots

The ACF plot displays weak geometric decay (can be seen by plotting the ACF till 500) whereas the PACF cuts off after 1-lag (0-lag has not been plotted). This suggests an AR(1) model or an ARMA(1,0) model. For the arima() function it implies an ARIMA(1,0,0) Model. This can further be verified using the Akaike Information Criterion (AIC) presented for different models by the arima() function.

The Akaike information criterion (AIC) is a mathematical method for evaluating how well a model fits the data it was generated from. It is used to compare different possible models and determine which one is the best fit for the data. AIC is calculated from:

- the number of independent variables used to build the model.
- the maximum likelihood estimate of the model (how well the model reproduces the data).

The best-fit model according to AIC is the one that explains the greatest amount of variation using the fewest possible independent variables.

3.3 ARMA Model Selection

The best estimate for a model would be one with the least AIC value (given the upper bound for the order of ARMA process is 5). The values can be reproduced using the code attached in the Appendix.

Calculating the AIC values for all models upto the ARMA(5,5), it can be seen that the ARMA(1,0) has the least AIC value and hence is the best model for the given dataset. This complies with the observation made from the ACF anf PACF plots. The values can be referred in the Appendix.

According to the results, we chose the model :

$$\boxed{X_t = 0.9966 X_{t-1} + \epsilon_t}$$

with ϵ_t being a zero-mean white noise with variance 0.0007594

3.4 Residual Diagnostics

A well-fit model is characterized by its residuals being white noise, thus to verify the model, we analyse it's residuals.

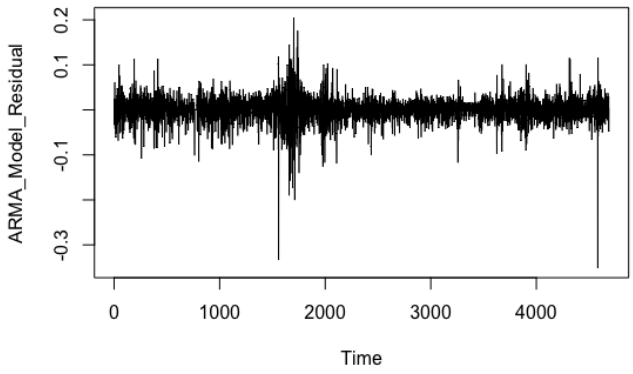


Figure 27: Residuals Plot

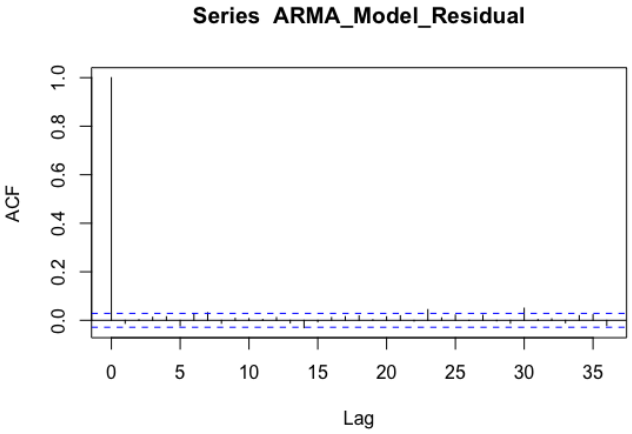


Figure 28: ACF of Residuals

Figure 29: Residual Diagnostics Plots

From the figure, the residuals can be considered as white noise. The mean is 4.297635e-05, and can be considered to be negligible. The ACF Plot confirms that the residuals are indeed white noise. The ACF Plot deplays only 1 peak, at 0-lag, while the others can be considered insignificant, since they lie below the significance interval.

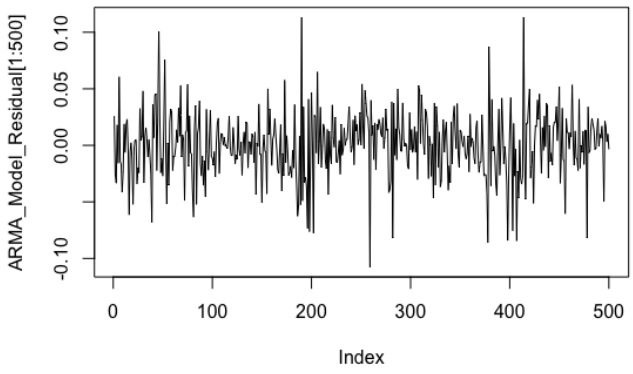


Figure 30: Time Frame 1:500

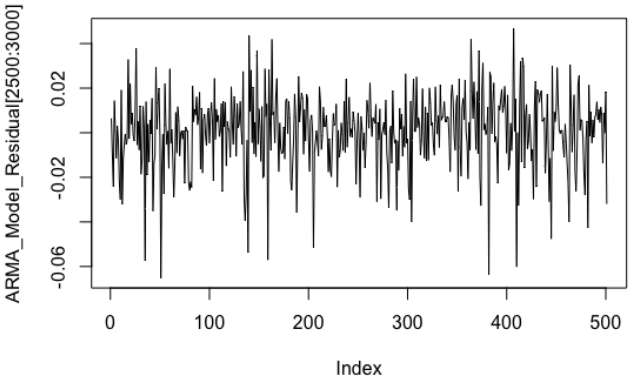


Figure 31: Time Frame 2500:3000

Figure 32: Residual Plot

Thus is it verified that the estimated ARMA Model, is a good fit for the data.

3.5 Spectral Density Function

The periodogram is hard to extract inference from hence taking a log of the periodogram is important in revealing its features.

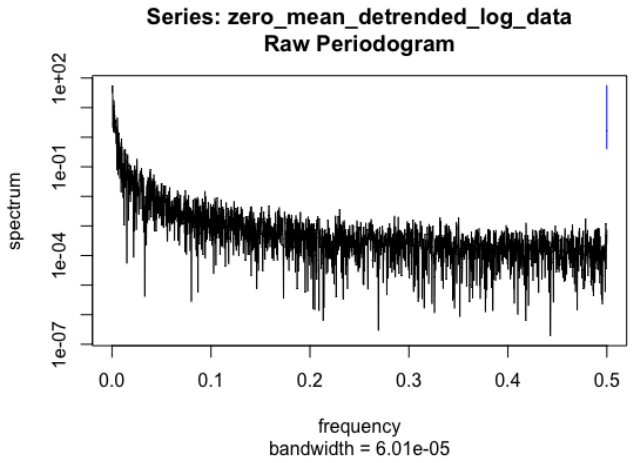


Figure 33: Raw Periodogram

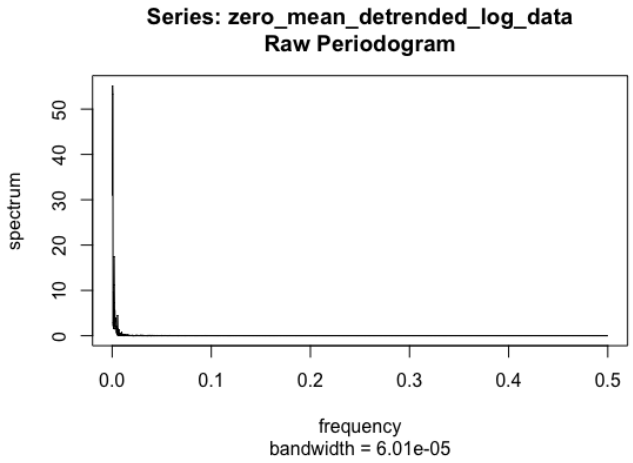
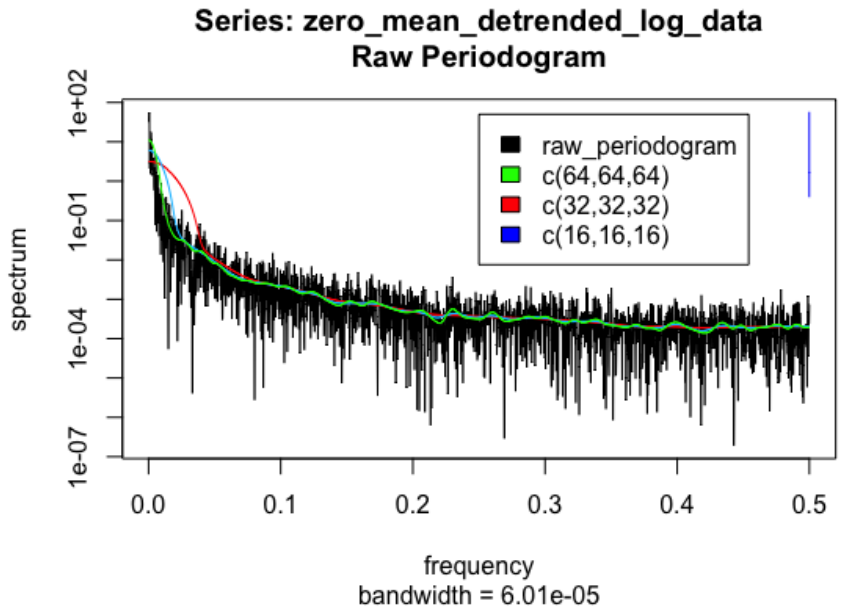


Figure 34: log(Periodogram)

Figure 35: Periodogram Plots

It is evident that low frequencies are more prominent than any high frequencies in the data. To estimate the Spectral Density Function, we apply window on the Periodogram. This is done to smooth out the periodogram. We do this for different window sizes and using different kernels provided by R.

For error to vanish as $N \to \infty$, the window size M must be such that $M << N$. Thus, for $N = 4687$, the window sizes were varying from $M = \sqrt[3]{N}$ to $M = \sqrt{N}$ were selected. Optimizing the Time-Window Size, only powers of 2 were selected, although this is not mandatory. Thus, the window sizes used were *16,32 and 64*.



4 ECG Analysis

4.1 Preliminary Observation

For the preliminary observation, we plot the data. We also detrend it and make the mean zero.

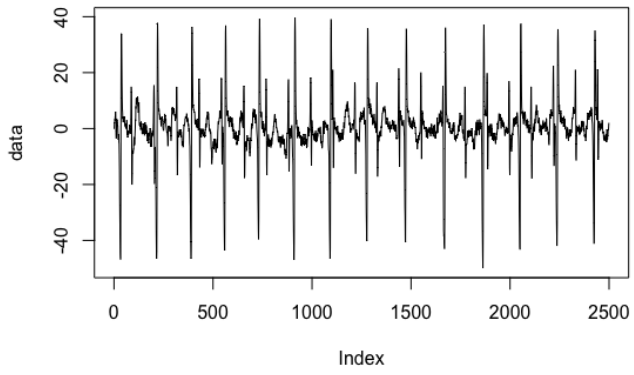


Figure 36: Raw Data Plot

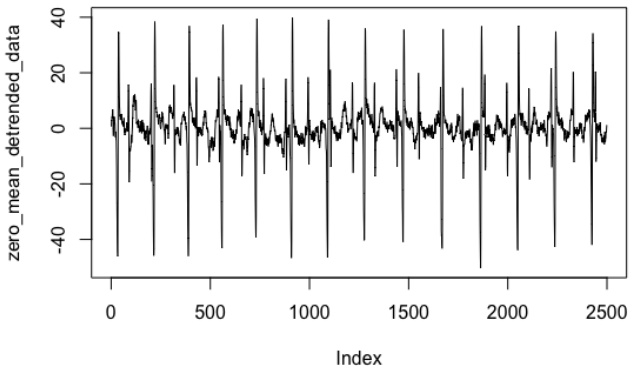


Figure 37: Zero-Mean Detrended Data Plot

Figure 38: Plots for Preliminary Observation

There is a certain repetitive pattern visible in the data, also since we are using ARMA models, we will not difference them away. The Data looks pretty dense near the mean which indicates a steep probability density function.

4.2 Correlation Functions

The ACF plot is decaying, hence AR component is present. The PACF is not abruptly reduced to zero and a slight decay is visible in it as well, hence the model is expected to be ARMA. To get the order of ARMA Process, we use the AIC value of ARMA Model.

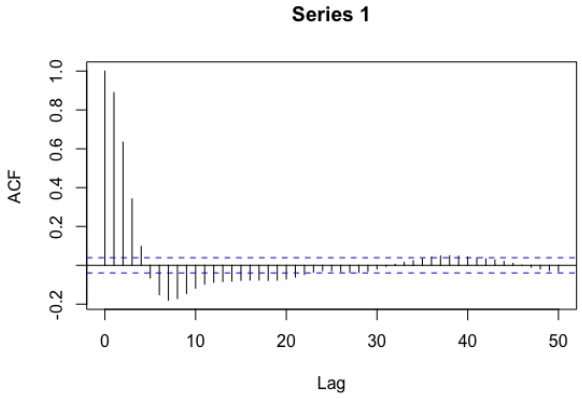


Figure 39: ACF Plot

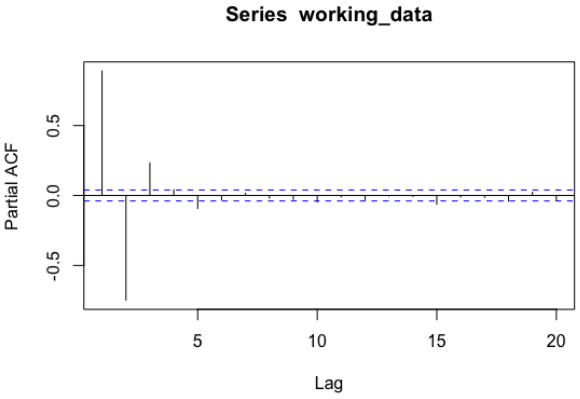


Figure 40: PACF Plot

Figure 41: Autocorrelation Functions Plots

4.3 ARMA Model Estimation

Calculating the AIC values for all models upto the ARMA(5,5), it can be seen that the ARMA(4,5) has the least AIC value and hence is the best model for the given dataset. ARMA(5,4) is a similarly good model. This complies with the observation made from the ACF and PACF plots. The values can be referred in the Appendix.

According to the results, we chose the model :

$$\boxed{1.1436X_{t-1} + 0.4225X_{t-2} - 0.943X_{t-3} + 0.318X_{t-4} + 0.57\epsilon_{t-1} - 0.4815\epsilon_{t-2} - 0.4158\epsilon_{t-3} - 0.3354\epsilon_{t-4} - 0.0905\epsilon_{t-5} + \epsilon_t}$$

with ϵ_t being a zero-mean white noise with variance 7.557

4.4 Residual Diagnostics

A well-fit model is characterized by its residuals being white noise, thus to verify the model, we analyse it's residuals.

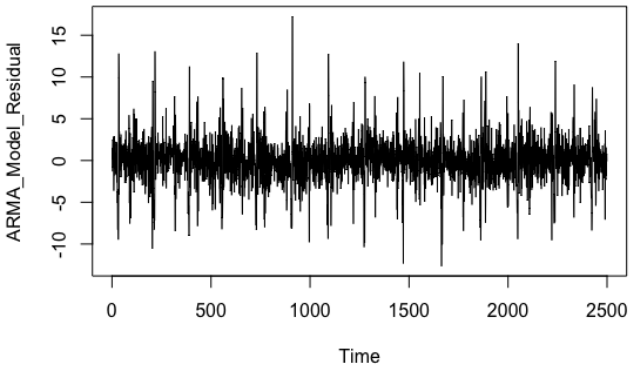


Figure 42: Residuals Plot

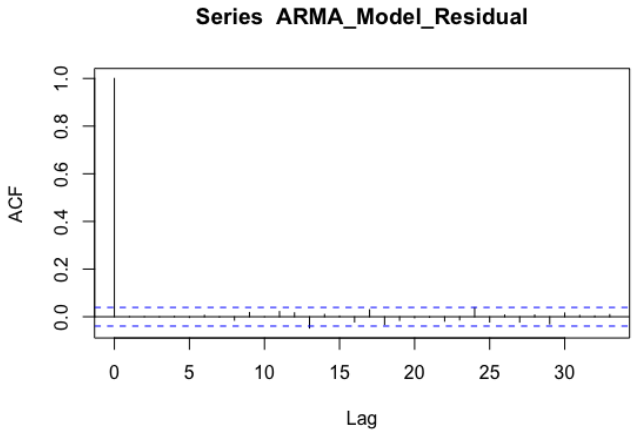


Figure 43: ACF of Residuals

Figure 44: Residual Diagnostics Plots

From the figure, the residuals can be considered as white noise. The mean is -6.724934e-05, and can be considered to be negligible. The ACF Plot confirms that the residuals are indeed white noise. The ACF Plot deplays only 1 peak, at 0-lag, while the others can be considered insignificant, since they lie below the significance interval.

4.5 Spectral Density Function

The periodogram is hard to extract inference from hence taking a log of the periodogram is important in revealing its features.(Figure 47)

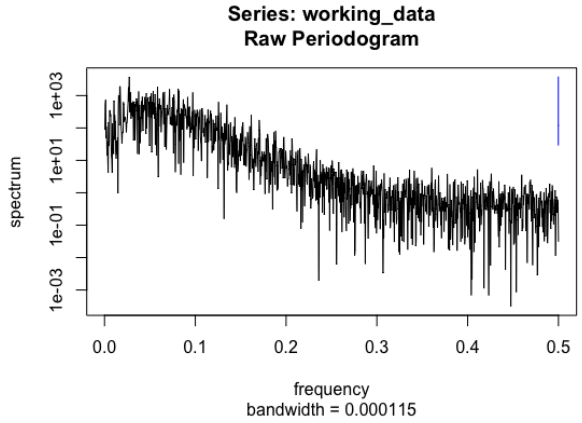


Figure 45: Raw Periodogram

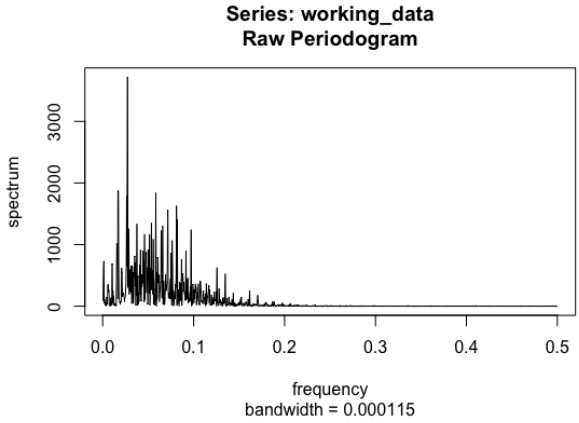
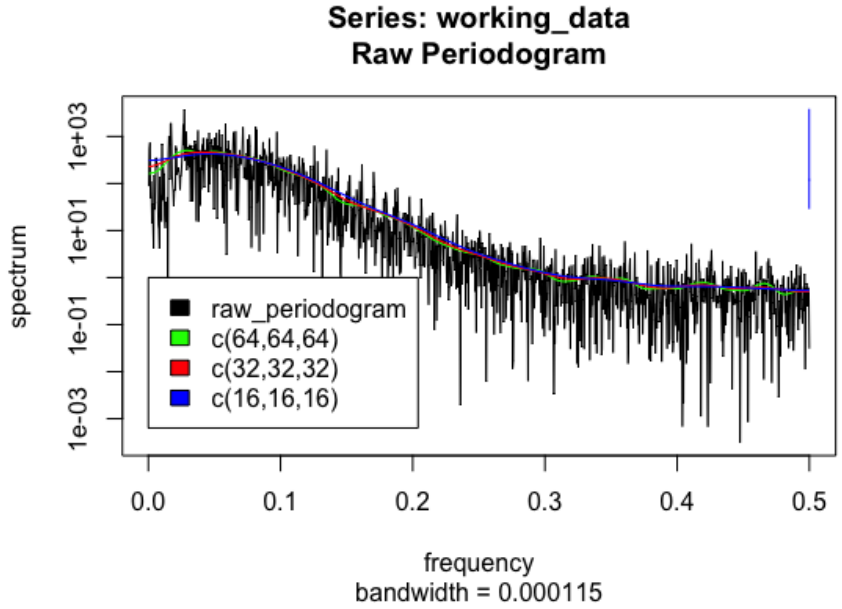


Figure 46: log(Periodogram)

Figure 47: Periodogram Plots

It is evident that low frequencies are more prominent than any high frequencies in the data. To estimate the Spectral Density Function, we apply window on the Periodogram. This is done to smooth out the periodogram. We do this for different window sizes and using different kernels provided by R.

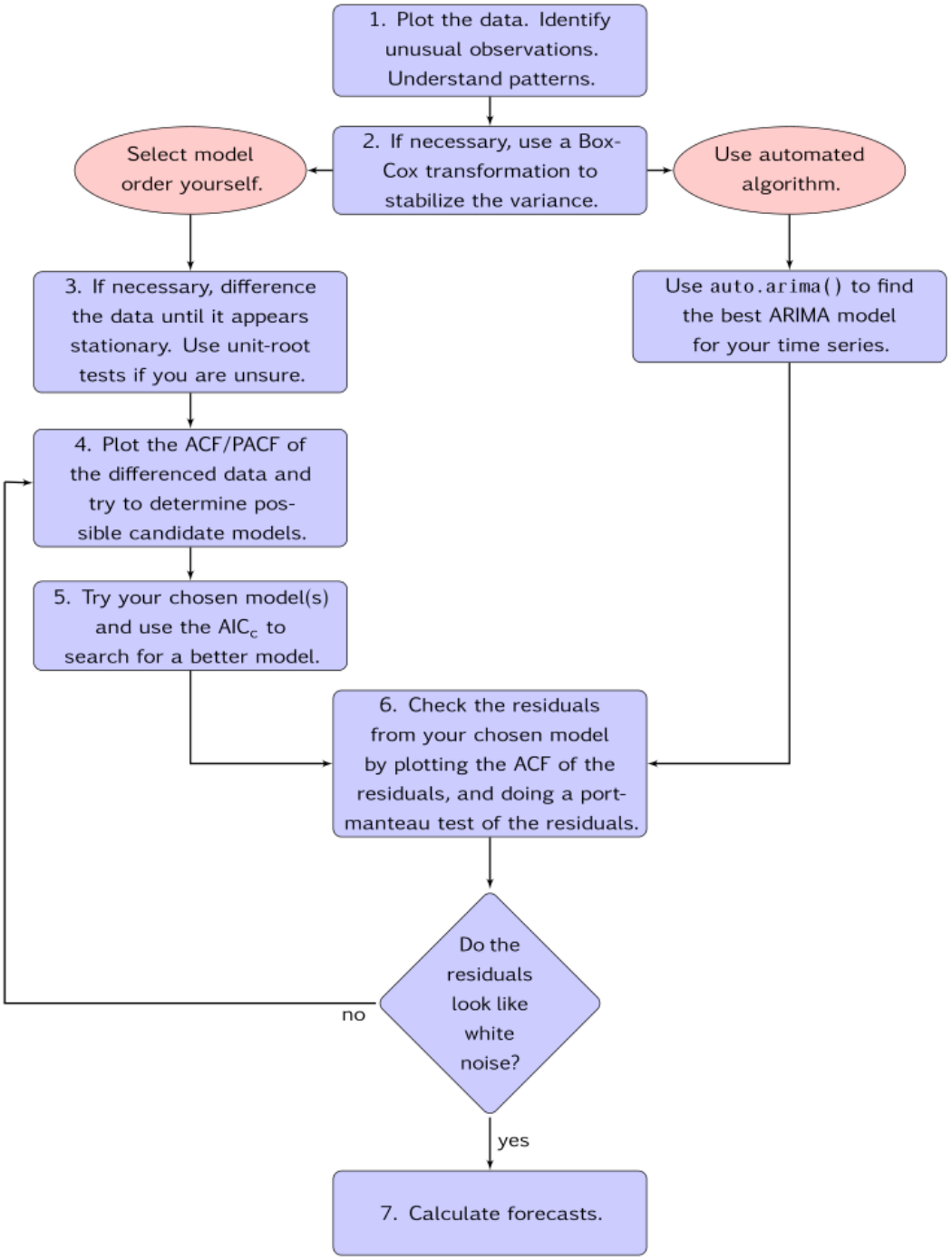
For error to vanish as $N \to \infty$, the window size M must be such that $M << N$. Thus, for $N = 2499$, the window sizes were varying from $M = \sqrt[3]{N}$ to $M = \sqrt{N}$ were selected. The window sizes used were *14,30 and 50*.



Other plots are attached in the Appendix for reference.

5 Appendix

5.1 ARIMA Modelling Flowchart



[8]

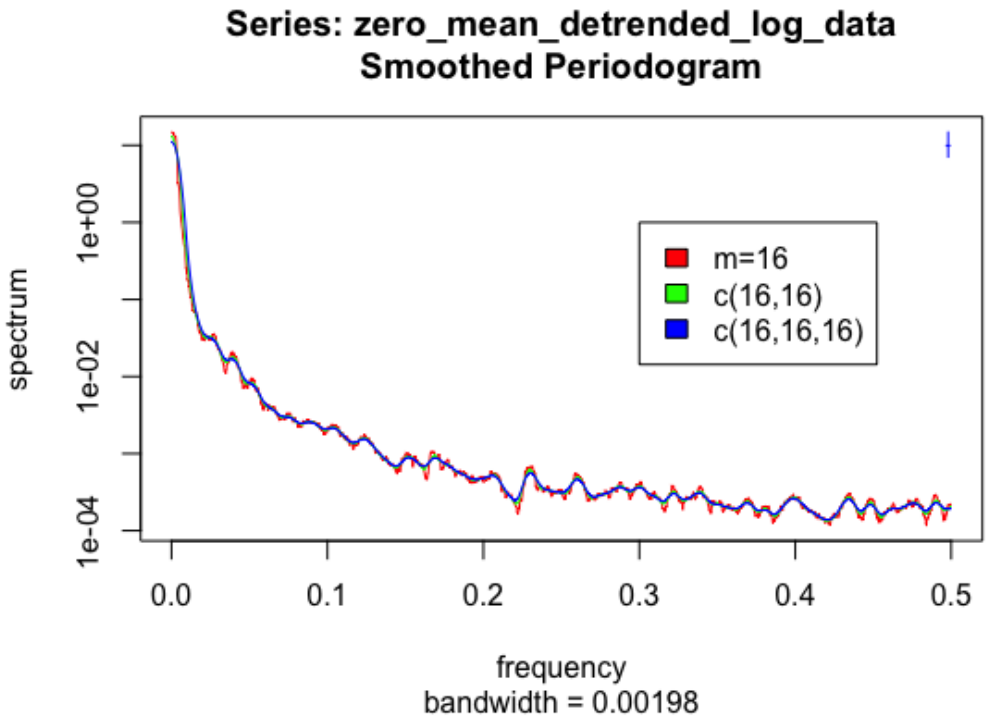
Figure 48: ARIMA Modelling Flowchart

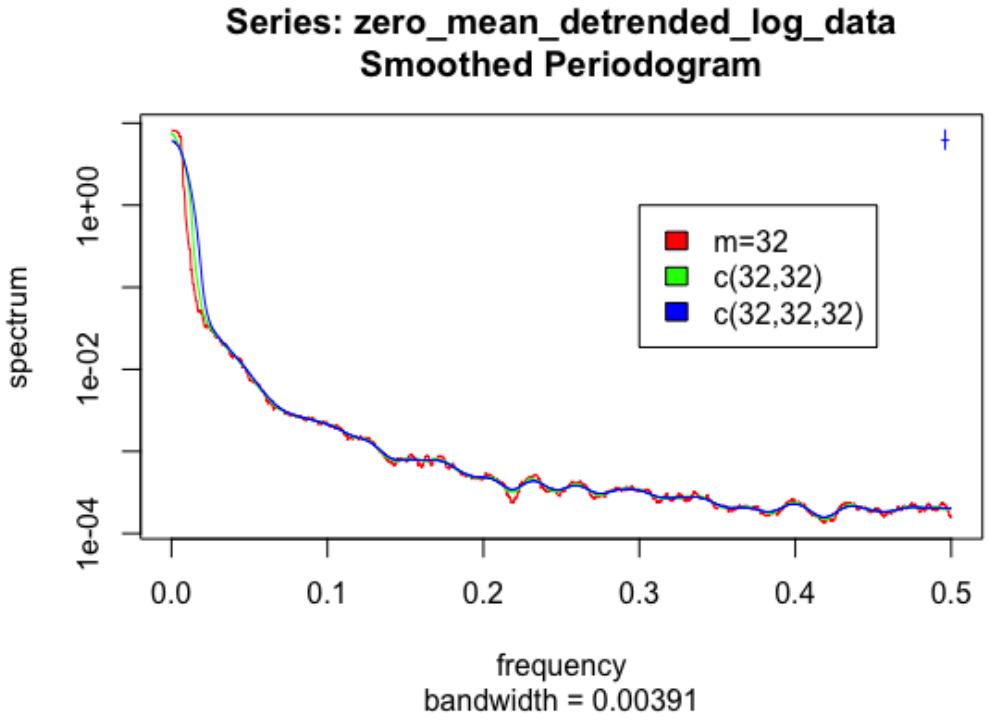
5.2 Dow Jones Index Analysis ARMA Model Data

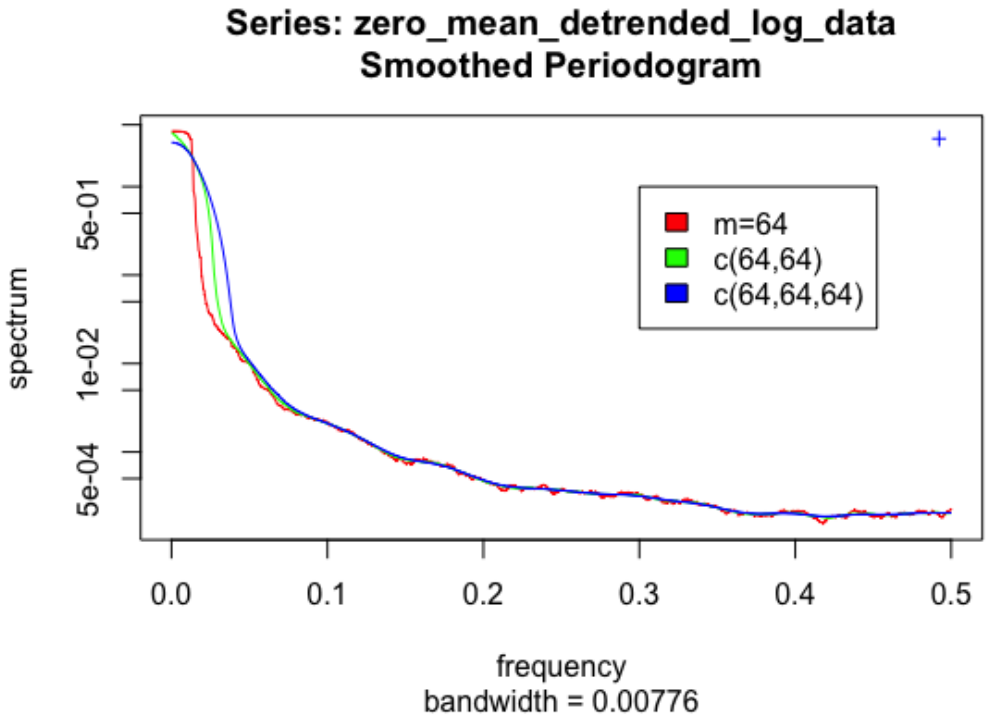
AIC	AR(0)	AR(1)	AR(2)	AR(3)	AR(4)	AR(5)
MA(0)	2,807.06	**-20,354.34**	-20,353.04	-20,351.07	-20,349.8	-20,348.88
MA(1)	-3,129.68	-20,353.04	-20,351.04	-20,349.22	-20,347.09	-20,348.64
MA(2)	-7,580.71	-20,351.08	-20,349.53	-20,352.08	-20,345.31	
MA(3)	-10,505.64	-20,349.82	-20,348.19	-20,352.22		
MA(4)	-12,612.04	-20,348.69	-20,348.28			
MA(5)	-14,123.45	-20,349.03				

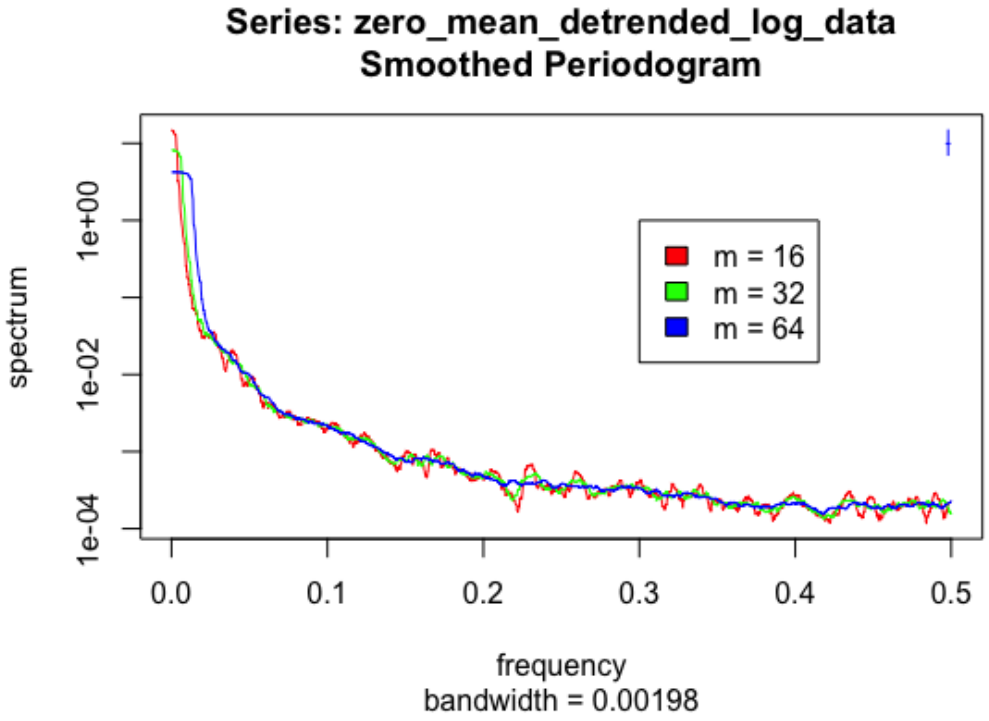
Table 1: ARMA Model AIC Data

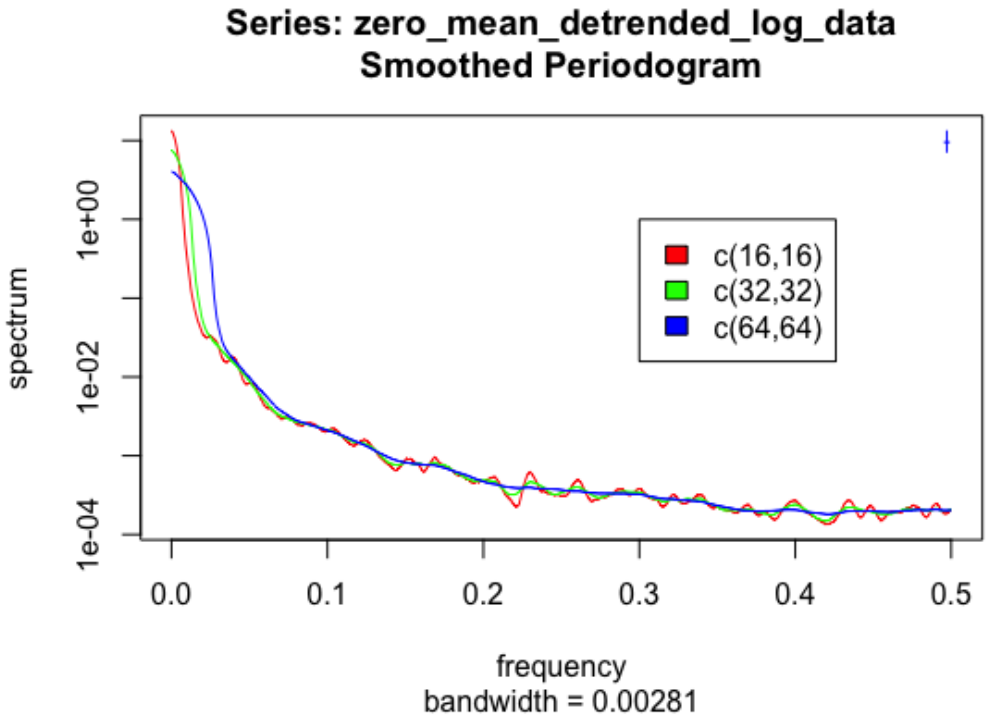
5.3 Dow Jones Index Analysis Periodogram Smoothening

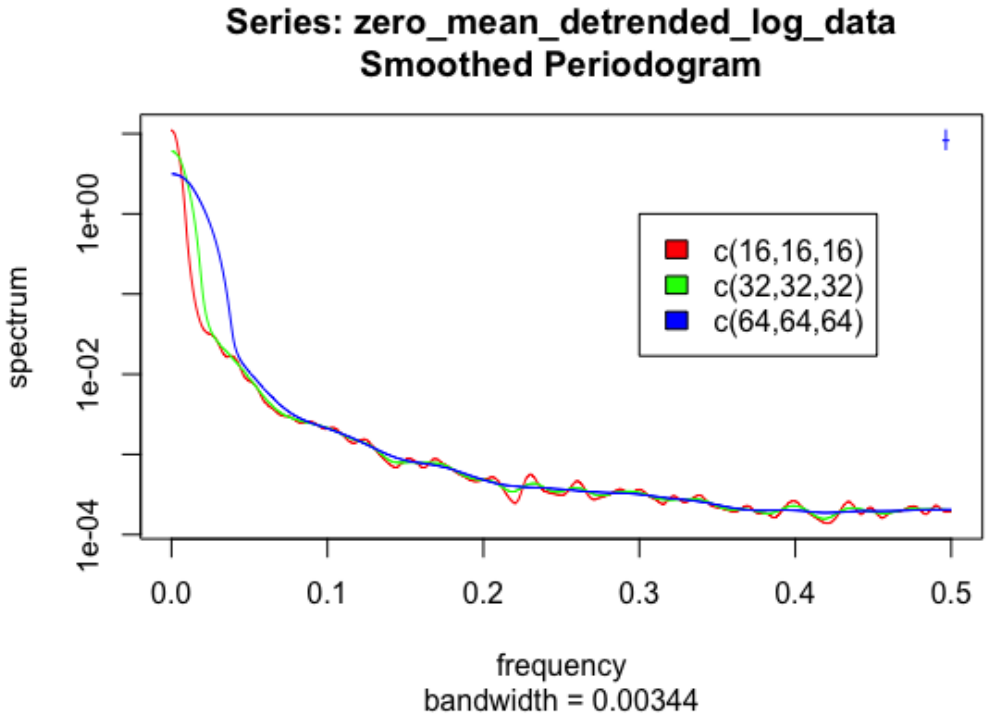



Series: zero_mean_detrended_log_data
Smoothed Periodogram
spectrum
1e+00
1e-02
1e-04
0.0
0.1
0.2
0.3
0.4
0.5
m=32
c(32,32)
c(32,32,32)
frequency
bandwidth = 0.00391


Series: zero_mean_detrended_log_data
Smoothed Periodogram
spectrum
5e-01
1e-02
5e-04
0.0
0.1
0.2
0.3
0.4
0.5
m=64
c(64,64)
c(64,64,64)
frequency
bandwidth = 0.00776


Series: zero_mean_detrended_log_data
Smoothed Periodogram
spectrum
1e+00
1e-02
1e-04
m = 16
m = 32
m = 64
0.0
0.1
0.2
0.3
0.4
0.5
frequency
bandwidth = 0.00198


Series: zero_mean_detrended_log_data
Smoothed Periodogram
spectrum
1e+00
1e-02
1e-04
c(16,16)
c(32,32)
c(64,64)
0.0
0.1
0.2
0.3
0.4
0.5
frequency
bandwidth = 0.00281

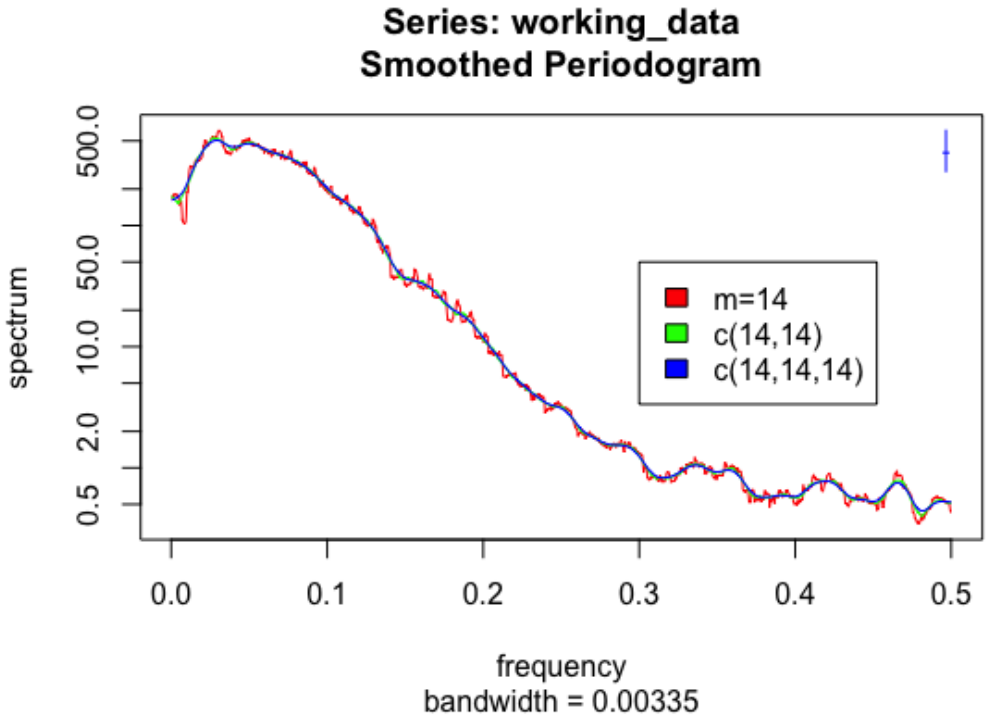


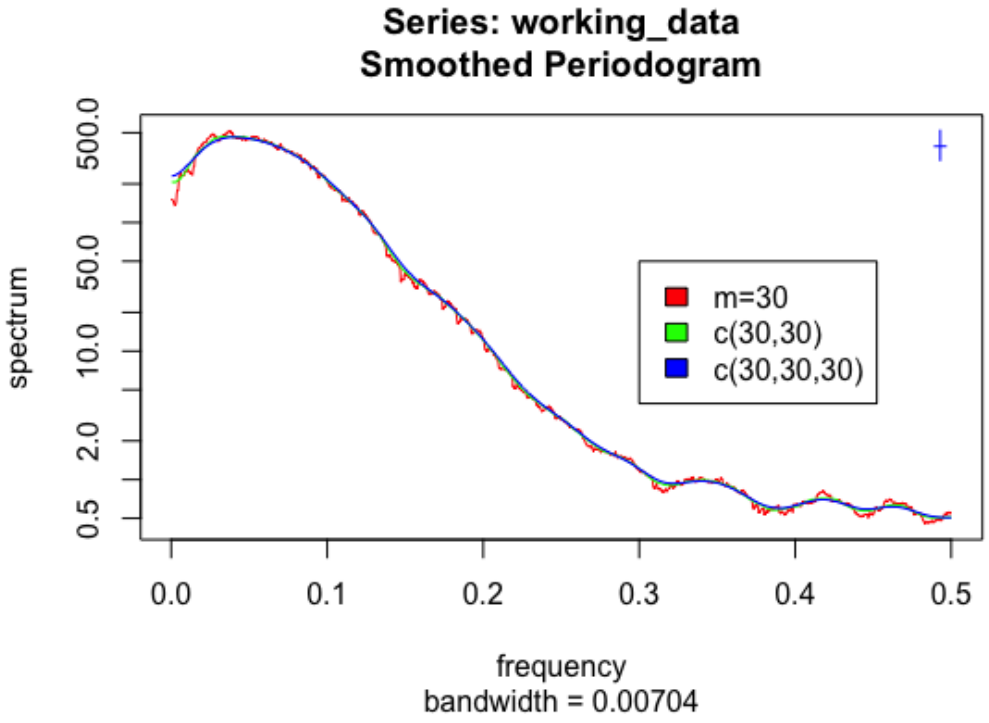
5.4 ECG Analysis ARMA Model Data

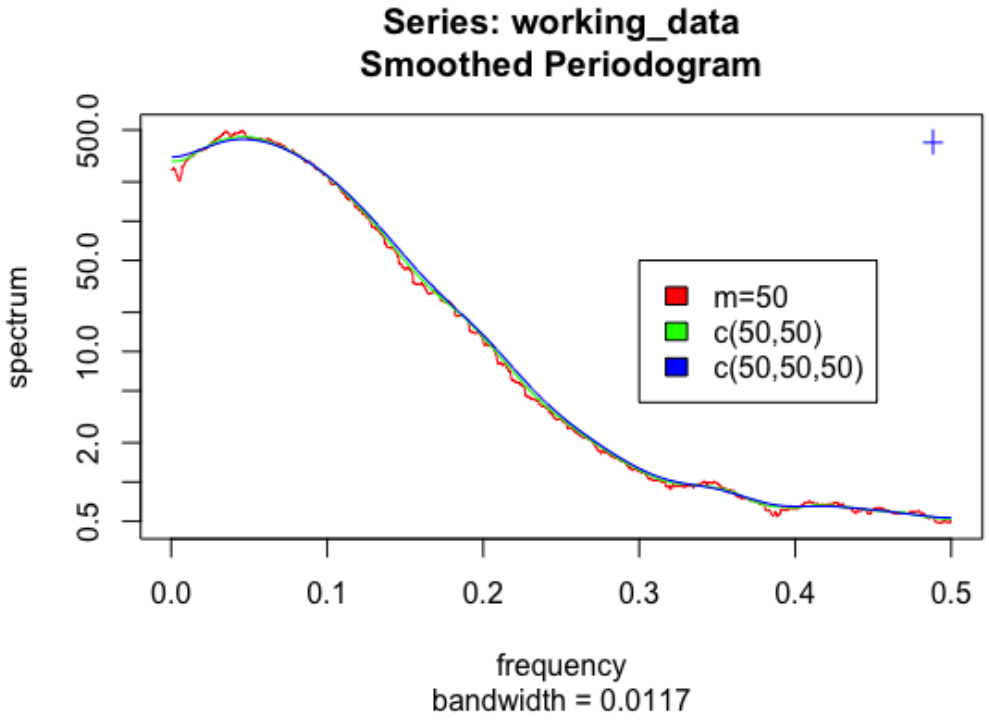
AIC	AR(0)	AR(1)	AR(2)	AR(3)	AR(4)	AR(5)
MA(0)	18,301.59	14,393.77	12,341.56	12,204.47	12,202.17	12,182.65
MA(1)	15,494.96	12,965.60	12,231.07	12,204.12	12,197.29	12,182.66
MA(2)	13,739.93	12,430.98	12,185.09	12,186.42	12,195.96	12,194.08
MA(3)	12,796.37	12,256.24	12,185.75	12,186.29	12,188.26	12,185.96
MA(4)	12,391.98	12,219.34	12,182.59	12,188.28	12,190.2	12,171.90
MA(5)	12,237.97	12,208.12	12,172.15	12,186.59	**12,171.66**	12,173.56

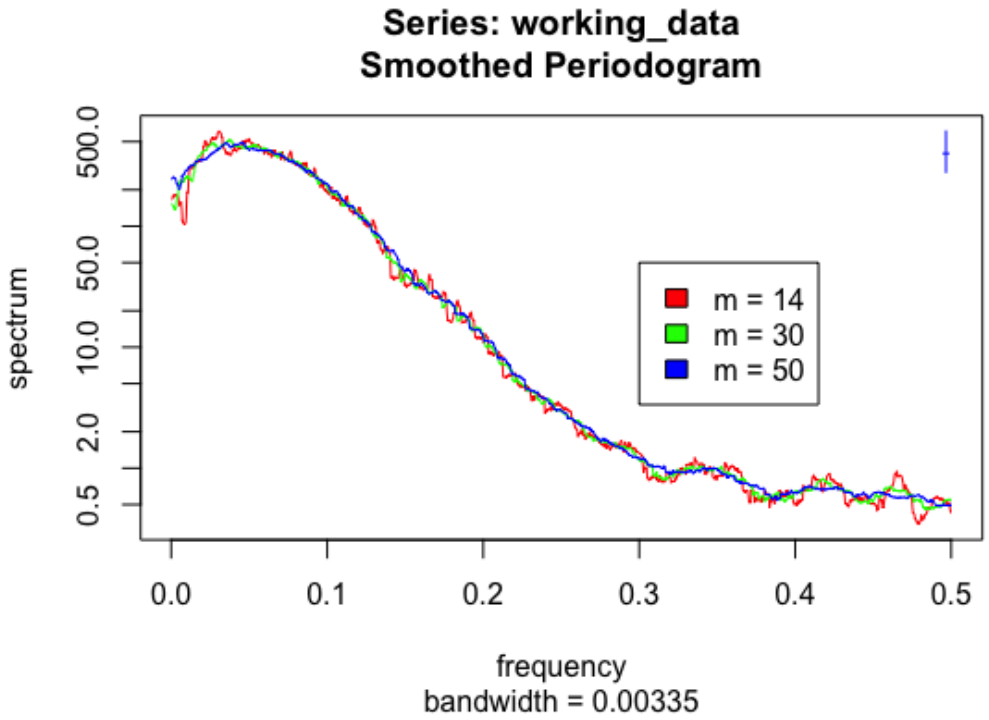
Table 2: ARMA Model AIC Data

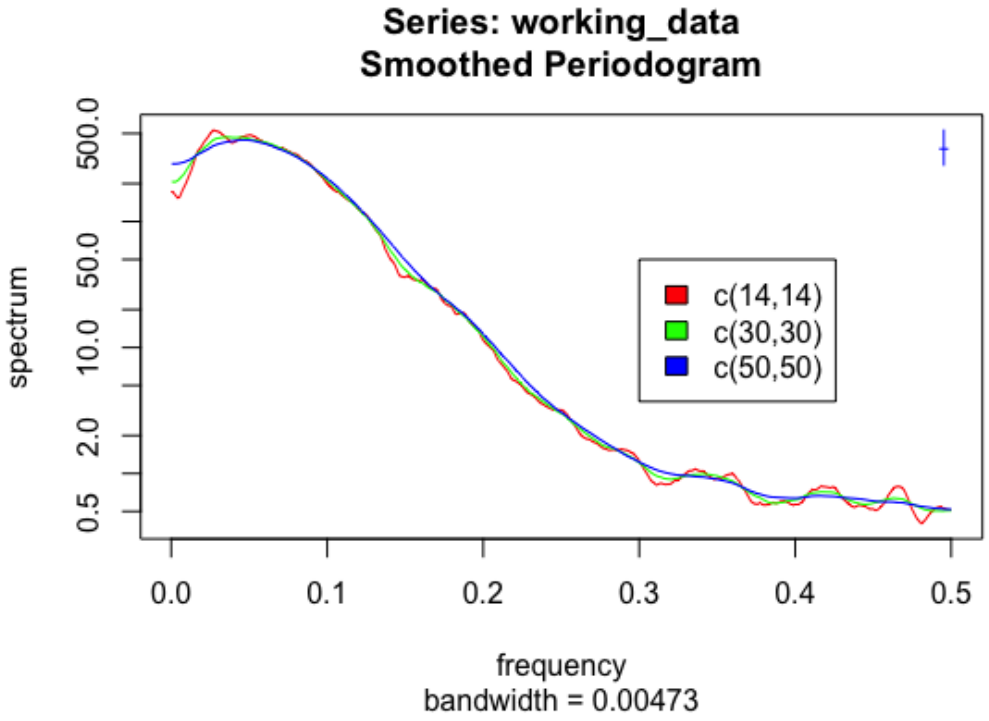
5.5 ECG Analysis Periodogram Smoothening

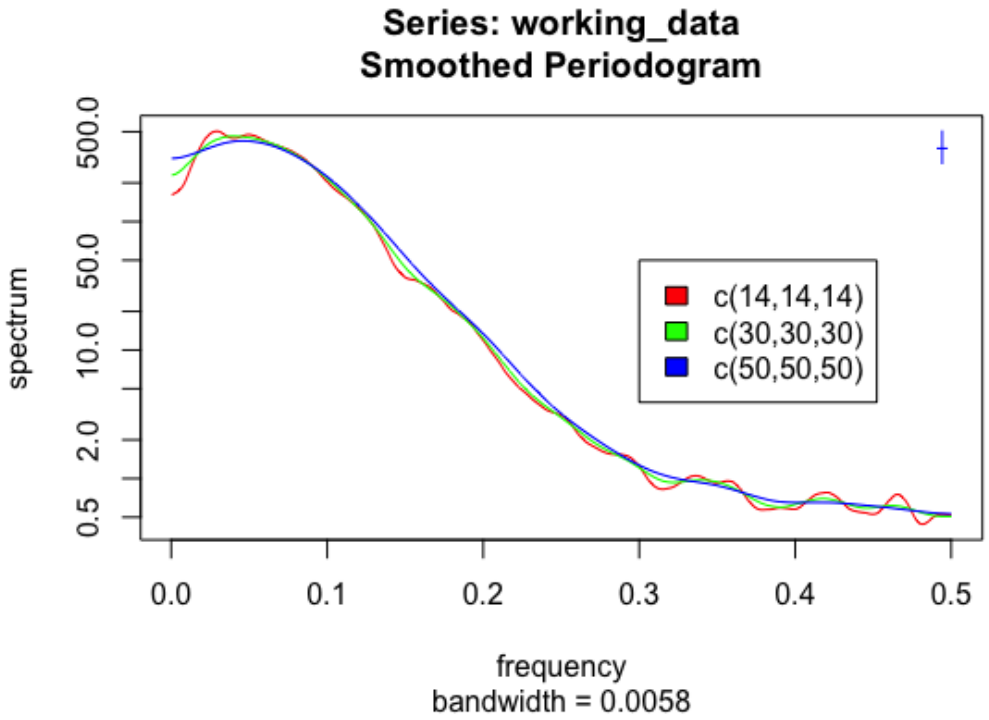





Series: working_data
Smoothed Periodogram
spectrum
500.0
50.0
10.0
2.0
0.5
0.0
0.1
0.2
0.3
0.4
0.5
m=50
c(50,50)
c(50,50,50)
frequency
bandwidth = 0.0117


Series: working_data
Smoothed Periodogram
spectrum
500.0
50.0
10.0
2.0
0.5
0.0
0.1
0.2
0.3
0.4
0.5
m = 14
m = 30
m = 50
frequency
bandwidth = 0.00335


Series: working_data
Smoothed Periodogram
spectrum
500.0
50.0
10.0
2.0
0.5
0.0
0.1
0.2
0.3
0.4
0.5
c(14,14)
c(30,30)
c(50,50)
frequency
bandwidth = 0.00473


Series: working_data
Smoothed Periodogram
spectrum
500.0
50.0
10.0
2.0
0.5
0.0
0.1
0.2
0.3
0.4
0.5
c(14,14,14)
c(30,30,30)
c(50,50,50)
frequency
bandwidth = 0.0058

5.6 Exercise 1 Code

```
# Libraries
library(ggplot2)
library(dplyr)
library(pracma)

# load AirPassengers data from Rdatasets
data("AirPassengers")

# converting time series to vector for easy manipulation
AirPassengers_data = as.numeric(AirPassengers)

# plotting data for preliminary observation
plot(AirPassengers_data, type = 'l')

# scaling original data using log
log_AirPassengers = log(AirPassengers_data)
plot(log_AirPassengers, type='l')


# estimating the additive model
## eliminating trend
detrended_log_AirPassengers = detrend(log_AirPassengers)
plot(detrended_log_AirPassengers, type='l')

## finding and eliminating non-zero mean
m_tilde = mean(detrended_log_AirPassengers)
zero_mean_detrended_log_AirPassengers = detrended_log_AirPassengers - m_tilde
plot(zero_mean_detrended_log_AirPassengers, type='l')

## estimating seasonality - we know seasonality is annual, and data is for 12 years
## number of seasons M = 12
## number of periods P = 12
## therefore,can be verified as, total number of datapoints N = M*P = 144
M = 12; P = 12;

s0 = numeric(P)

for (i in 1:M)
{
  start_index = (((i-1)*P)+1)
  end_index = i*P
  s0 = s0 + (zero_mean_detrended_log_AirPassengers[start_index:end_index])/M
}

s_unit = s0 - mean(s0)
plot(s_unit, type='l')

s = rep(s_unit, times = M)
plot(s, type='l')
```

```
w = (detrend(log_AirPassengers-s))
w = w - mean(w);
plot(w, type='l')

m = log_AirPassengers - s - w
plot(m, type='l')
```

5.7 Exercise 2 Codes

5.7.1 Dow Jones Index Analysis Code

```
# Libraries
library(ggplot2)
library(dplyr)
library(pracma)
library(readr)
library(psd)

# load Dow Jones Index data from file in working directory of Rscript
data = read_tsv("dow_jones.txt", col_names = FALSE)
data = data[["X2"]]

# plotting data for preliminary observation
plot(data, type = 'l')

# scaling original data using log
log_data = log(data)
plot(log_data, type='l')


# eliminating trend
detrended_log_data = detrend(log_data)
plot(detrended_log_data, type='l')

# finding and eliminating non-zero mean
m_tilde = mean(detrended_log_data)
zero_mean_detrended_log_data = detrended_log_data - m_tilde
plot(zero_mean_detrended_log_data, type='l')

# renaming dataset
working_data = zero_mean_detrended_log_data

# computing and plotting correlations
acf_data = acf(working_data)
pacf_data = pacf(working_data)
```

```
# Estimating ARMA Model
ARMA_Model = arima(working_data, c(1,0,0))
print(ARMA_Model)

## Residual Diagnostics
ARMA_Model_Residual = residuals(ARMA_Model)
plot(ARMA_Model_Residual)
plot(ARMA_Model_Residual[1:500], type = 'l')
plot(ARMA_Model_Residual[2500:3000], type = 'l')
acf(ARMA_Model_Residual)

# computing spectral density function (AKA periodogram)
## raw periodogram
periodogram_data = spec.pgram(working_data, taper = 0, plot = FALSE)
plot(periodogram_data)

## raw periodogram with log
raw.spec <- spec.pgram(working_data, taper = 0, plot = FALSE)
plot(raw.spec, log = "no")

## smoothing the periodogram by taking moving averages using kernel functions

### N = 4687, M = cbrt(N) = nearly 17 <<< N , we take powers of 2 i.e 16
periodogram_data_16_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 16))
periodogram_data_16_02 = spec.pgram(working_data, kernel = kernel("daniell", c(16,16)))
periodogram_data_16_03 = spec.pgram(working_data, kernel = kernel("daniell", c(16,16,16)))

plot(periodogram_data_16_01,type="l",col="red")
lines(periodogram_data_16_02,type="l",col="green")
lines(periodogram_data_16_03,type="l",col="blue")

legend(0.3, 1,
       legend=c("m=16", "c(16,16)", "c(16,16,16)"),
       fill = c("red","green", "blue"))

### N = 4687, M = 32 << N , we take an intermediate power of 2 i.e 32
periodogram_data_32_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 32))
periodogram_data_32_02 = spec.pgram(working_data, kernel = kernel("daniell", c(32,32)))
periodogram_data_32_03 = spec.pgram(working_data, kernel = kernel("daniell", c(32,32,32)))

plot(periodogram_data_32_01,type="l",col="red")
lines(periodogram_data_32_02,type="l",col="green")
lines(periodogram_data_32_03,type="l",col="blue")

legend(0.3, 1,
       legend=c("m=32", "c(32,32)", "c(32,32,32)"),
       fill = c("red","green", "blue"))
```

```
### N = 4687, M = sqrt(N) = nearly 69 << N , we take powers of 2 i.e 64
periodogram_data_64_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 64))
periodogram_data_64_02 = spec.pgram(working_data, kernel = kernel("daniell", c(64,64)))
periodogram_data_64_03 = spec.pgram(working_data, kernel = kernel("daniell", c(64,64,64)))

plot(periodogram_data_64_01,type="l",col="red")
lines(periodogram_data_64_02,type="l",col="green")
lines(periodogram_data_64_03,type="l",col="blue")

legend(0.3, 1,
       legend=c("m=64", "c(64,64)", "c(64,64,64)"),
       fill = c("red","green", "blue"))

## plotting across window sizes
### m = 16, 32, 64
plot(periodogram_data_16_01,type="l",col="red")
lines(periodogram_data_32_01,type="l",col="green")
lines(periodogram_data_64_01,type="l",col="blue")

legend(0.3, 1,
       legend=c("m = 16", "m = 32", "m = 64"),
       fill = c("red","green", "blue"))

### (c,c) = 16, 32, 64
plot(periodogram_data_16_02,type="l",col="red")
lines(periodogram_data_32_02,type="l",col="green")
lines(periodogram_data_64_02,type="l",col="blue")

legend(0.3, 1,
       legend=c("c(16,16)", "c(32,32)", "c(64,64)"),
       fill = c("red","green", "blue"))

### (c,c,c) = 16, 32, 64
plot(periodogram_data_16_03,type="l",col="red")
lines(periodogram_data_32_03,type="l",col="green")
lines(periodogram_data_64_03,type="l",col="blue")

legend(0.3, 1,
       legend=c("c(16,16,16)", "c(32,32,32)", "c(64,64,64)"),
       fill = c("red","green", "blue"))

# spectrum comparison
plot(periodogram_data)
lines(periodogram_data_64_03,type="l",col="red")
lines(periodogram_data_32_03,type="l",col="deepskyblue")
lines(periodogram_data_16_03,type="l",col="green")
legend(0.25, 50,
       legend=c("raw_periodogram", "c(64,64,64)", "c(32,32,32)", "c(16,16,16)"),
       fill = c("black","green", "red", "blue"))
```

5.7.2 ECG Analysis Code

```
# Libraries
library(ggplot2)
library(dplyr)
library(pracma)
library(readr)
library(psd)

# load Dow Jones Index data from file in working directory of Rscript
data = read_tsv("ecg.txt", col_names = TRUE)
data = data[["x1"]]

# plotting data for preliminary observation
plot(data, type = 'l')

# eliminating trend
detrended_data = detrend(data)
plot(detrended_data, type='l')

# finding and eliminating non-zero mean
m_tilde = mean(detrended_data)
zero_mean_detrended_data = detrended_data - m_tilde
plot(zero_mean_detrended_data, type='l')

# renaming dataset
working_data = zero_mean_detrended_data
plot(working_data, type = 'l')

# computing and plotting correlations
acf_data = acf(working_data, 50)
pacf_data = pacf(working_data, 20)

# Estimating ARMA Model
ARMA_Model = arima(working_data, c(4,0,5))
print(ARMA_Model)

## Residual Diagnostics
ARMA_Model_Residual = residuals(ARMA_Model)
plot(ARMA_Model_Residual)
acf(ARMA_Model_Residual)

# computing spectral density function (AKA periodogram)
## raw periodogram
periodogram_data = spec.pgram(working_data, taper = 0, plot = FALSE)
plot(periodogram_data)

## raw periodogram with log
raw.spec <- spec.pgram(working_data, taper = 0, plot = FALSE)
plot(raw.spec, log = "no")
```

```
## smoothing the periodogram by taking moving averages using kernel functions

### N = 2499, M = cbrt(N) = nearly 14 <<< N
periodogram_data_14_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 14))
periodogram_data_14_02 = spec.pgram(working_data, kernel = kernel("daniell", c(14,14)))
periodogram_data_14_03 = spec.pgram(working_data, kernel = kernel("daniell", c(14,14,14)))

plot(periodogram_data_14_01,type="l",col="red")
lines(periodogram_data_14_02,type="l",col="green")
lines(periodogram_data_14_03,type="l",col="blue")

legend(0.3, 50,
       legend=c("m=14", "c(14,14)", "c(14,14,14)"),
       fill = c("red","green", "blue"))

### N = 2499, M = 30 << N, an intermediate value
periodogram_data_30_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 30))
periodogram_data_30_02 = spec.pgram(working_data, kernel = kernel("daniell", c(30,30)))
periodogram_data_30_03 = spec.pgram(working_data, kernel = kernel("daniell", c(30,30,30)))

plot(periodogram_data_30_01,type="l",col="red")
lines(periodogram_data_30_02,type="l",col="green")
lines(periodogram_data_30_03,type="l",col="blue")

legend(0.3, 50,
       legend=c("m=30", "c(30,30)", "c(30,30,30)"),
       fill = c("red","green", "blue"))

### N = 2499, M = sqrt(N) = nearly 50 << N
periodogram_data_50_01 = spec.pgram(working_data, kernel = kernel("daniell", m = 50))
periodogram_data_50_02 = spec.pgram(working_data, kernel = kernel("daniell", c(50,50)))
periodogram_data_50_03 = spec.pgram(working_data, kernel = kernel("daniell", c(50,50,50)))

plot(periodogram_data_50_01,type="l",col="red")
lines(periodogram_data_50_02,type="l",col="green")
lines(periodogram_data_50_03,type="l",col="blue")

legend(0.3, 50,
       legend=c("m=50", "c(50,50)", "c(50,50,50)"),
       fill = c("red","green", "blue"))


## plotting across window sizes
### m = 14, 30, 50
plot(periodogram_data_14_01,type="l",col="red")
lines(periodogram_data_30_01,type="l",col="green")
lines(periodogram_data_50_01,type="l",col="blue")

legend(0.3, 50,
       legend=c("m = 14", "m = 30", "m = 50"),
       fill = c("red","green", "blue"))
```

```
### (c,c) = 16, 32, 64
plot(periodogram_data_14_02,type="l",col="red")
lines(periodogram_data_30_02,type="l",col="green")
lines(periodogram_data_50_02,type="l",col="blue")

legend(0.3, 50,
       legend=c("c(14,14)", "c(30,30)", "c(50,50)"),
       fill = c("red","green", "blue"))

### (c,c,c) = 16, 32, 64
plot(periodogram_data_14_03,type="l",col="red")
lines(periodogram_data_30_03,type="l",col="green")
lines(periodogram_data_50_03,type="l",col="blue")

legend(0.3, 50,
       legend=c("c(14,14,14)", "c(30,30,30)", "c(50,50,50)"),
       fill = c("red","green", "blue"))

# spectrum comparison
plot(periodogram_data)
lines(periodogram_data_14_03,type="l",col="green")
lines(periodogram_data_30_03,type="l",col="red")
lines(periodogram_data_50_03,type="l",col="blue")
legend(0.0, 1,
       legend=c("raw_periodogram", "c(64,64,64)", "c(32,32,32)", "c(16,16,16)"),
       fill = c("black","green", "red", "blue"))
```

References

[1] M. Peixeiro, The Complete Guide to Time Series Analysis and Forecasting, https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775, accessed: 10 July 2022.

[2] T. Srivastava, A Complete Tutorial on Time Series Modeling in R, https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/, accessed: 10 July 2022.

[3] R. G. Mohammad Mirzavand, A Stochastic Modelling Technique for Groundwater Level Forecasting in an Arid Environment Using Time Series Methods, https://www.researchgate.net/publication/268153169_A_Stochastic_Modelling_Technique_for_Groundwater_Level_Forecasting_in_an_Arid_Environment_Using_Time_Series_Methods, accessed: 10 July 2022.

[4] R. Box, Jenkins, *Time Series Analysis - Forcasting and Control*, 5th ed., John Wiley & Sons, **2016**.

[5] P. T. J.D Long, *R Cookbook*, 2nd ed., O'Reilly Media, Inc., **2019**.

[6] RStudio, Spectral Analysis of Time Series, https://rpubs.com/nnnagle/TimeSeries2, accessed: 10 July 2022.

[7] K. Hyndman, *Journal of Statistical Software* **2008**, *27*, 1–22.

[8] ARIMA modelling in R, https://otexts.com/fpp2/arima-r.html, accessed: 10 July 2022.

[9] H. K. Gjerrit Meinsma, Annika Betken, *Time Series Analysis*, **2022**.