# lab4_carlos

## Data Loading

Here we load the data and construct the time series object.

```
# Load libraries
library(xts)
```

```
## Warning: package 'xts' was built under R version 3.4.3
## Loading required package: zoo
## Warning: package 'zoo' was built under R version 3.4.1
##
## Attaching package: 'zoo'
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(tseries)
```

```
## Warning: package 'tseries' was built under R version 3.4.3
```

```
library(forecast)
```

```
## Warning: package 'forecast' was built under R version 3.4.3
```

```
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
## The following object is masked from 'package:forecast':
##
##     autolayer
```

```
library(reshape2)

# Load data
data = read.csv('Lab4-series2.csv')

# Visualize a few records
head(data)
```

```
##   X     x
## 1 1 5.544
## 2 2 5.555
## 3 3 5.172
## 4 4 4.878
## 5 5 4.851
## 6 6 4.686
```

The data has the lag number and the numeric value. Let's build the time series starting on 1/1/1990.

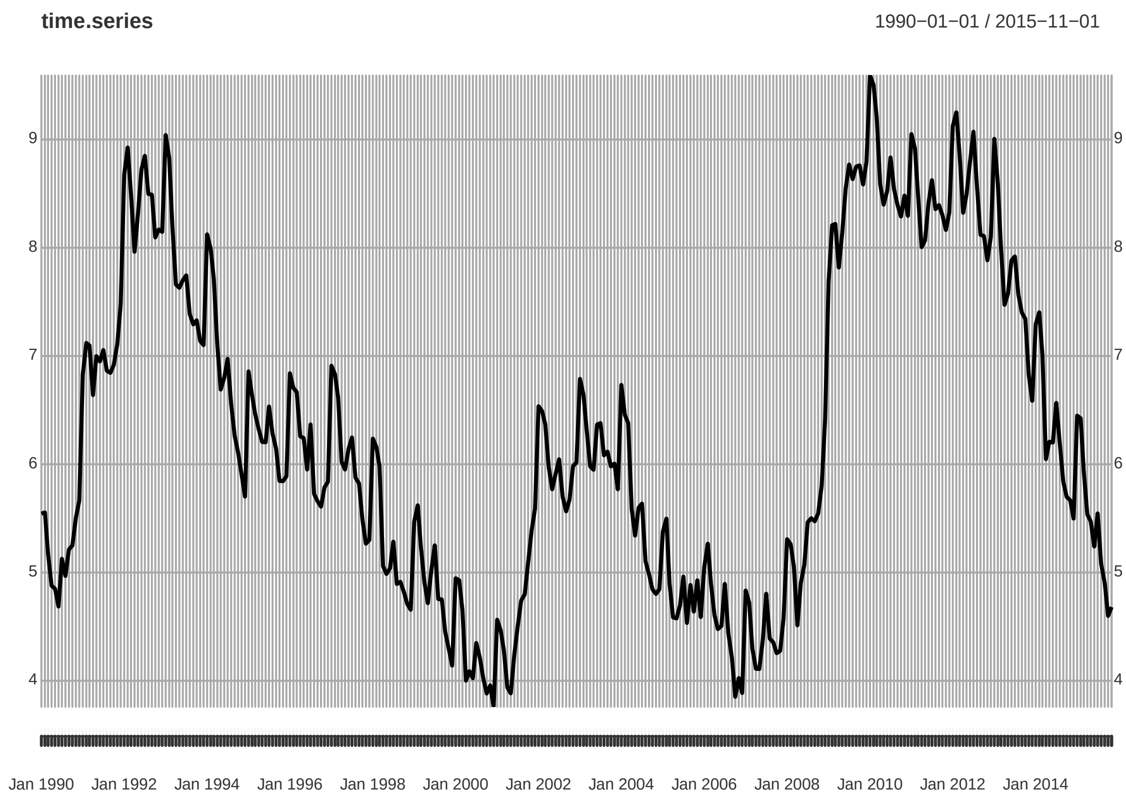
```r
sequence <- seq.Date(from=as.Date("1990-1-1"), to=as.Date("2015-11-1"), by="month")
time.series <- xts(data$x, order.by = sequence)
colnames(time.series) = 'x'
```

# Eda

## Series plot

Let's first visualize the raw series:

```r
plot(time.series)
```

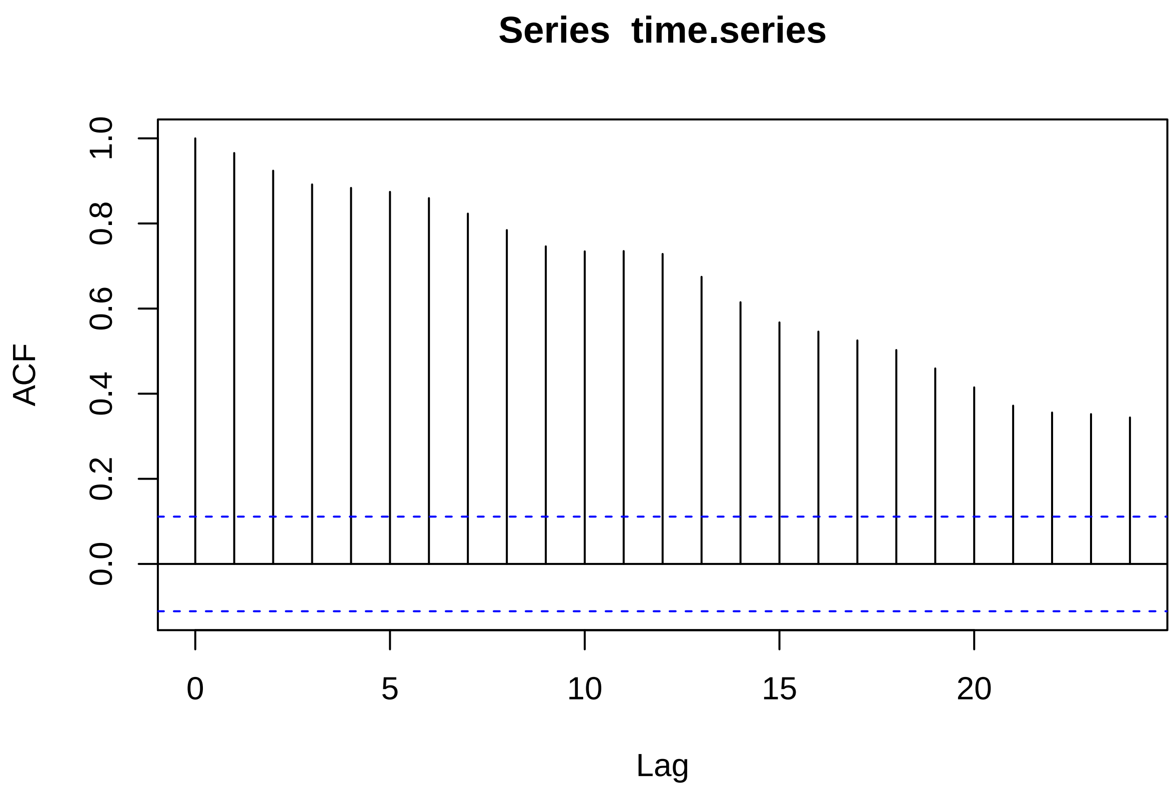



We observe strong seasonal trends with peaks on the same quarter of each year, plus general trends ranging from 5 to 10 years.

Let's now observe the ACF and PACF to understand more about the series.

# ACF

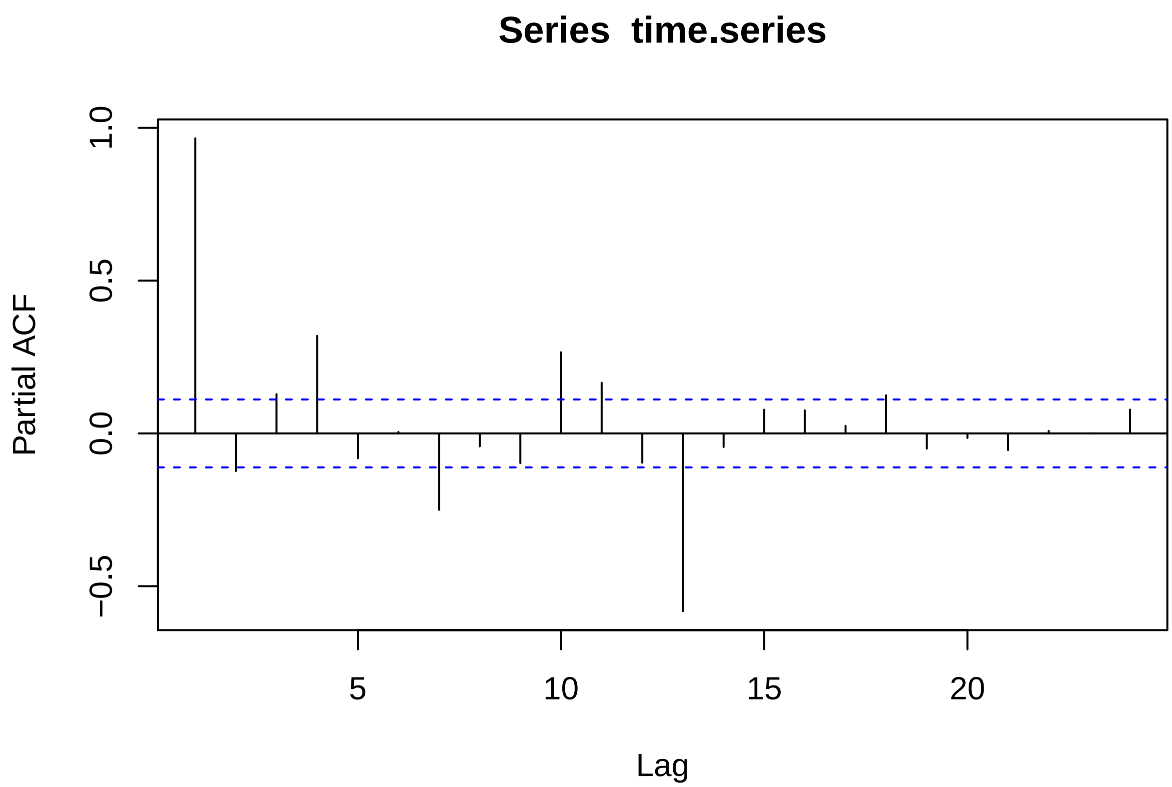
```r
acf(time.series)
```




The gradual decline at a reasonably steady pace suggests an AR model with $p \geq 1$.

## PACF

```r
pacf(time.series)
```




In general, straightforward seasonal trends show a PACF with only a significant trend at lag $s$ where $s$ is the length of the season in lags. However, here the length of the season is $s = 12$, but the most significant PACF occurrences are observed at lags 1 and 13. This is likely a result from the longer term repeated up and down trends that we observe beyond the yearly seasonality.

## Dicky Fuller Test

We now run a Dicky Fuller test on our series.

```
adf.test(time.series$x)
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  time.series$x
## Dickey-Fuller = -1.9739, Lag order = 6, p-value = 0.5874
## alternative hypothesis: stationary
```

With a p-value $p > 0.05$ we fail to reject the null hypothesis that the data is non-stationary.

## Differencing

We now consider differencing, given the non-stationarity we observe in the series.

```r
#Try Seasonal Differencing Here
d.time.series = diff(time.series$x, lag = 12)[13:311]
plot(d.time.series)
```


```r
adf.test(d.time.series$x)
```

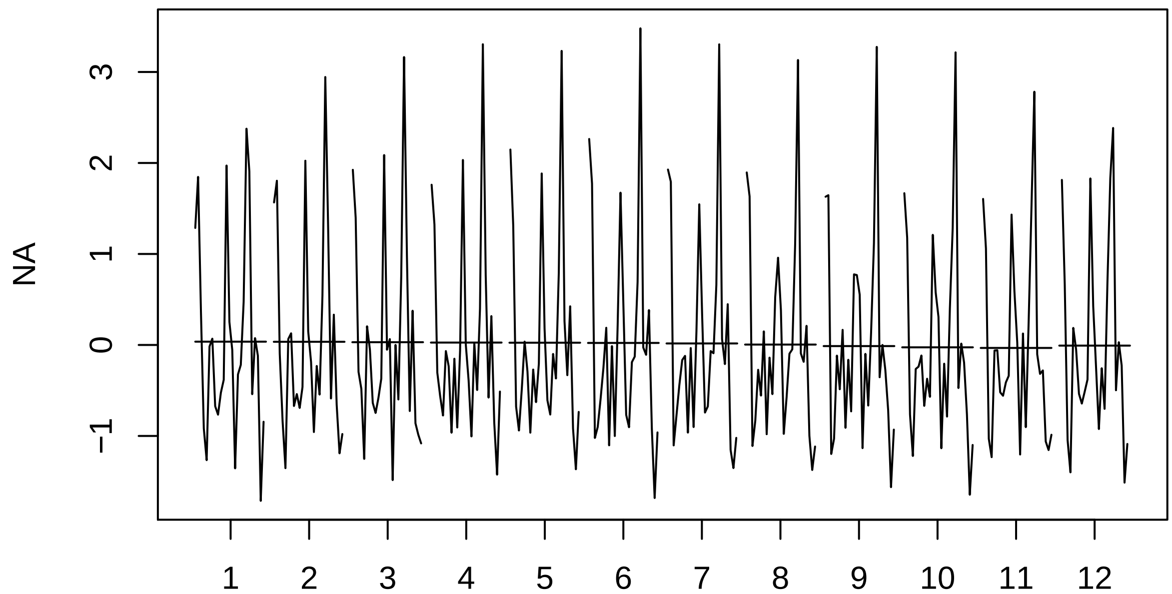
```
## Warning in adf.test(d.time.series$x): p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  d.time.series$x
## Dickey-Fuller = -4.0272, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

We can see that once differenced, the Dickey-Fuller test rejects the null hypothesis that the series is non-stationary, suggesting $d = 1$.

Let's now analyze the monthly means for our differenced series.

```r
monthplot(d.time.series)
```




The montly means are quite flat, suggesting $D = 1$;

## Models

Now we work towards selecting a model and forecasting with it.

### Methodology

Given that we've chosen values for $D = 1$ and $d = 1$ in our EDA, we now proceed to build models with varying parameters for $p$ and $q$. We want to be able to forecast and avoid overfitting, so we use AIC as primary method to select a model, since we want to favor parsimonious models. Given similar AIC's, we'll also consider RMSE or MAPE. To test our forecasting capacity, we will divide the series into a train and test set.

### Train and Test Set Partition

```
# Train data will be from 1990 to 2014
train.time.series = time.series["1990/2014"]
colnames(train.time.series) = 'x'

# We'll forecast the time series during 2015 to test our models
```

```
test.time.series = time.series["2015"]
colnames(test.time.series) = 'x'
```

### Parameter Selection

We now iterate through parameter combinations and estimate and assess different models with those parameters.

```
results = data.frame(p=NA, q=NA, P=NA, Q=NA, AIC=NA, RMSE=NA, MAPE=NA)

for (P in 0:2) {
  for (Q in 0:2) {
    for (p in 0:4) {
      for (q in 0:4) {

        tryCatch({
          model <- Arima(train.time.series$x, order = c(p, 1, q), seasonal = list(order = c(P,1,Q), per
          AIC <- model$aic
          model.summary <- as.data.frame(summary(model))
          RMSE <- model.summary$RMSE
          MAPE <- model.summary$MAPE

          result <- data.frame(p=p, q=q, P=P, Q=Q, AIC=AIC, RMSE=RMSE, MAPE=MAPE)
          results <- rbind(results, result)
        }, warning = function(w){
          AIC <- NA
          RMSE <- NA
          MAPE <- NA
          }, error = function(e){
          AIC <- NA
          RMSE <- NA
          MAPE <- NA
        })
      }
    }
  }
}
results <- results[2:nrow(results),]
```

Let's now analyze which models are best if we choose the ones with the lowest AIC, or lowest MAPe or lowest RMSE:

```
results[results$AIC==min(results$AIC, na.rm=T),]
```

```
##     p q P Q       AIC      RMSE    MAPE
## 201 2 1 2 2 -129.3362 0.1753747 2.28454
```

```
results[results$RMSE==min(results$RMSE, na.rm=T),]
```

```
##     p q P Q      AIC      RMSE     MAPE
## 207 3 4 2 2 -124.559 0.1738167 2.259259
```

```
results[results$MAPE==min(results$MAPE, na.rm=T),]
```

```
##     p q P Q      AIC      RMSE     MAPE
## 207 3 4 2 2 -124.559 0.1738167 2.259259
```

Naturally, the model with the lowest AIC has lower order values for the parameters, which potentially might translate into less overfitting. Also it is worth noting that for the lowest AIC, the values for the RMSE and MAPE are less than 5% higher than the lowest MAPE and RMSE, meaning that the lowest AIC model is more parsimonious, and at the same time almost the same in terms of prediction errors.

Thus, the chosen model has the parameters $p = 2$, $q = 1$, $P = 0$, $Q = 1$, $D = 1$ and $d = 1$.

## Final Model

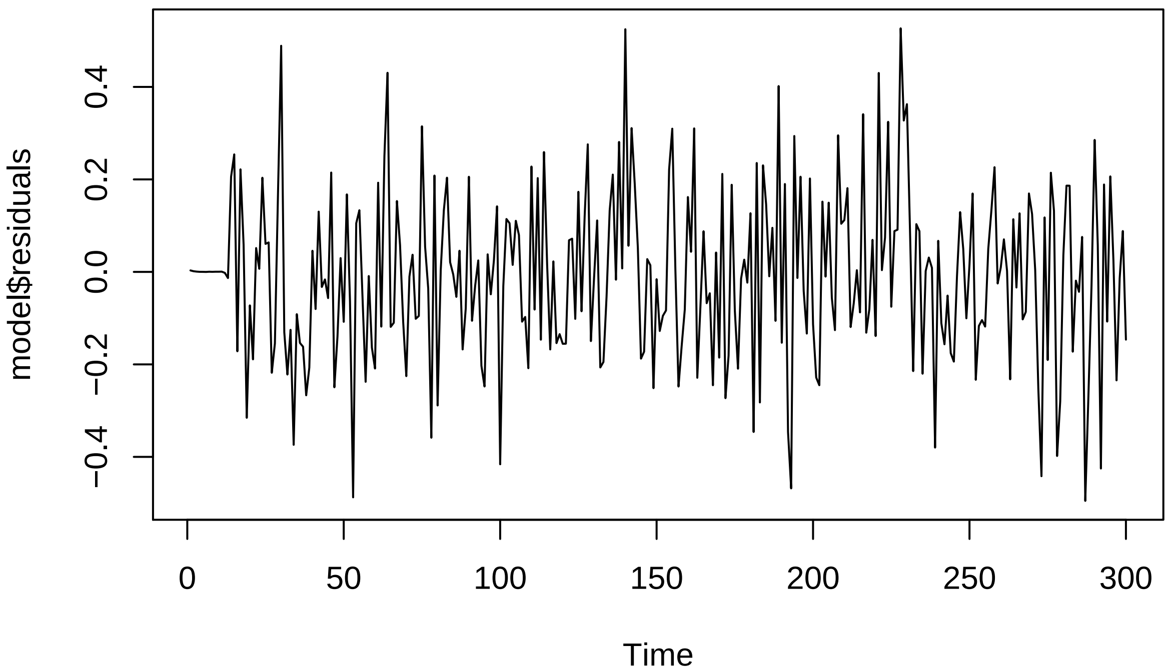
```
model <- Arima(train.time.series$x, order = c(2, 1, 1), seasonal = list(order = c(0, 1,1), period = 12)
summary(model)
```

```
## Series: train.time.series$x
## ARIMA(2,1,1)(0,1,1)[12]
##
## Coefficients:
##          ar1     ar2      ma1     sma1
##       0.7292  0.1591  -0.7039  -0.8825
## s.e.  0.1030  0.0680   0.0900   0.0512
##
## sigma^2 estimated as 0.03476:  log likelihood=67.93
## AIC=-125.86   AICc=-125.64   BIC=-107.56
##
## Training set error measures:
##                        ME      RMSE       MAE        MPE     MAPE     MASE
## Training set -0.008922925 0.1810804 0.1416213 -0.1016704 2.364658 0.503098
##                     ACF1
## Training set -0.00410307
```

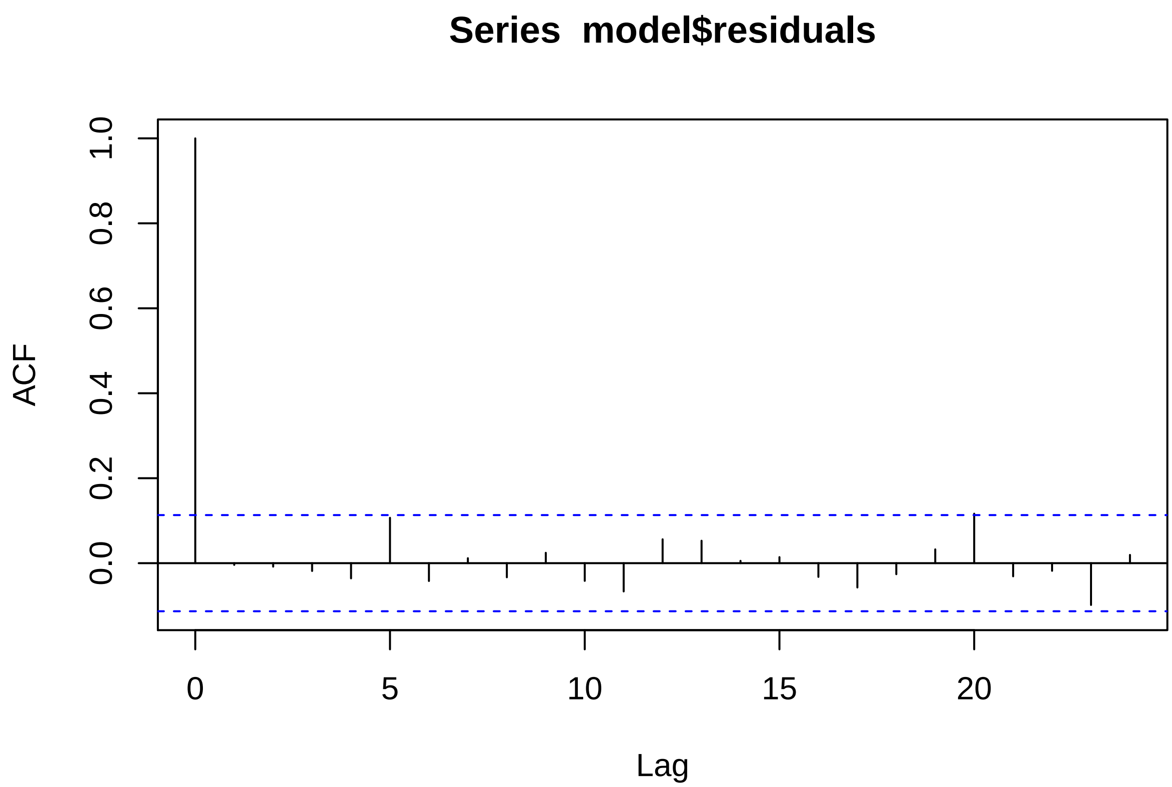
## Error Analysis

Now we analyze the residuals, including their to verify stationarity of residuals.

```
plot(model$residuals)
```




```r
acf(model$residuals)
```




```
adf.test(model$residuals)
```

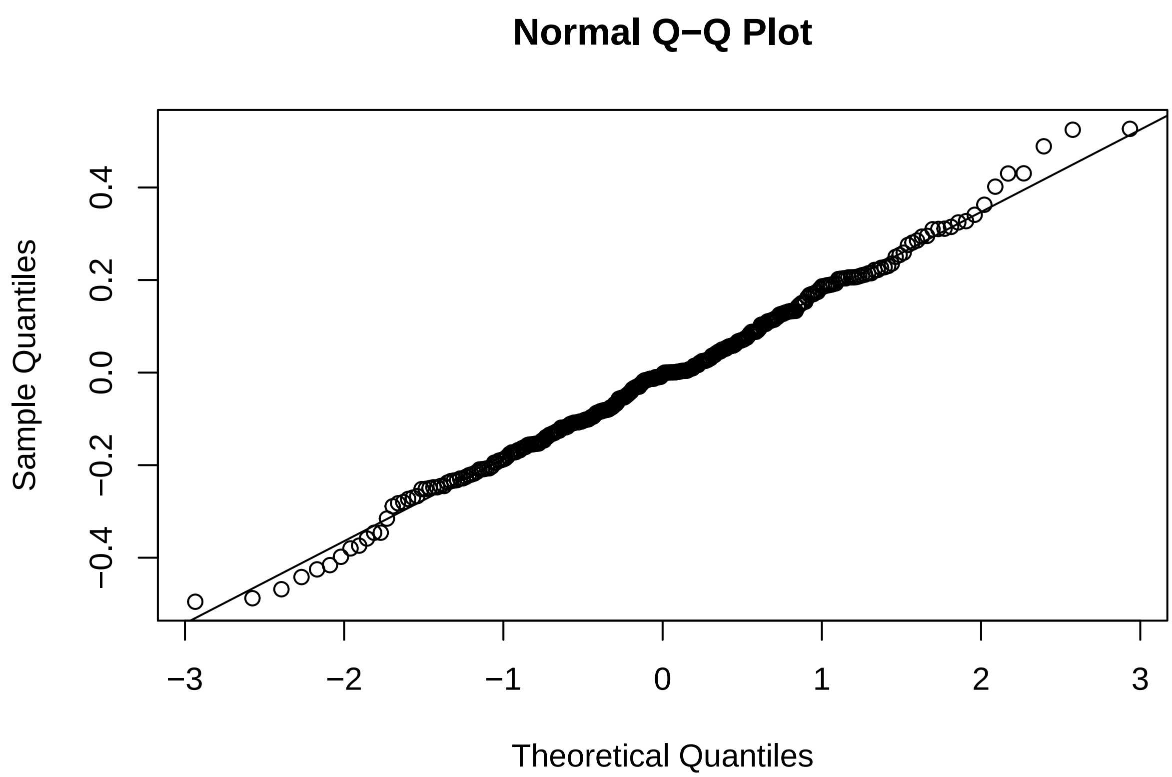
```
## Warning in adf.test(model$residuals): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  model$residuals
## Dickey-Fuller = -6.2383, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

A visual inspection of the residual series and its ACF suggests stationary residuals. Moreover, when performing Dickey-Fuller test we reject the null hypothesis that the residuals are non-stationary.

Finally, we visually assess normality of the residuals using a qqplot:

```
qqnorm(model$residuals)
qqline(model$residuals)
```




And indeed, a visual inspection suggests normality, supporting the other results obtained.

### Looking at in-sample fit and forecasting

We will plot the actual vs. fitted values from our model based on the training set to determine in-sample fit

```
model <- Arima(train.time.series$x, order = c(2,1,1), seasonal = list(order = c(2, 1, 2), period = 12),
```

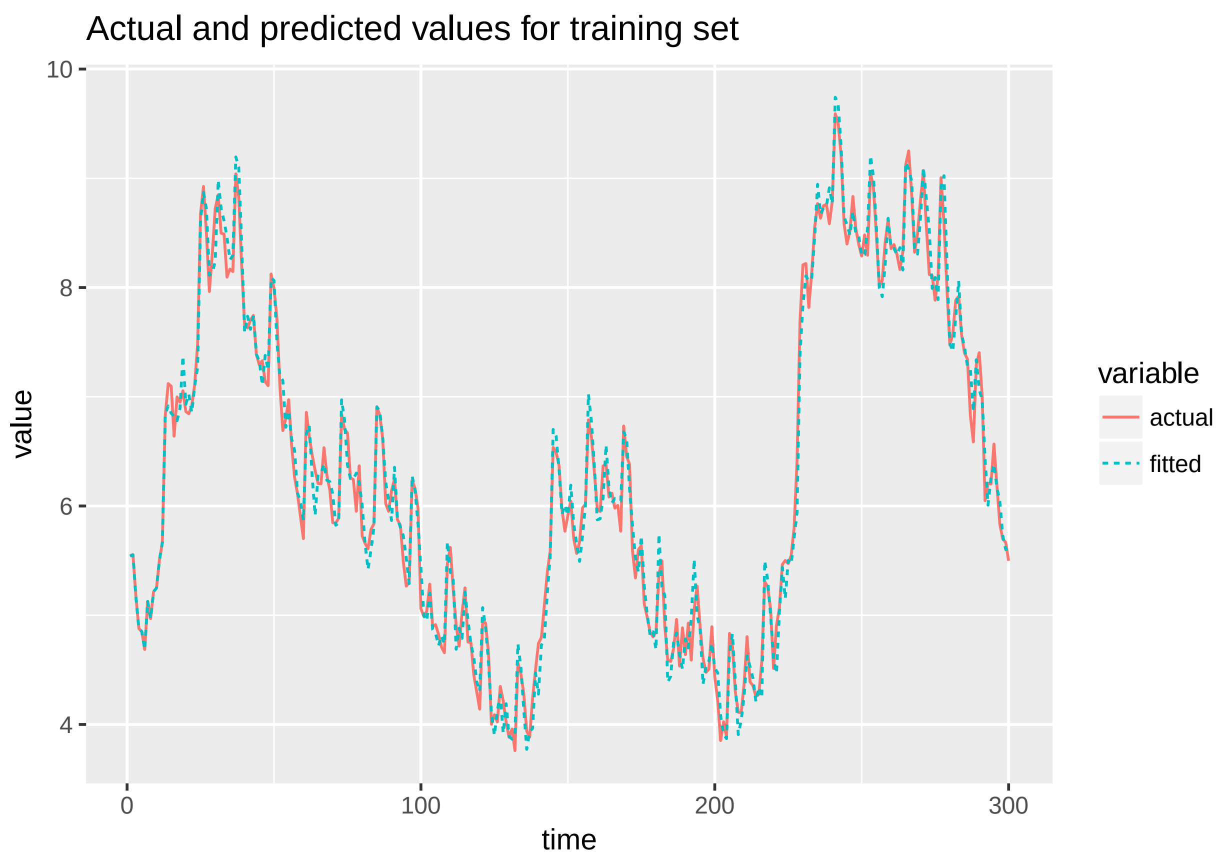
```
model_df <- data.frame(actual = as.numeric(train.time.series$x), fitted = as.numeric(model$fitted))

df_melt <- cbind(melt(model_df), rbind(cbind(1:300), cbind(1:300)))
```

```
## No id variables; using all as measure variables
```

```
colnames(df_melt) <- c("variable", "value", "time")
df_melt$variable <- factor(df_melt$variable, levels=c("actual", "fitted"))

ggp <- ggplot(df_melt, aes(x=time, y=value, group=variable, color=variable))
ggp+geom_line(aes(linetype=variable))+
  ggtitle("Actual and predicted values for training set")
```




Our model appears to do an excellent job fitting the data in-sample.

We will now plot the 11 forecasted values.

```
model_forecast <- forecast(model, h = 11)
model_forecast_df <- data.frame(actual = as.numeric(test.time.series$x),
                                forecasted = as.numeric(model_forecast$mean),
                                lower=as.numeric(model_forecast$lower[,2]),
                                upper=as.numeric(model_forecast$upper[,2]))
df_melt <- cbind(melt(model_forecast_df, id.vars=c("lower", "upper")),
                 rbind(cbind(1:11), cbind(1:11)))
head(df_melt)
```

```
##      lower    upper variable value rbind(cbind(1:11), cbind(1:11))
## 1 5.986144 6.701206   actual 6.449                               1
## 2 5.778516 6.807634   actual 6.425                               2
## 3 5.226962 6.577662   actual 5.929                               3
## 4 4.528965 6.192747   actual 5.536                               4
## 5 4.406898 6.379665   actual 5.472                               5
## 6 4.421315 6.698141   actual 5.240                               6
```

```
colnames(df_melt) <- c("lower", "upper", "variable", "value", "time")
df_melt$variable <- factor(df_melt$variable, levels=c("actual", "forecasted"),
                           labels=c("actual", "forecasted (with 95% CI)"))
df_melt[df_melt$variable=='actual',]$lower <- NA
df_melt[df_melt$variable=='actual',]$upper <- NA
```

```r
ggp <- ggplot(df_melt, aes(x=time, y=value, group=variable, color=variable))
ggp+geom_line(aes(linetype=variable))+
  geom_ribbon(aes(ymin=lower, ymax=upper), alpha=.1, color=F)+
  ggtitle("Actual and forecasted values for test set")
```


Our model seems to have captured the seasonal trends well, with our forecasted values generally going up and down with the actual values. The non-seasonal trend of the decline from the end of the training data was captured as well. Naturally, the 95% confidence interval represented by the gray ribbon increases as the time increases as there is less data in the "memory" of the model, and as the time increases, the fit of the forecasted value gets a little worse. Still, we believe this shows that our model does a satisfactory job in forecasting the test data.