Module 2: Basic operations and Decompositions

MATH1307 Forecasting

RMIT University, School of Science, Mathematical Sciences

Introduction

We need to be familiar with a bunch of basic operations with time series data.

These operations include taking a lag of series, ordinary and seasonal differencing, and transformations.

Differencing is used to deal with nonstationary and transformations are used to deal with changing variance over time.

Sometimes, series can turn out to be stationary after a transformation without any differencing.

Therefore, it is important to check stationarity after application of a transformation before going for a differencing.

Another important operation is to decompose a given series into its trend and seasonal components.

By this way, we can have an insight into the seasonal and trend characteristics of the series independent of each other.

Also, we can adjust the series for trend and seasonality. Eventually, is it is possible to use decomposition for forecasting.

In this module, we will

- start with basic operations with time series data and focus on
    - Lag operation,
    - Ordinary difference,
    - Seasonal difference, and
    - Transformations
- then we will study STL decomposition
    - Forecasting after decomposition.

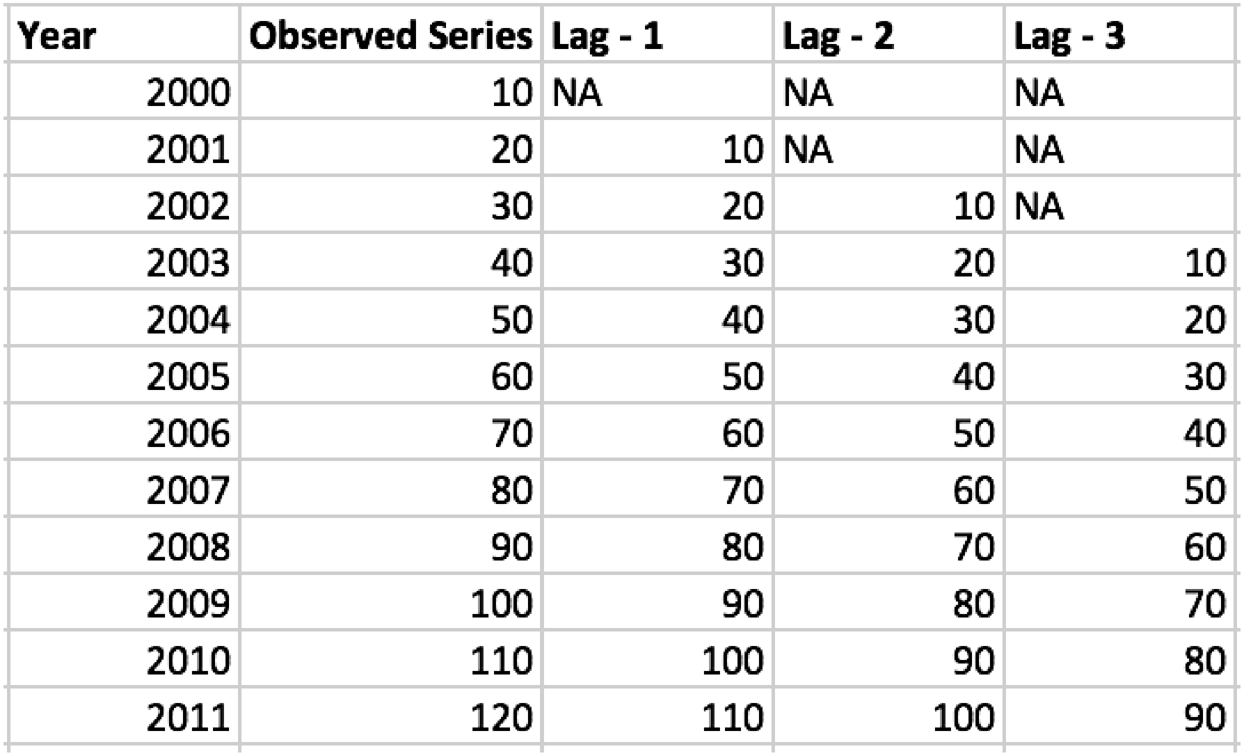
Basic operations with time series

Basic operations that we will apply with time series data include

- lag operation,
- differencing, and
- transformations.

Lag operation

To calculate kth lag, we push the series k steps down and leave the first k-elements empty.



Year	Observed Series	Lag - 1	Lag - 2	Lag - 3
2000	10	NA	NA	NA
2001	20	10	NA	NA
2002	30	20	10	NA
2003	40	30	20	10
2004	50	40	30	20
2005	60	50	40	30
2006	70	60	50	40
2007	80	70	60	50
2008	90	80	70	60
2009	100	90	80	70
2010	110	100	90	80
2011	120	110	100	90

The function `Lag()` from `Hmisc` package can make shifting operation in both ways.

```r
library(Hmisc)
x.series = ts(c(2,4,6,8,10,12,14,16,18,20))

# If the second argument is set to a negative integer,
# it shifts series values k steps up
k = -1
Lag(x.series, k)
```

```
## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##  [1]  4  6  8 10 12 14 16 18 20 NA
```

```r
# If the second argument is set to a negative integer,
# it shifts series values k steps down
k = 1
Lag(x.series, k) # This is the lag operation we will use
```

```
## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##  [1] NA  2  4  6  8 10 12 14 16 18
```

```r
k = -3
Lag(x.series, k)
```

```
## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##  [1]  8 10 12 14 16 18 20 NA NA NA
```

```r
k = 3
Lag(x.series, k) # This is the lag operation we will use
```

```
## Time Series:
## Start = 1
## End = 10
## Frequency = 1
##  [1] NA NA NA  2  4  6  8 10 12 14
```

Differencing

We have two kinds of difference operations: ordinary difference and seasonal difference.

The main aim of the ordinary differencing operation is to deal with non-stationarity.

The seasonal differencing is used to remove the overall effect of seasonality from the series.

Ordinary difference

Differencing is the operation to compute the change between two consecutive observations in the original series.

If we repeat this operation twice, we take the second difference of the series.

Or, if we repeat this operation k times, we take the kth difference of the series.

To write this operation mathematically, suppose we are given the series $\{Y_t\}$.

The difference of the original series is

$$\nabla Y_t = Y_t - Y_{t-1}.$$

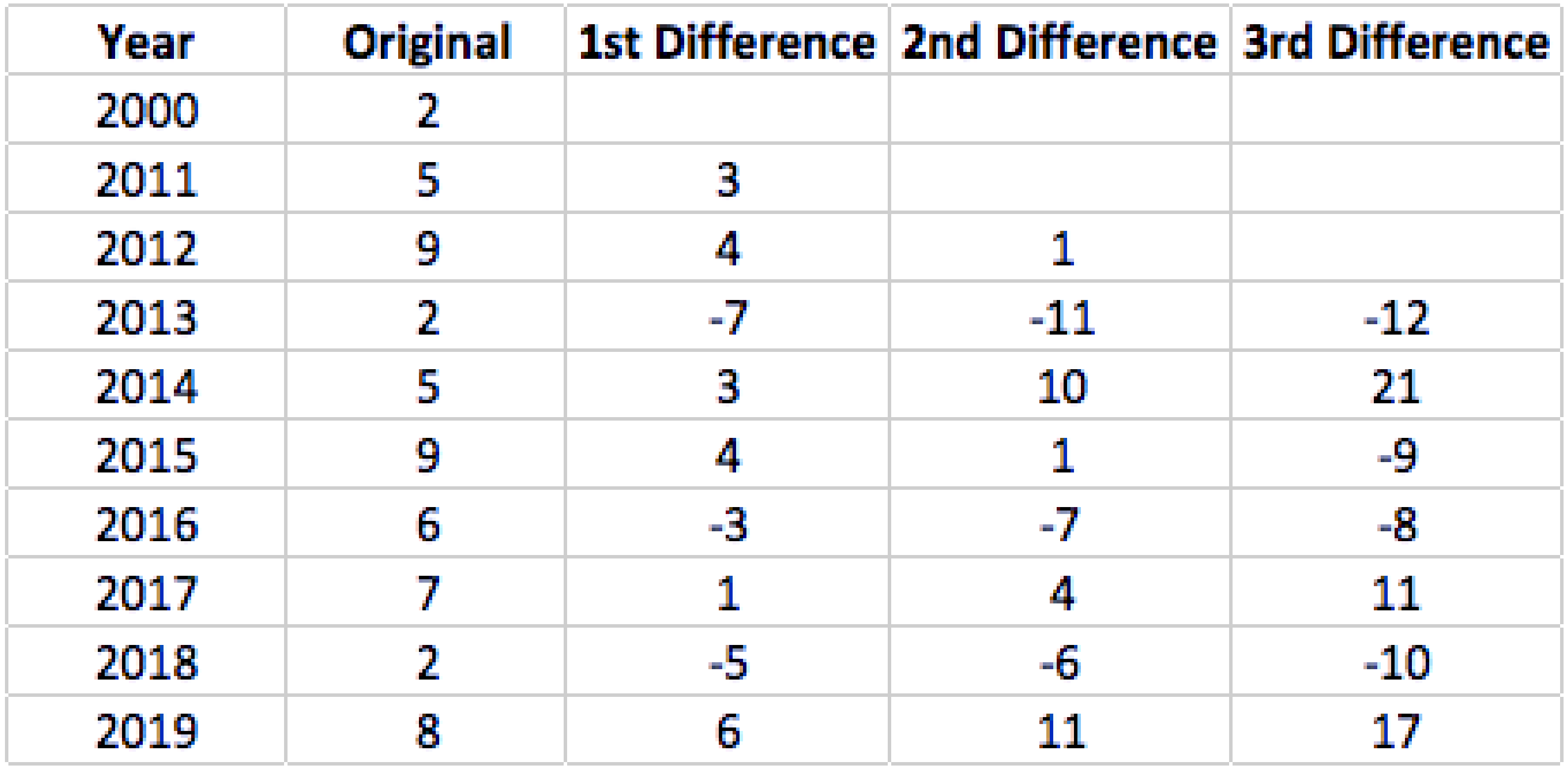
The second difference of the original series is

$$\begin{aligned}
\nabla^2 Y_t \quad &= \nabla Y_t - \nabla Y_{t-1} \\
&= (Y_t - Y_{t-1}) - (Y_{t-1} - Y_{t-2}) \\
&= Y_t - 2Y_{t-1} + Y_{t-2}.
\end{aligned}$$

We can go on in this way and find k times differenced series.

This kind of differencing is called *ordinary differencing*.

Differencing operation is illustrated in the following table.



Year	Original	1st Difference	2nd Difference	3rd Difference
2000	2			
2011	5	3		
2012	9	4	1	
2013	2	-7	-11	-12
2014	5	3	10	21
2015	9	4	1	-9
2016	6	-3	-7	-8
2017	7	1	4	11
2018	2	-5	-6	-10
2019	8	6	11	17

The following code chunk is used to apply ordinary differencing.

```r
x.series = ts(c(2,5,9,2,5,9,6,7,2,8))
# First difference
diff(x.series)
```

```
## Time Series:
## Start = 2
## End = 10
## Frequency = 1
## [1]  3  4 -7  3  4 -3  1 -5  6
```

```r
# Second difference
diff(x.series, differences = 2)
```

```
## Time Series:
## Start = 3
## End = 10
## Frequency = 1
## [1]   1 -11  10   1  -7   4  -6  11
```

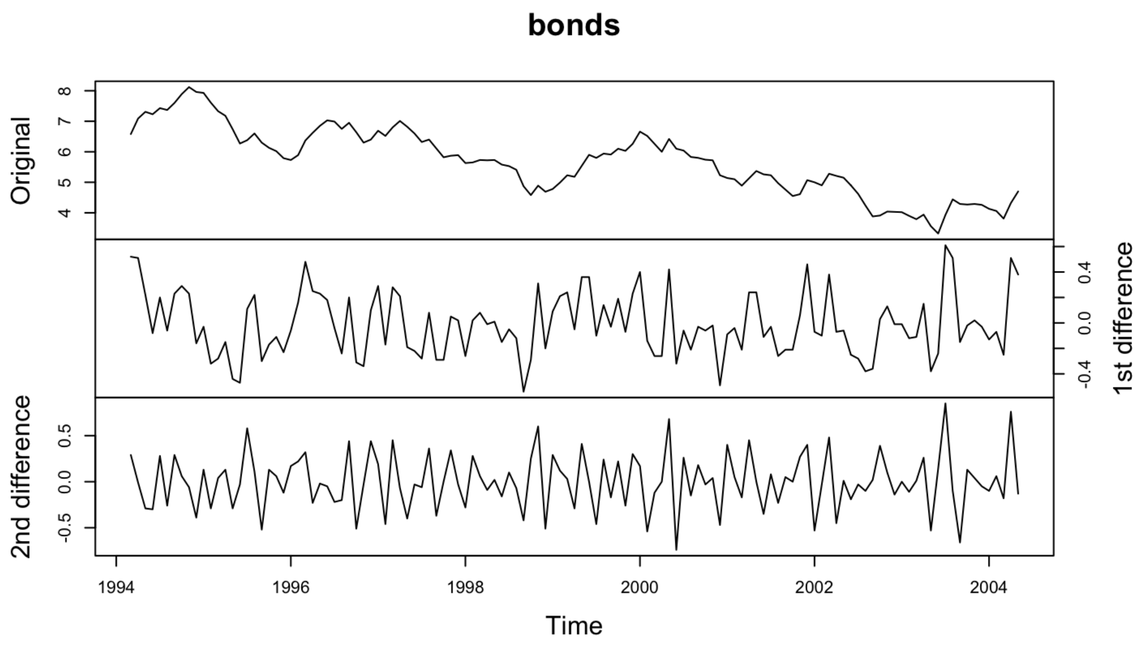
```r
# Third difference
diff(x.series, differences = 3)
```

```
## Time Series:
## Start = 4
## End = 10
## Frequency = 1
## [1] -12  21  -9  -8  11 -10  17
```

The following time series plots show the original series and its first difference for monthly US government bond yields series.

```r
data(bonds) # Data from expsmooth package
# Create an intersection of original, first and
# second difference of the series
bonds = ts.intersect(bonds , diff(bonds) , diff(bonds , differences = 2))
colnames(bonds) =  c("Original","1st difference","2nd difference")
```

```
plot(bonds , yax.flip=T)
```



While the first differencing detrended the series, the second differencing made it bounce more closely around the mean level.

Seasonal difference

Seasonal differencing is applied over the seasons in the observed series.

The time frame in which the seasonal pattern repeats itself is called *period* of the series.

Suppose p is the period of the series, then the first seasonal difference is

$$\nabla_p Y_t = Y_t - Y_{t-p}.$$

The second seasonal difference of the original series is

$$\begin{aligned}\nabla_p^2 Y_t &= \nabla_p Y_t - \nabla_p Y_{t-p-1} \\ &= (Y_t - Y_{t-p}) - (Y_{t-1} - Y_{t-p-1}) \\ &= Y_t - Y_{t-1} - y_{t-p} + Y_{t-p-1}.\end{aligned}$$

The following code chunk illustrates seasonal differencing with artificial seasonal data.

```
x.series = ts(c(2,5,9,12,3,6,8,11,1,3,6,10,5,6,9,14),frequency = 4)
x.series
```

```
##   Qtr1 Qtr2 Qtr3 Qtr4
## 1    2    5    9   12
## 2    3    6    8   11
## 3    1    3    6   10
## 4    5    6    9   14
```

```r
p = 4
# First seasonal difference with period 4
diff(x.series,lag = frequency(x.series))
```

```
##   Qtr1 Qtr2 Qtr3 Qtr4
## 2    1    1   -1   -1
## 3   -2   -3   -2   -1
## 4    4    3    3    4
```

```r
# Second seasonal difference with period 4
diff(x.series, differences = 2, lag=4)
```

```
##   Qtr1 Qtr2 Qtr3 Qtr4
## 3   -3   -4   -1    0
## 4    6    6    5    5
```

```r
# Third seasonal difference with period 4
diff(x.series, differences = 3, lag=4)
```

```
##   Qtr1 Qtr2 Qtr3 Qtr4
## 4    9   10    6    5
```

We can apply ordinary and seasonal differencing successively.

If there is a strong seasonality in the original series, application of seasonal differencing first is recommended because sometimes the resulting series will be stationary and there will be no need for a further first difference.

If first differencing is done first, we need to keep in mind that the seasonality will remain in the series.

Transformations

There are lots of different transformation that can be applied to time series data.

But the frequently used ones are natural logarithm, square root, and reciprocal transformations.

All of these transformations can be collected under a wide transformation family called *Box-Cox* or *Power* Transformations (Box and Cox, 1964).

Box-Cox transformation is defined by

$$\begin{aligned} g(x) &= \frac{x^\lambda - 1}{\lambda}, && \text{for } \lambda \neq 0 \\ &= \log(x), && \text{for } \lambda = 0 \end{aligned}$$

The Box-Cox transformation turns out to be

- **square root** transformation when $\lambda = 0.5$, which is useful useful with Poisson-like data,
- **reciprocal** transformation when $\lambda = -1$.

The power transformation applies only to positive data values.

If some of the values are negative or zero, a positive, *as small as possible* constant may be **added** (**not** *multiplied*) to all of the values to make them all positive before doing the power transformation.

The shift is often determined subjectively.

We can consider λ as an additional parameter in the model to be estimated from the observed data.

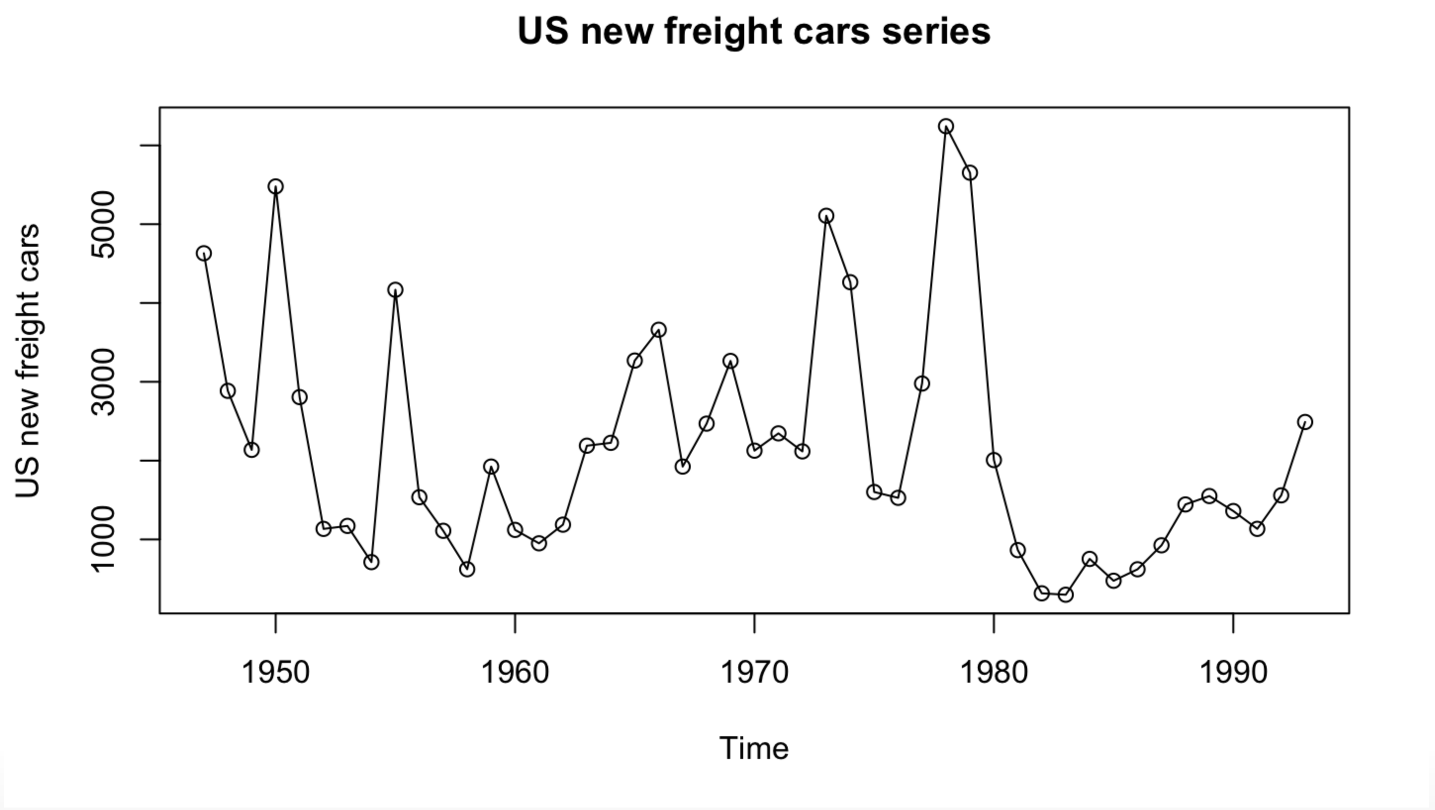
However, a precise estimation of λ is usually not warranted.

Evaluation of a range of transformations based on a grid of λ values, say $\pm 1, \pm 1/2, \pm 1/3, \pm 1/4$, and 0, will usually suffice and may have some intuitive meaning.

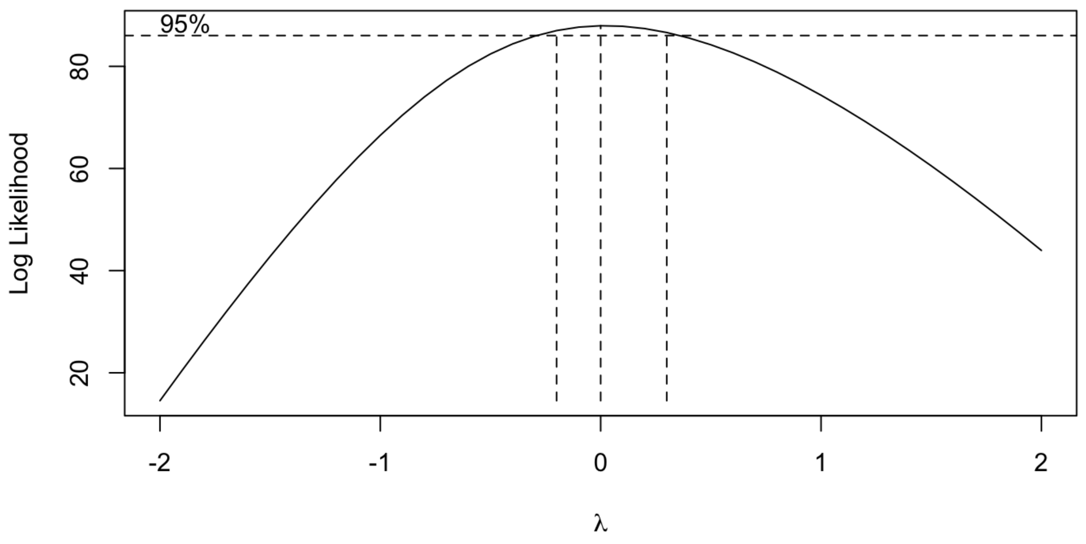
The software allows us to consider a range of lambda values and calculate a log-likelihood value for each lambda value based on a normal likelihood function.

A plot of these values is generated for the annual US new freight cars between 1947 and 1993 data.

```r
data(freight) # Data from expsmooth package
plot(freight,ylab='US new freight cars',xlab='Time',type='o', main="US new freight cars series")
```



```
BC = BoxCox.ar(freight)
```

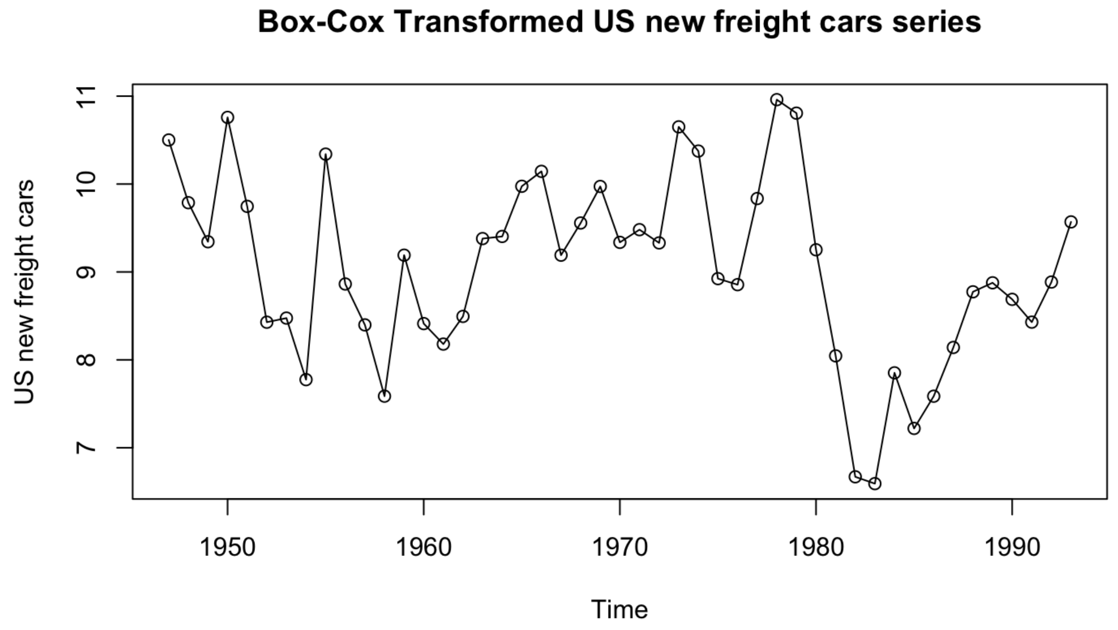


```
BC$ci
```

```
## [1] -0.2  0.3
```

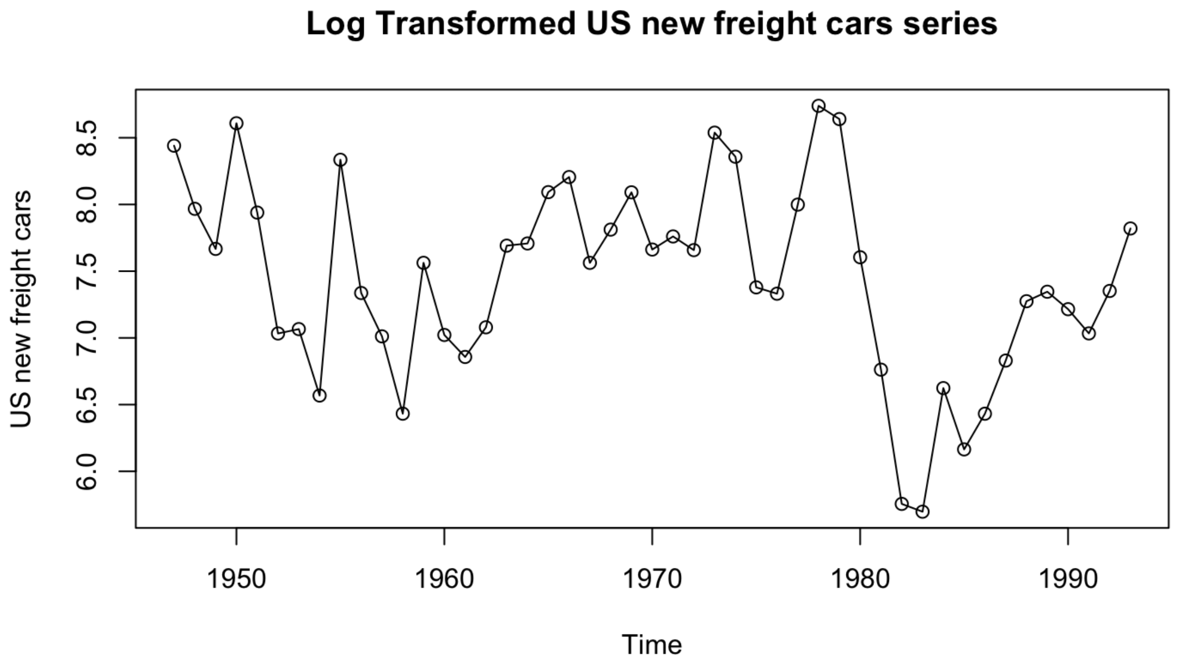
The centre of 95% confidence interval is 0.05. So, we can use $\lambda = 0.05$ for the transformation.

```
lambda = 0.05
BC.freight = ((freight^(lambda)) - 1) / lambda
plot(BC.freight,ylab='US new freight cars',xlab='Time',type='o', main="Box-Cox Transformed US new freight cars s
```



The 95% confidence interval for λ contains the value of $\lambda = 0$ quite near its centre and strongly suggests a logarithmic transformation ($\lambda = 0$) for these data. So, we can directly apply the **log** transformation.

```
BC.freight = log(freight)
plot(BC.freight,ylab='US new freight cars',xlab='Time',type='o', main="Log Transformed US new freight cars serie
```



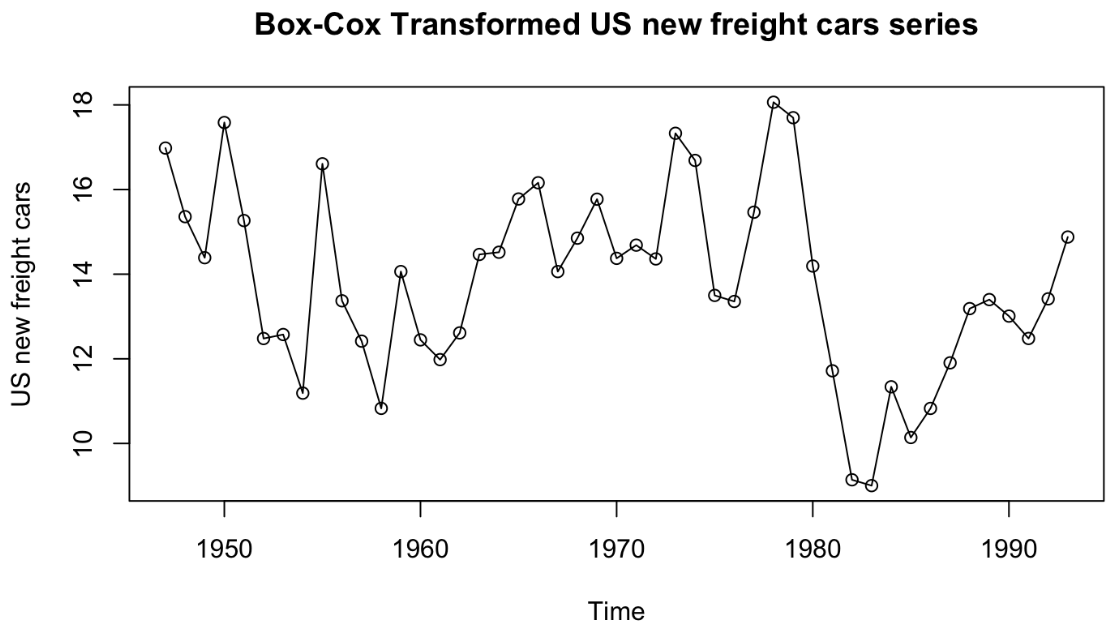
It is also possible to apply Box-Cox transformation using `BoxCox()` function from the `forecast` package. Also, `BoxCox.lambda()` from the same package provides another way of choosing the value of λ.

```
lambda = BoxCox.lambda(freight, method = "loglik")
lambda
```

```
## [1] 0.15
```

```
BC.freight2 = BoxCox(freight, lambda = lambda)
```

```
plot(BC.freight2,ylab='US new freight cars',xlab='Time',type='o', main="Box-Cox Transformed US new freight cars
```



Decomposition of time series

Decomposition of time series into different components is useful to observe the individual effects of the existing components and historical effects occurred in the past.

The components that a time series can be decomposed into are

- *seasonal*,
- *trend*, and
- *remainder*,

which includes other effects that are not captured by the seasonal and trend components.

Basically there are three main decomposition methods for time series.

The most basic one is the *classical decomposition*, which provides the basis for other decomposition methods. The *X-12-ARIMA decomposition* is another decomposition which is more complex than the classical decomposition.

It is mostly used for quarterly and monthly data. One of the very robust and commonly used decomposition methods is the Seasonal and Trend decomposition using Loess (STL) decomposition (Cleveland et al., 1990).

When a time series is displayed, it includes trend and seasonal effect in a confounded way; hence, it would be very difficult to infer about the main characteristics of the series under the effect of seasonality.

Therefore, we use time series decomposition to extract each component from the series and adjust the series for various effects like seasonality.

Classical decomposition

This method originates back to 1920s. it has additive and multiplicative forms.

The additive form is formulated as

$$Y_t = T_t + S_t + R_t$$

where T_t is the trend component, S_t is the seasonal component, and R_t is the remainder after exracting trend and seasonal components form the series of interest.

Alternatively, a multiplicative model is written as

$$Y_t = T_t \cdot S_t \cdot R_t.$$

In the classical decomposition with any of the additive or multiplicative models, first, the series is detrended by extracting the trend component using moving averages and then the seasonal component is estimated by averaging, for each time unit, over all periods.

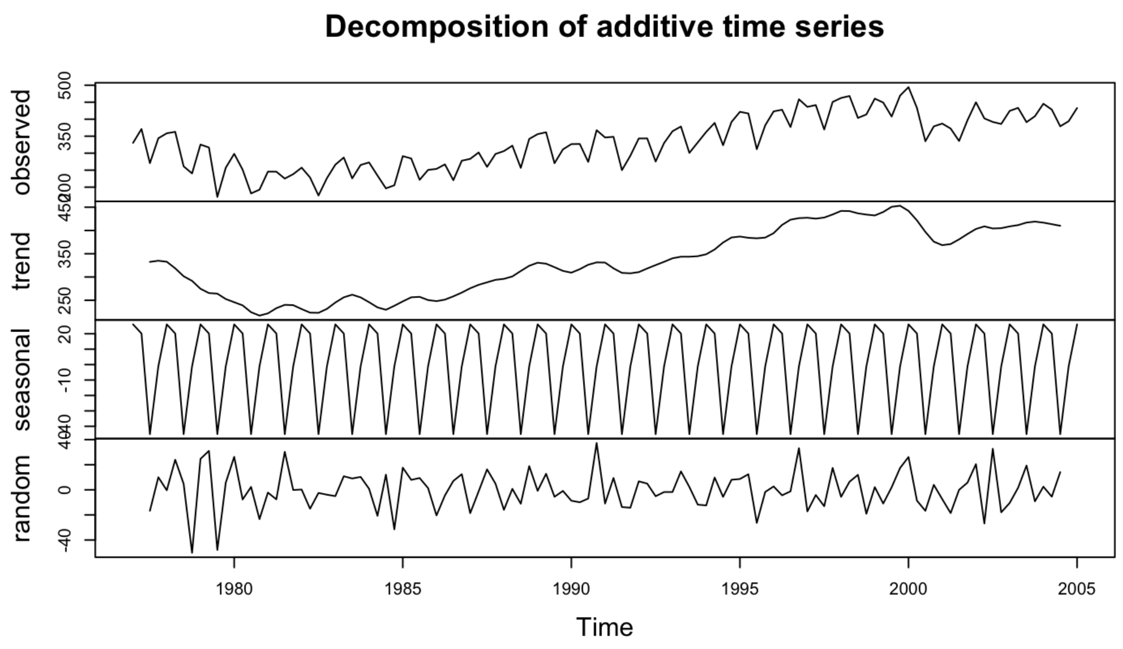
Notice that to get precise estimates, we should have an integer number of complete periods in the series of interest.

To implement the classical decomposition, we use `decompose()` function from the `stats` package.

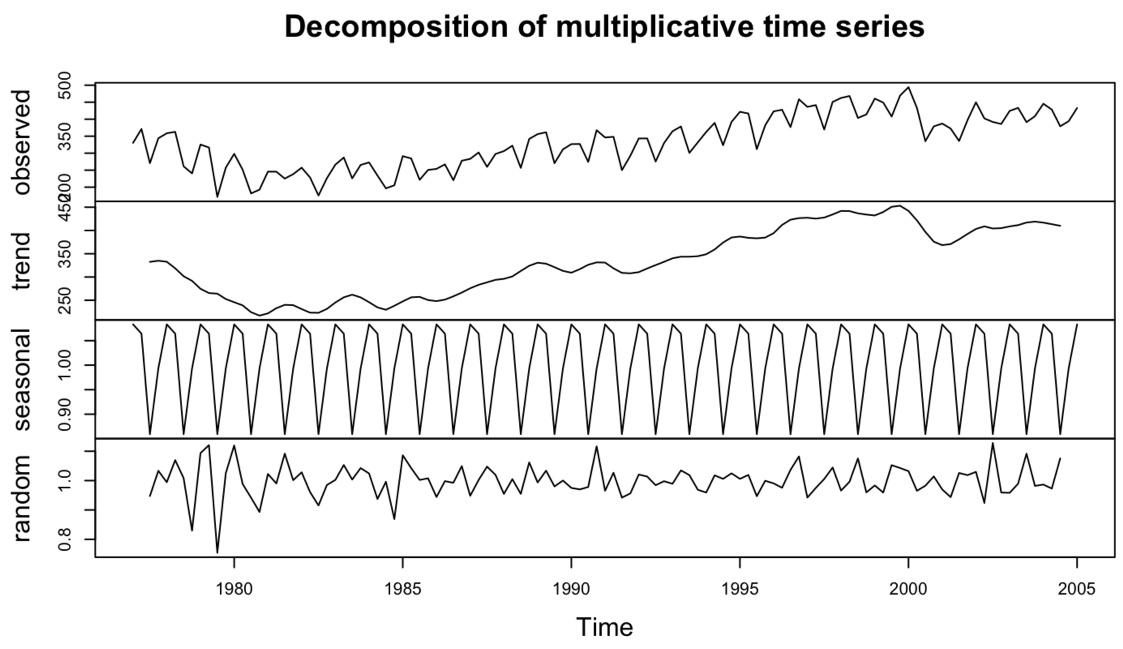
The argument `type = c("additive", "multiplicative")` is used to speecify the type of model used for the decomposition.

Let’s illustrate the classical decomposition using the quarterly numbers of passenger motor vehicle production in the UK (thousands of cars) series.

```r
fit.classical.add <- decompose(ukcars, type="additive")
plot(fit.classical.add)
```



```
fit.classical.mult <- decompose(ukcars, type="multiplicative")
plot(fit.classical.mult)
```



We got very similar estimates from the additive and multiplicative methods. It seems from the remainder series that the multiplicative model handles the decomposition slightly better.

Notice that because there are better decomposition methods available, the use of the classical decomposition is not recommended (Hyndman and Athanasopoulos, 2014).

X-12-ARIMA decomposition

This method was developed by the US Bureau of the Census and is widely used by the Bureau and government agencies around the world (Hyndman and Athanasopoulos, 2014).

Earlier version is called *X-11* and the new version is called *X-13*.

The X-12 is an upgraded version of the classical decomposition which is not suffering from the major drawbacks of the classical decomposition.

The components are refined by some additional steps, which account for extreme values and apply different moving average techniques, in X-12.

It can also account for day variation, holiday effects and effects of other predictors.

The R package `x12` has recently been introduced to implement the method.

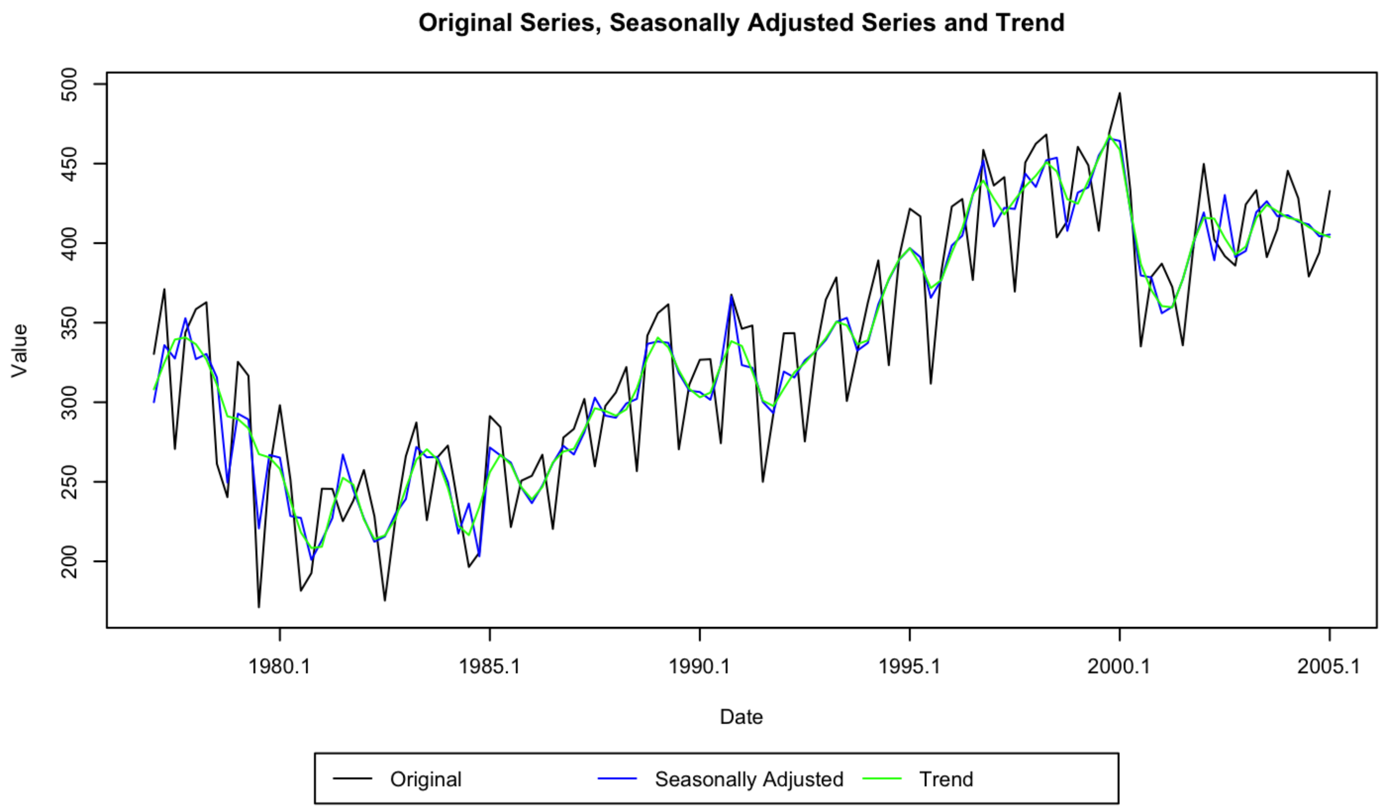
This is a large package and includes lots of different operations around the X-12 method.

We mainly use the function `x12()` to implement the decomposition.

The `plot()` function displays the decomposed elements with the argument `sa=FALSE`, `trend=FALSE`, `log_transform=FALSE`.

The following code chunk applies X-12 decomposition to the passenger motor vehicle series.

```r
fit.x12 = x12(ukcars)
plot(fit.x12 , sa=TRUE , trend=TRUE)
```



From the seasonally adjusted series, we conclude that the decrease in the number of motor vehicles is not due to the seasonal effect.

We observe some deviances from the trend component in the seasonally adjusted series.

When we apply X-12 decomposition, we can also display seasonal-irregular (SI) components and also seasonal factors and corresponding mean levels for each period of seasonal series.

SI ratio is the ratio of the original series to the estimated trend so they are estimates of detrended series.

We can use SI ratios to investigate whether short-term movements are caused by seasonal or irregular fluctuations.

The expectation around the seasonal factor is that it does not change dramatically over the years.

But seasonal pattern would change after some intervention effects.

So, SI ratio helps us the detect such changes in the seasonal pattern (You can find detailed information.

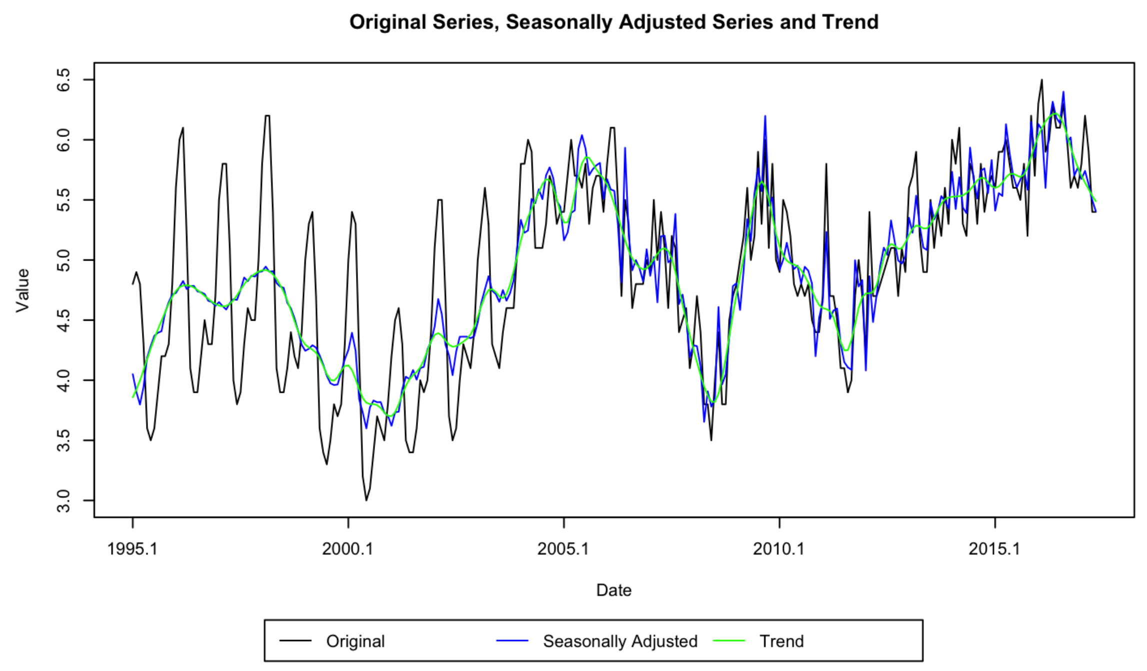
Let’s illustrate the idea over the monthly unemployment rate data of Austria between January 1995 and May 2017.

The data is available through the website of EuroStat.

The following code chunk applies the X-12 decomposition:

```
unemp = read.csv("~/Documents/MATH1307_Forecasting/notes/Module 2/Austria_Unemployment.csv")
unemp.ts = ts(unemp[,2],start = c(1995,1),frequency = 12)
fit.unemp.x12 = x12(unemp.ts)
```

```r
plot(fit.unemp.x12 , sa=TRUE , trend=TRUE)
```



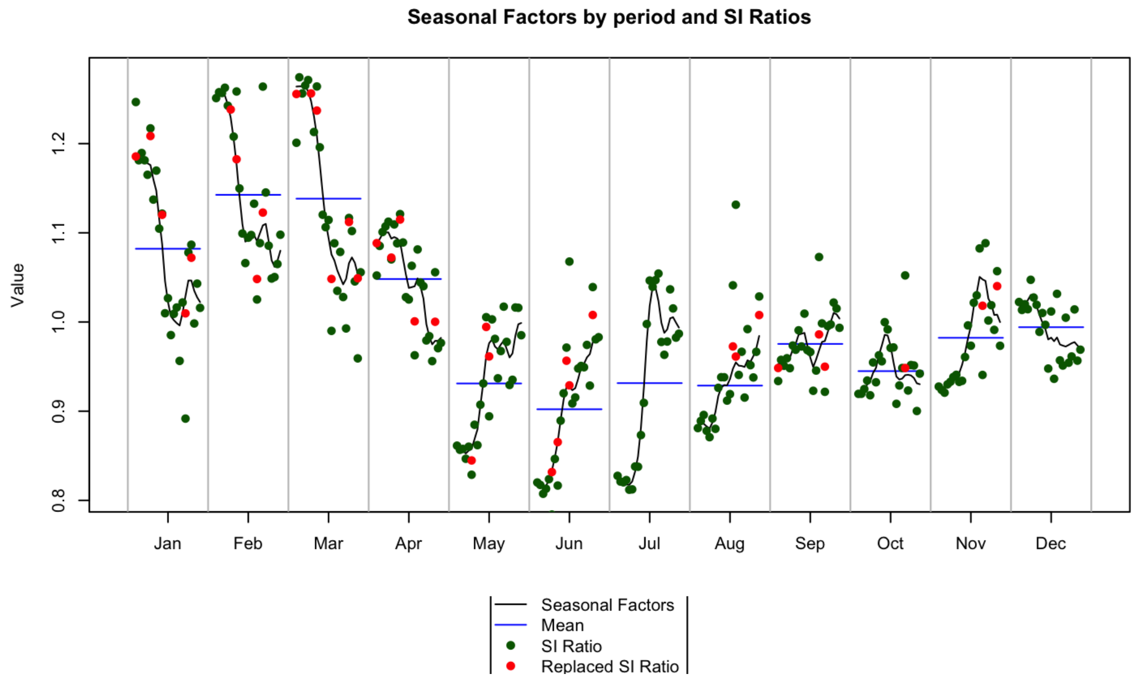
As can be observed from the original series, the pattern of seasonality changes after 2005.

Also, seasonally adjusted series has a different characteristic after this date.

This implies that there are some other factors affecting the series apart from the seasonal effect.

Now, let’s display seasonal factor plots and SI ratios with the following code chunk:

```
plotSeasFac(fit.unemp.x12)
```



The expected thing with this plot is to have seasonal patterns fluctuating around the mean level if there is no change in the seasonal pattern.

However, they divert from the mean level for the months from January to July.

In this plot, replaced SI ratios are the SI ratios with the extreme values replaced.

So, it is also observed from SI ratios that we have some outlier or influential observations for the months January, June, and August.

STL decomposition

STL decomposition has some advantages over the classical decomposition methods such as (Hyndman and Athanasopoulos, 2014)

- STL handles any type of seasonality, whiles others are somewhat limited to only monthly and/or quarterly series.
- The seasonal component can change by the time and the rate of change can be controlled by the user.
- The smoothness of the trend-cycle can also be controlled by the user.
- We can make it robust to outliers by sending the effect of occasional unusual observations to the remainder component.

Notice that STL cannot handle calendar variation and it works over an additive model

$$Y_t = T_t + S_t + R_t$$

where $t = 1, \ldots, N$ (Cleveland et al., 1990).

It is possible to make it multiplicative by using the log transformation.

Also, it is possible to work somewhere between additive and multiplicative settings using the Box-Cox transformation.

To implement STL, we use `stl()` function from the `stats` package of R.

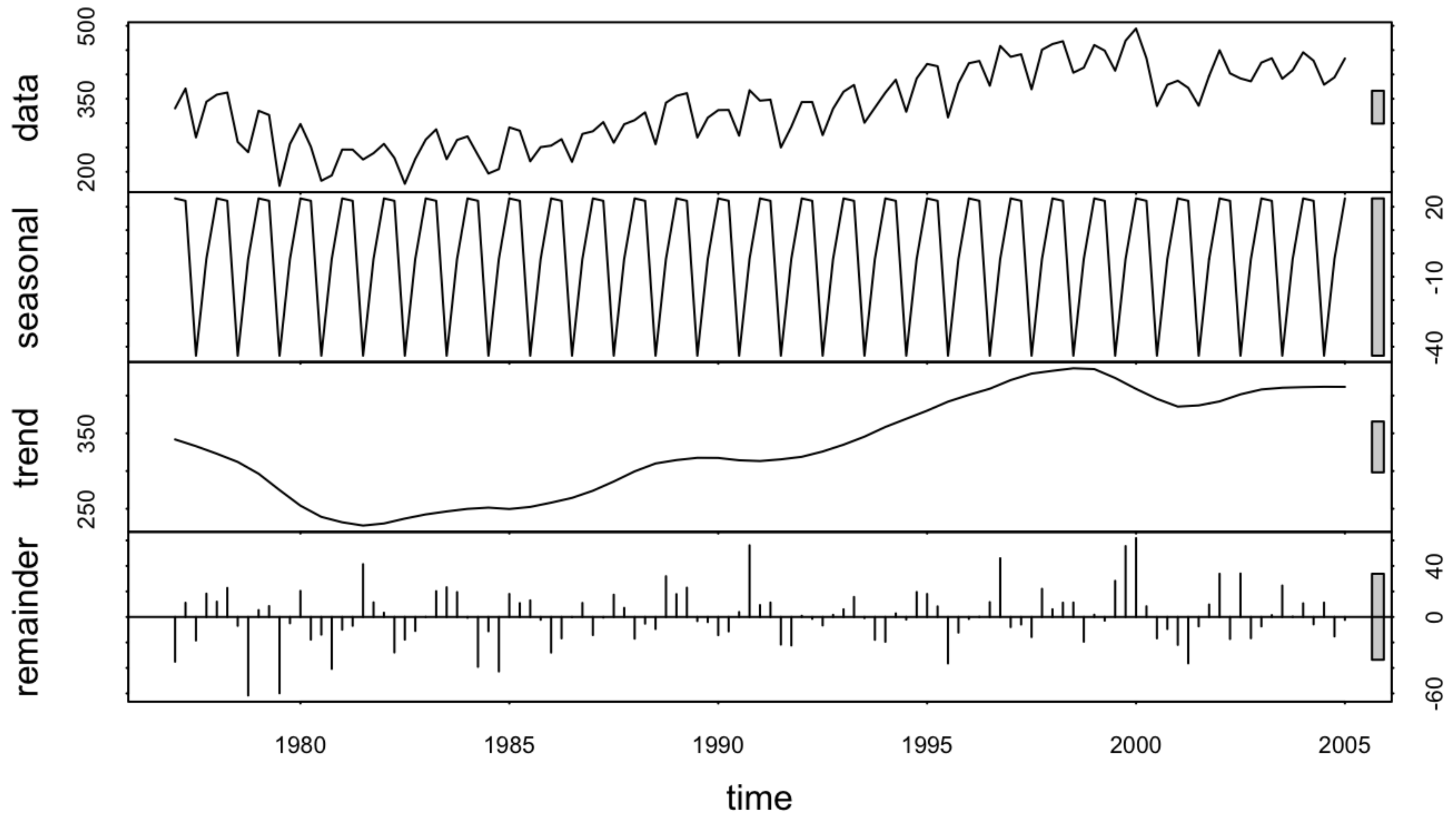
There are two main arguments with this function. `t.window` and `s.window` arguments are used to control the rate at which the trend and seasonal components will change.

The smaller the value is the more rapid the change is.

If we set the seasonal window to infinity, we force the seasonal component to be periodic, which implies that it will be identical across years (Hyndman and Athanasopoulos, 2014).

The following code chunk implements the STL decomposition for quarterly numbers of passenger motor vehicle production in the UK (thousands of cars) series, which we have thought to be seasonal.

```r
fit.vehicles <- stl(ukcars, t.window=15, s.window="periodic", robust=TRUE)
plot(fit.vehicles)
```


data
seasonal
trend
remainder
500
350
200
20
-10
-40
350
250
40
0
-60
1980
1985
1990
1995
2000
2005
time

In this output, the first display it the smoothed version of the time series plot of the original series.

The second one shows the seasonal component on the vehicle series.

The seasonal pattern shows that the higher numbers of production were recorded in the first, second and fourth quarters.

The third one is for the trend component and the last one is the time series *bar* chart for the remainder series.

The grey bars to the right of each panel show the relative scales of the components.

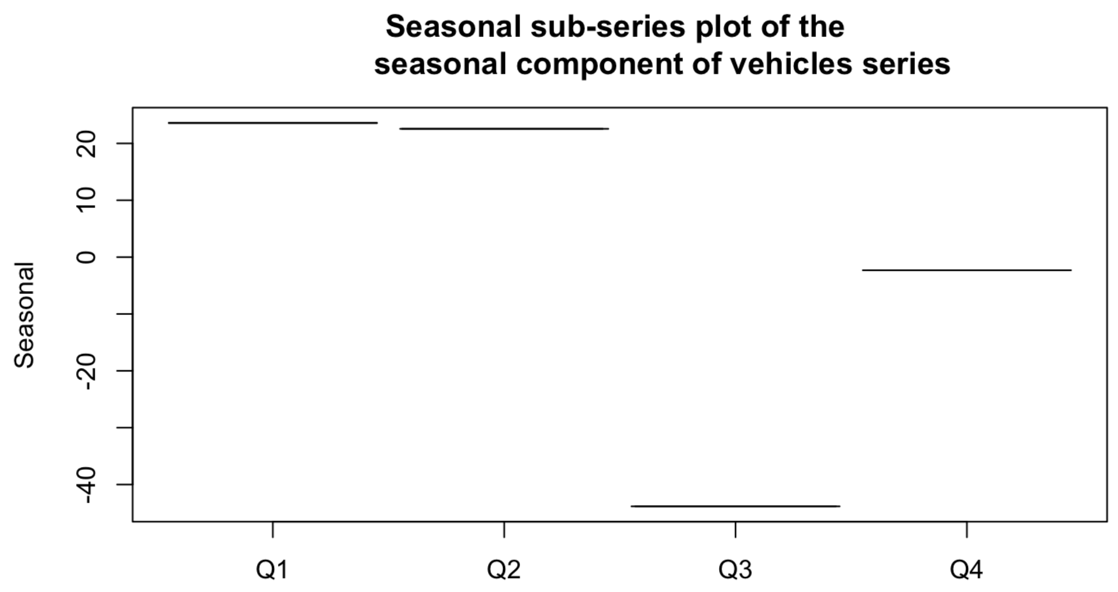
Each grey bar represents the same length but because the plots are on different scales, the bars vary in size (Hyndman and Athanasopoulos, 2014).

Notice that we should not use this decomposition to decide on the existence of trend or seasonality.

We should use this technique after deciding that there is seasonality in the series and the use it to infer about the characteristics of the smoothed seasonal component.

We can display seasonal sub-series using the `monthplot()` function. This helps us to get a sense of the variation in the seasonal component over time.

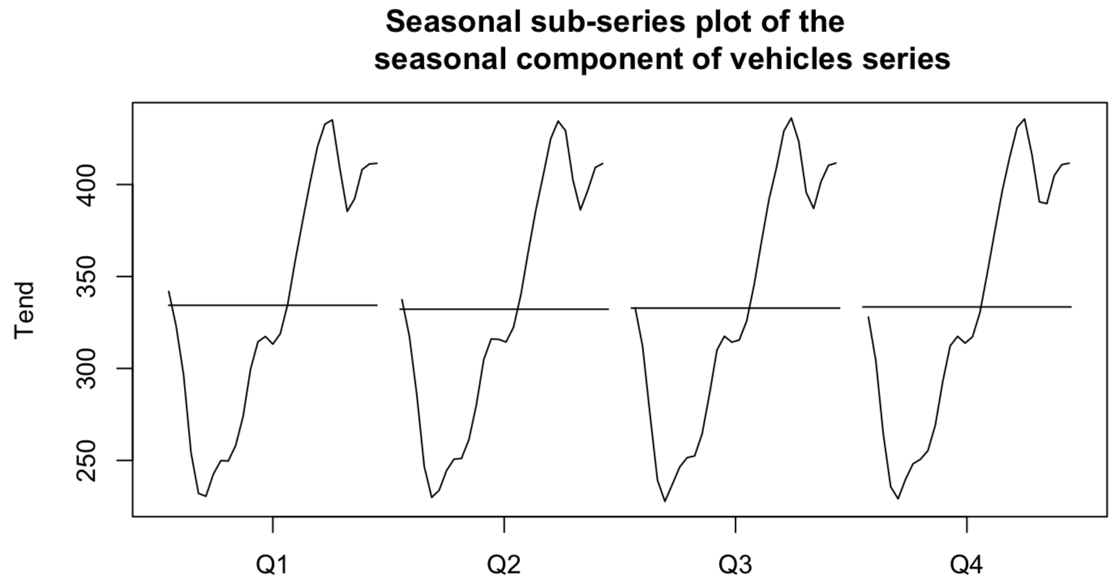
```r
monthplot(fit.vehicles$time.series[,"seasonal"],choice = "seasonal", main="Seasonal sub-series plot of the
          seasonal component of vehicles series", ylab="Seasonal")
```



We get only mean levels in this plot because the seasonal component repeats itself without changing. So there are no fluctuations around the mean level.

We can also display monthly trend component by setting `choice = "trend"`.

```r
monthplot(fit.vehicles,choice = "trend", main="Seasonal sub-series plot of the
          seasonal component of vehicles series", ylab="Tend")
```

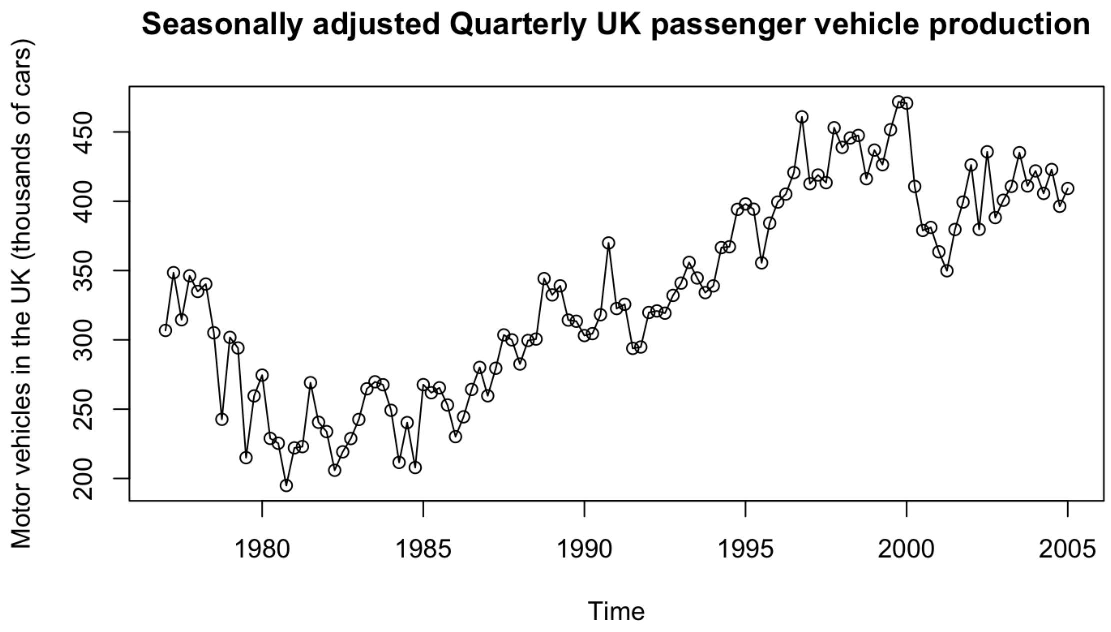


From this visualisation, we can observe the same trend pattern for each quarter.

After having the decomposition, we can adjust the series for seasonality by subtracting the seasonal component from the original series.

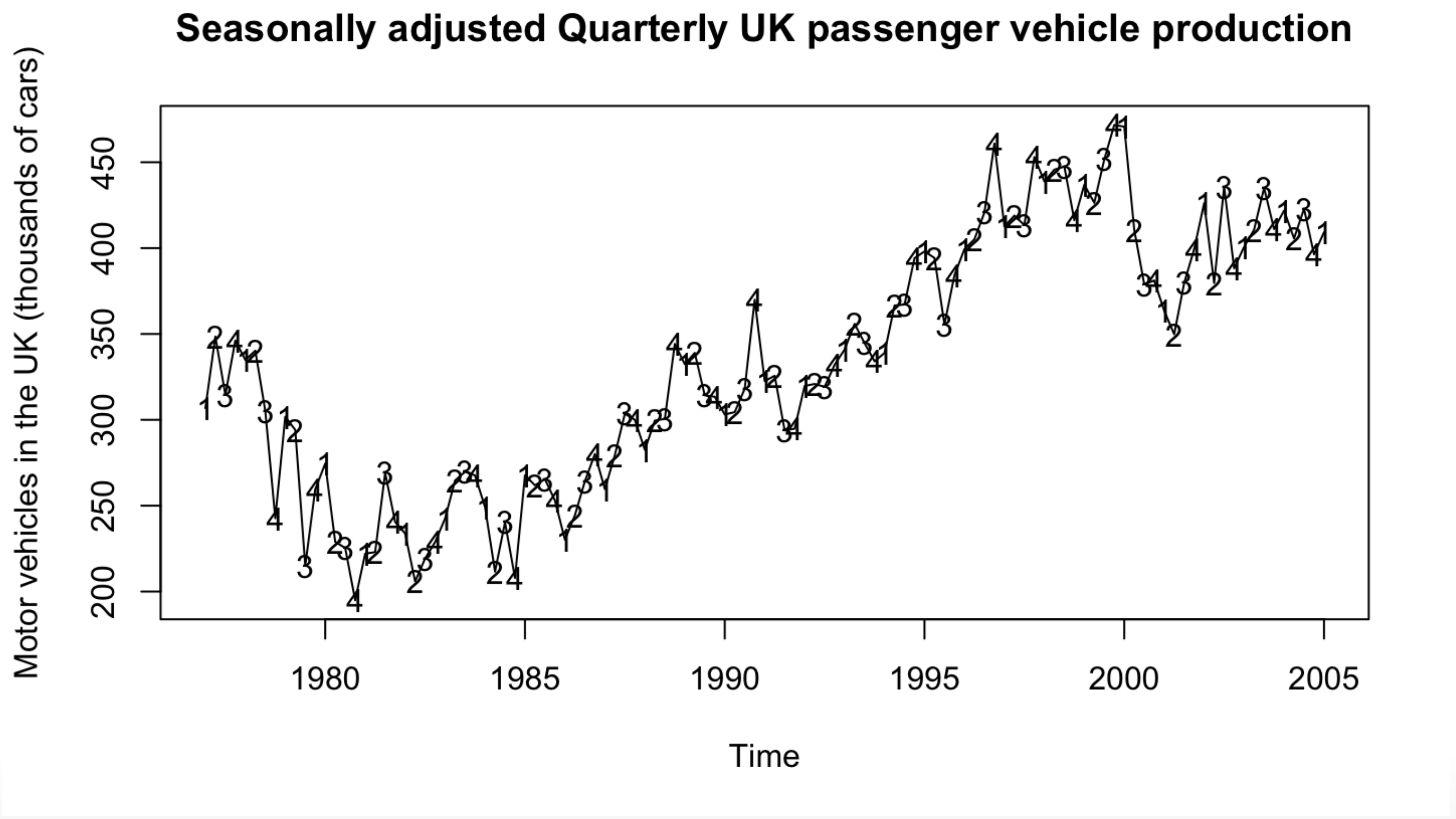
The following code chunk illustrates this approach.

```
fit.vehicles.seasonal = fit.vehicles$time.series[,"seasonal"] # Extract the seasonal component from the output
vehicles.seasonally.adjusted = ukcars - fit.vehicles.seasonal
plot(vehicles.seasonally.adjusted, ylab='Motor vehicles in the UK (thousands of cars)',xlab='Time',type='o', mai
```



We can write labels on the plot to get a clearer sense.

```
plot(vehicles.seasonally.adjusted, ylab='Motor vehicles in the UK (thousands of cars)',xlab='Time', main = "Seas
points(y=vehicles.seasonally.adjusted,x=time(vehicles.seasonally.adjusted), pch=as.vector(season(vehicles.season
```

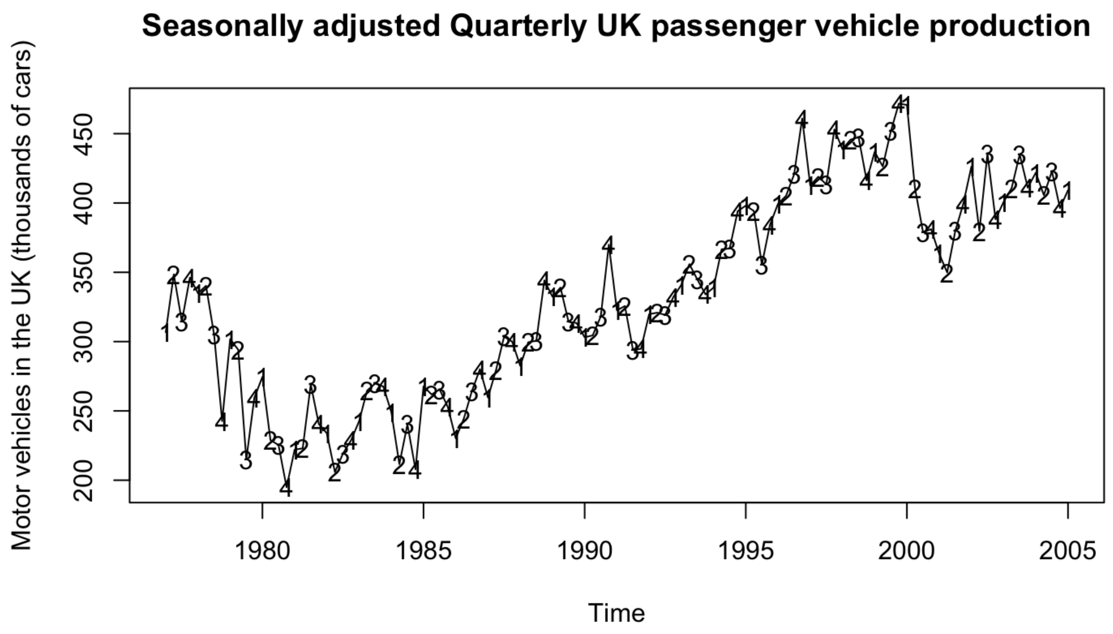


As observed from the plots above, we filtrated the effect of seasonal component.

We see that the labels for quarters are swapping with each other in more random order.

Alternatively, we can use `seasadj()` to get seasonally adjusted series directly.

```
plot(seasadj(fit.vehicles), ylab='Motor vehicles in the UK (thousands of cars)',xlab='Time', main = "Seasonally
points(y=seasadj(fit.vehicles),x=time(seasadj(fit.vehicles)), pch=as.vector(season(seasadj(fit.vehicles))))
```

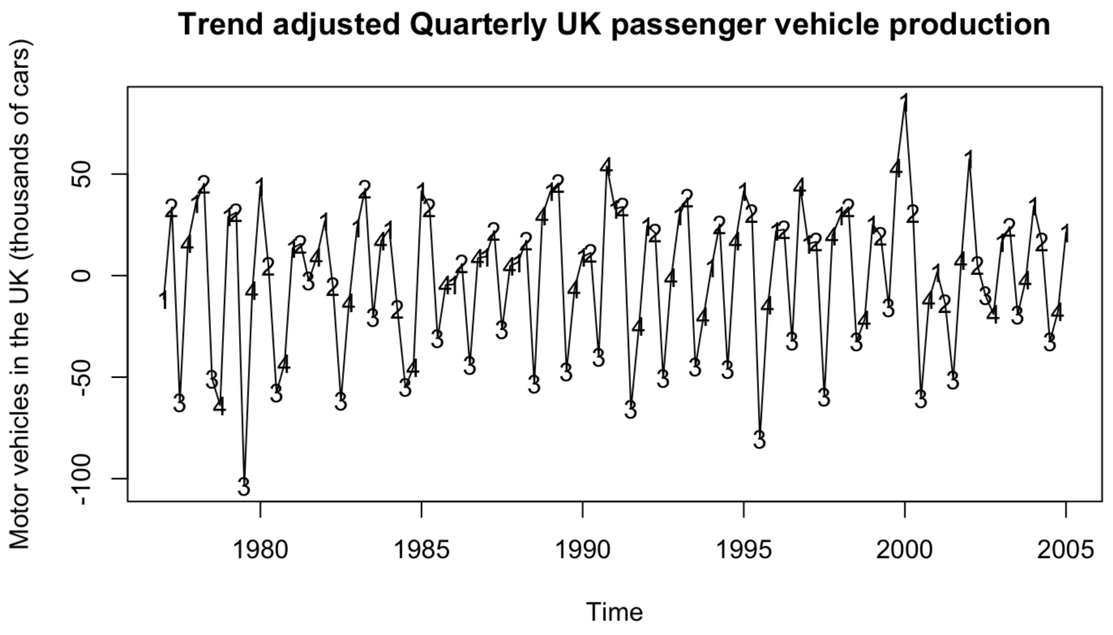


After the decomposition, we can adjust for trend as well.

For this, we need to subtract the trend component from the original series.

The following code chunk illustrates this approach.

```
fit.vehicles.trend = fit.vehicles$time.series[,"trend"] # Extract the trend component from the output
vehicles.trend.adjusted = ukcars - fit.vehicles.trend
plot(vehicles.trend.adjusted, ylab='Motor vehicles in the UK (thousands of cars)',xlab='Time', main = "Trend adj
points(y=vehicles.trend.adjusted,x=time(vehicles.trend.adjusted), pch=as.vector(season(vehicles.trend.adjusted))
```



We filtrated the effect of the trend from the original series and now display the seasonal component without the trends in the original series.

Forecasting with decomposition

We use the decomposition model to produce forecasts for future time points.

If we used the additive model, our forecasting model becomes

$$\hat{Y}_t = \hat{T}_t + \hat{S}_t + \hat{R}_t$$

or if we used multiplicative mode, we have

$$\hat{Y}_t = \hat{T}_t \cdot \hat{S}_t \cdot \hat{R}_t.$$

We can assume that the seasonal component is unchanging, or changing extremely slowly.

So we will forecast by simply taking the last year of the estimated component.

In other words, a seasonal naïve method is used for the seasonal component.

A seasonal naïve method is implemented by `naive()` function.

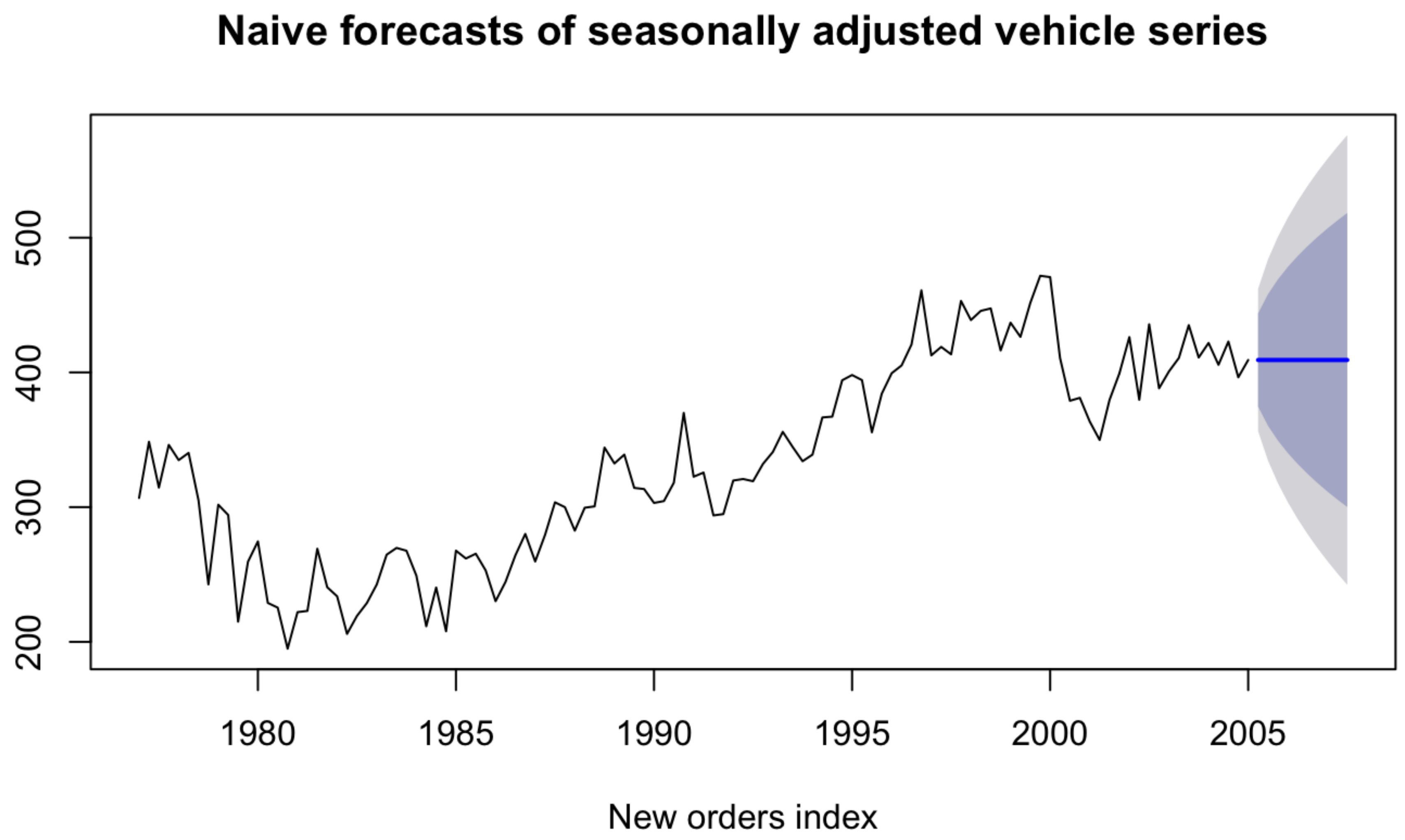
We can first produce forecasts for the seasonally adjusted series.

One of the forecasting methods covered in this course can be applied at this stage.

Then we will *reseasonalize* by adding in the seasonal naïve forecasts of the seasonal component (Hyndman and Athanasopoulos, 2014).

Let’s apply this over the vehicles series. The next visualisation shows the naïve forecasts of the seasonally adjusted vehicle series with a random walk with drift model.

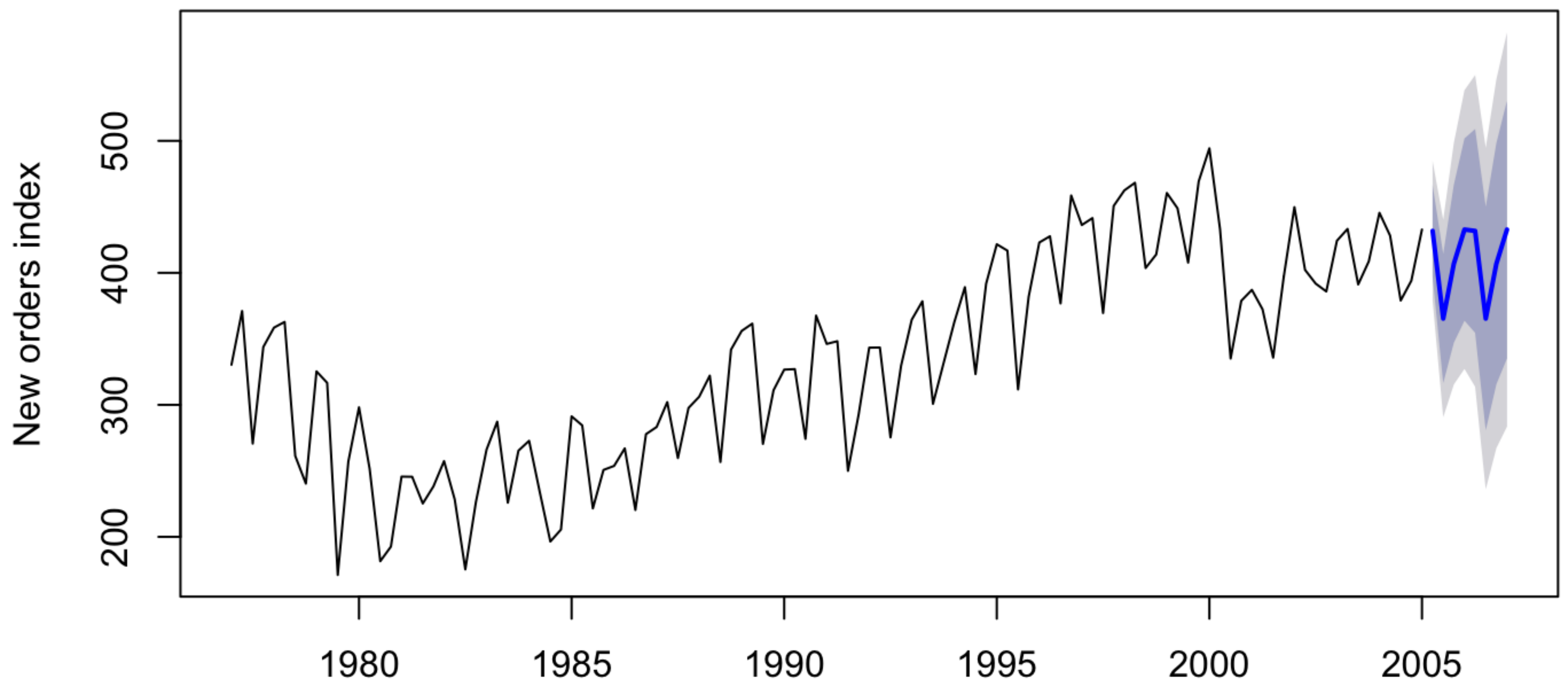
```
vehicles.seasonally.adjusted = seasadj(fit.vehicles)
plot(naive(vehicles.seasonally.adjusted), xlab="New orders index",
  main="Naive forecasts of seasonally adjusted vehicle series")
```


Naive forecasts of seasonally adjusted vehicle series
200
300
400
500
1980
1985
1990
1995
2000
2005
New orders index

The shaded areas in this plot show 80% and 95% confidence limits. The following plot shows the forecasts.

```
forecasts <- forecast(fit.vehicles, method="naive")
plot(forecasts, ylab="New orders index")
```

Forecasts from STL + Random walk

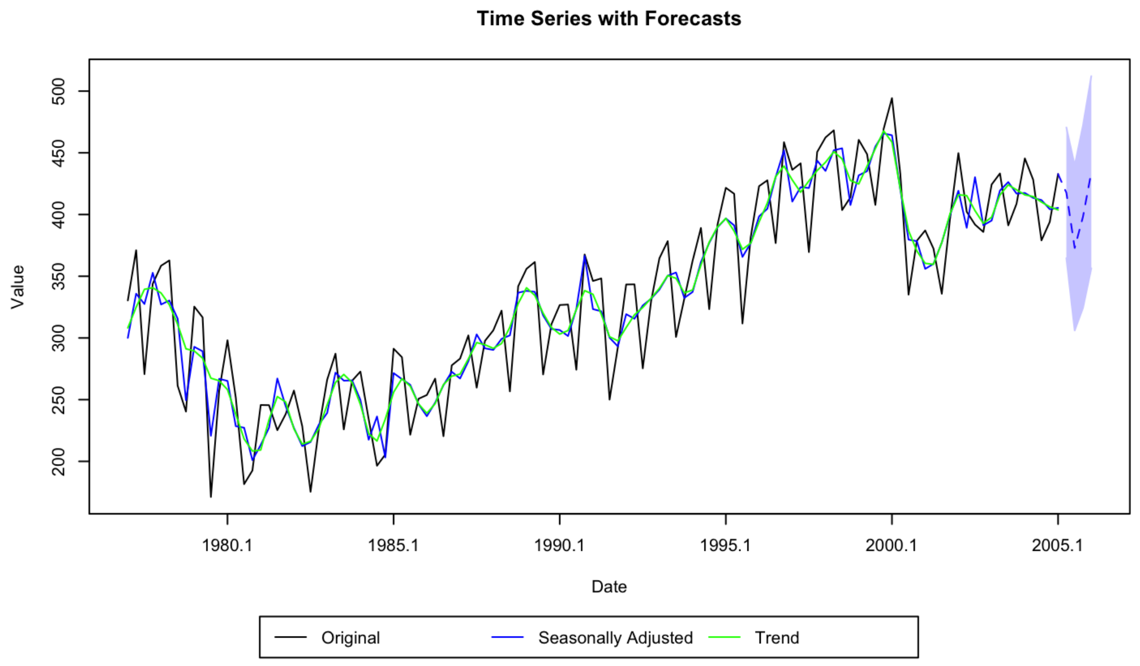


Notice that the uncertainty in the forecasts of the seasonal component has been ignored here as the uncertainty in the seasonal component is much smaller than that for the seasonally adjusted data.

We can use X-12 decomposition to get forecasts as well.

Forecasts and the decomposition as well for the passenger motor vehicle series are displayed by the following code chunk.

```
fit.x12 = x12(ukcars)
plot(fit.x12 , sa=TRUE , trend=TRUE , forecast = TRUE)
```



Practical Application

In this application, we will focus on Land-Ocean Temperature Index (LOTI) to analyse anomalies in land and ocean temperature.

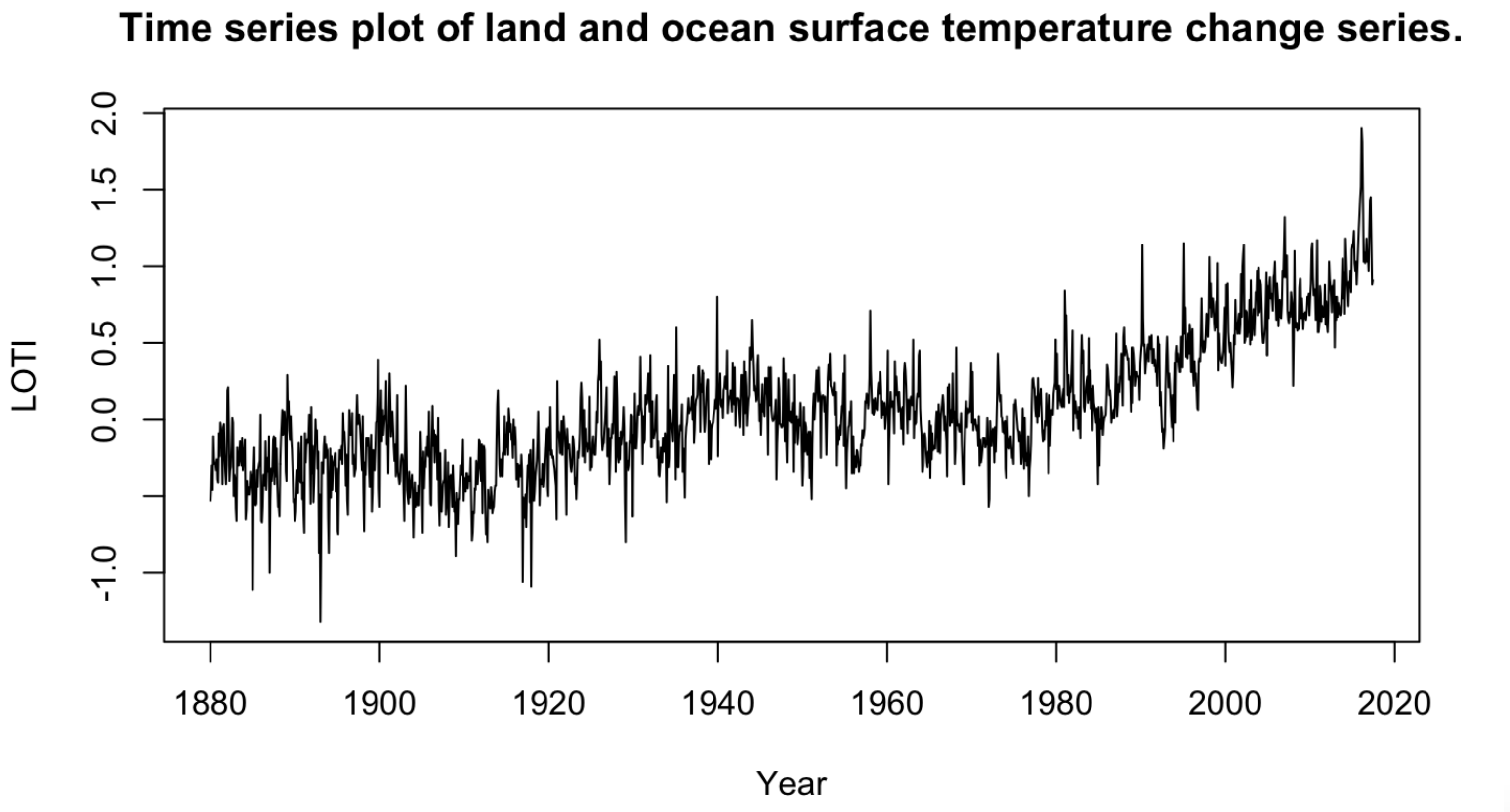
LOTI provides a realistic representation of the global mean trends.

Our main goal is to forecast global surface temperature change accurately.

The data series comes from NASA, National Aeronautics and Space Administration, Goddard Institute for Space Studies.

We will start our analysis by reading the data into R and converting it to a `ts` object. And then the first thing to do is to display the time series plot.

```
landSeaTemp <- read.csv("~/Documents/MATH1307_Forecasting/presentations/Module 2/landSeaTemp.csv",header = FALSE
landSeaTemp = ts(as.vector(t(as.matrix(landSeaTemp))),
                 start = c(1880,1), end = c(2017,6), frequency = 12)
plot(landSeaTemp,ylab='LOTI',xlab='Year', main = "Time series plot of land and ocean surface temperature change
```



Before 1980, the trend is not that significant. But after 1980, there is an obvious increasing trend in the series.

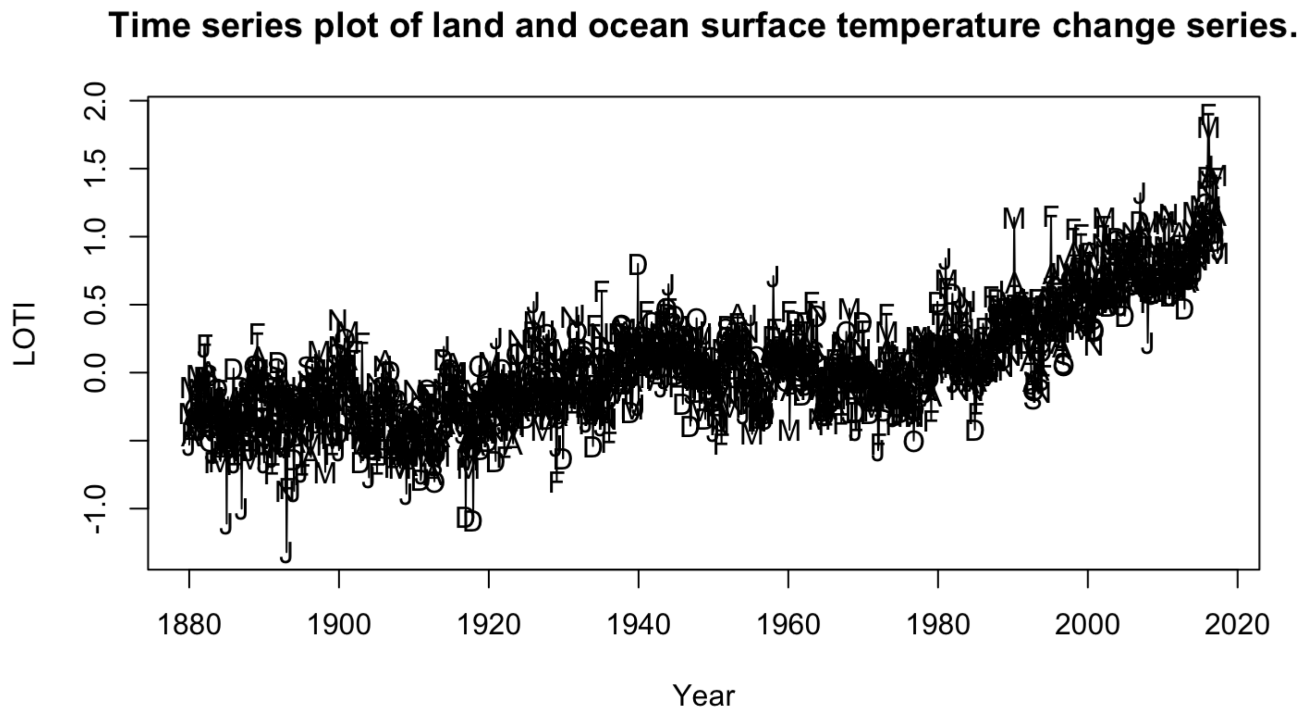
There are some regions with a relatively greater variance but it is hard to claim a significantly changing variance in this series.

The effect of seasonality is not apparent. And there is no obvious intervention effect seen from this time series plot.

But we need to keep in mind that the confounding the effect of seasonality and trend would conceal the existence of intervention effects or other trends in the series.

Let’s put the label on the time series plot to take a better look for the seasonality.

```r
plot(landSeaTemp,ylab='LOTI',xlab='Year', main = "Time series plot of land and ocean surface temperature change
points(y=landSeaTemp,x=time(landSeaTemp), pch=as.vector(season(landSeaTemp)))
```

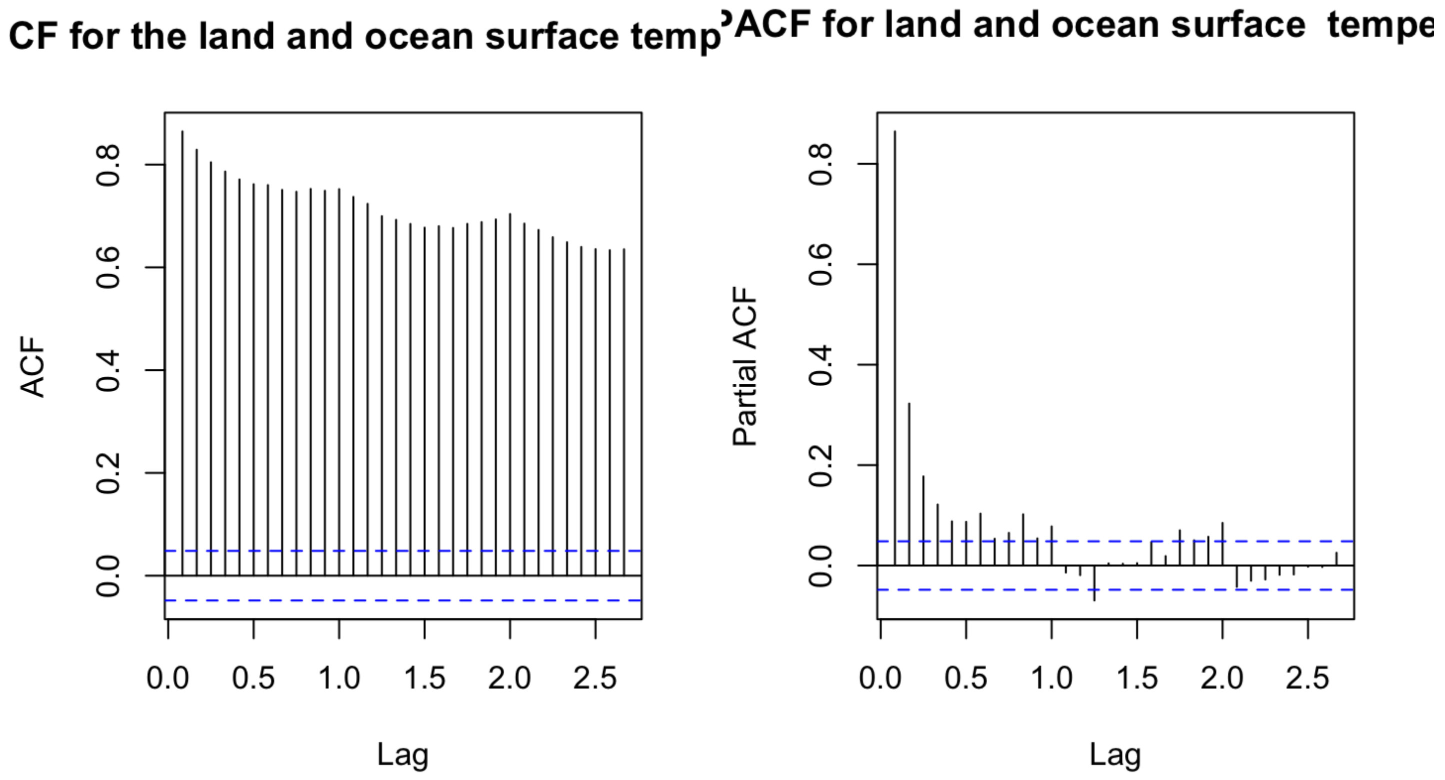


From the time series plot with labels, we observe that high values are mostly coming from summer months and lower values are recorded in winter months as expected.

After 1980, March appears to give some of the highest values. This would be due to having longer summer periods as a result of global warming. Also, a couple of Novembers seen in that period due to having longer winters.

Let’s display sample ACF and PACF to see the structure of the serial correlation in the series.

```r
par(mfrow=c(1,2))
acf(landSeaTemp,max.lag = 48, main="Sample ACF for the land and ocean surface temperature series")
pacf(landSeaTemp,max.lag = 48, main="Sample PACF for land and ocean surface  temperature series")
```



Because we have a slowly decreasing pattern in ACF and a very high first autocorrelation in PACF, we conclude that the series is nonstationary.

Wave pattern at the top of ACF values implies the existence of seasonality.

So, first, we need to deal with nonstationarity and then seasonality to display lean serial correlation in the series.

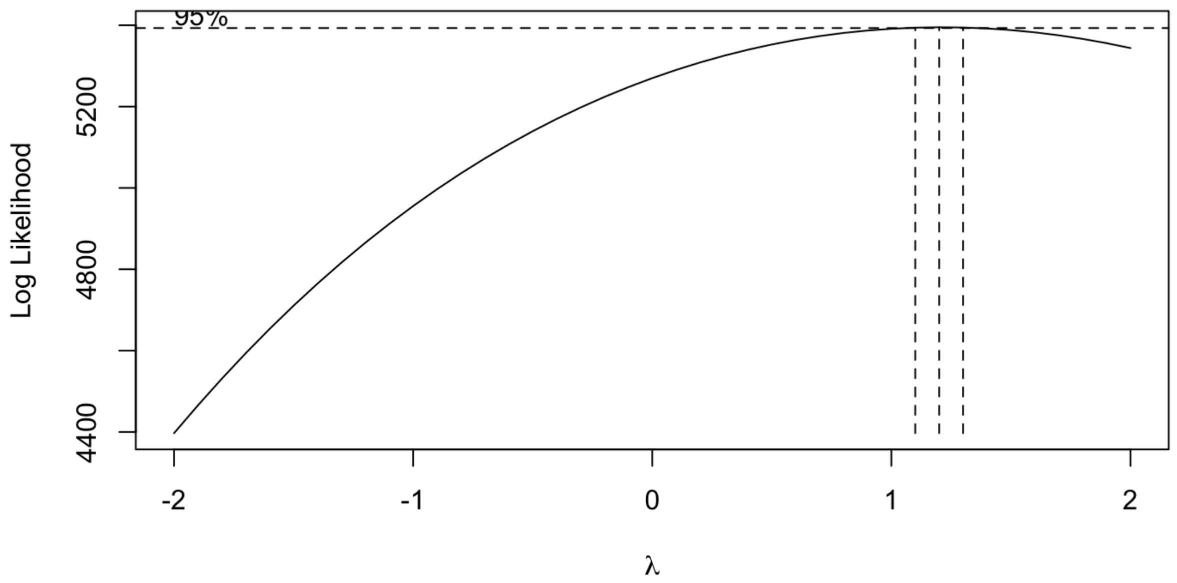
First, we will apply the Box-Cox transformation and see if it helps to deal with nonstationarity.

We have two options for the application of the Box-Cox transformation. We can use `Box.Cox.lambda()` function from the `forecast` package or `BoxCox.ar()` function from the `TSA` package.

```r
lambda = BoxCox.lambda(landSeaTemp)
lambda
```

```
## [1] 0.9843569
```

```r
BC = BoxCox.ar(landSeaTemp+2)
```



```r
BC$ci
```

```
## [1] 1.1 1.3
```

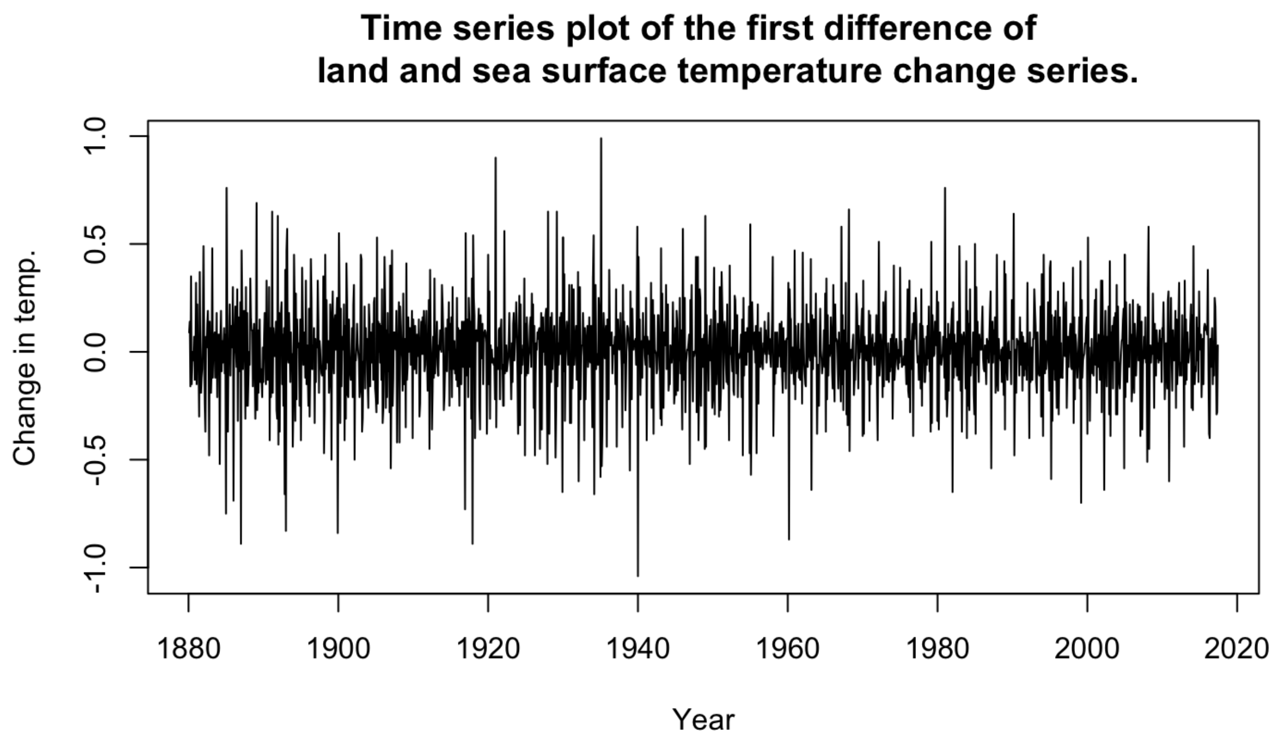
So, we get $\lambda = 0.98$ from the `Box.Cox.lambda()` function and $\lambda = (1.1 + 1.3)/2 = 1.2$ from the `BoxCox.ar()` function.

These are very close to each other and 1 as well. So, we will go on with the original series without any transformation.

Then, we will take the first difference an see if the series gets stationary.

```
landSeaTemp.diff = diff(landSeaTemp)
```

```r
plot(landSeaTemp.diff,ylab='Change in temp.',xlab='Year', main = "Time series plot of the first difference of
      land and sea surface temperature change series.")
```



Now, the series shows the pattern of stationary time series. It would be good to support this conclusion with a unit root test.

```
adf.test = adf.test(landSeaTemp.diff)
adf.test
```

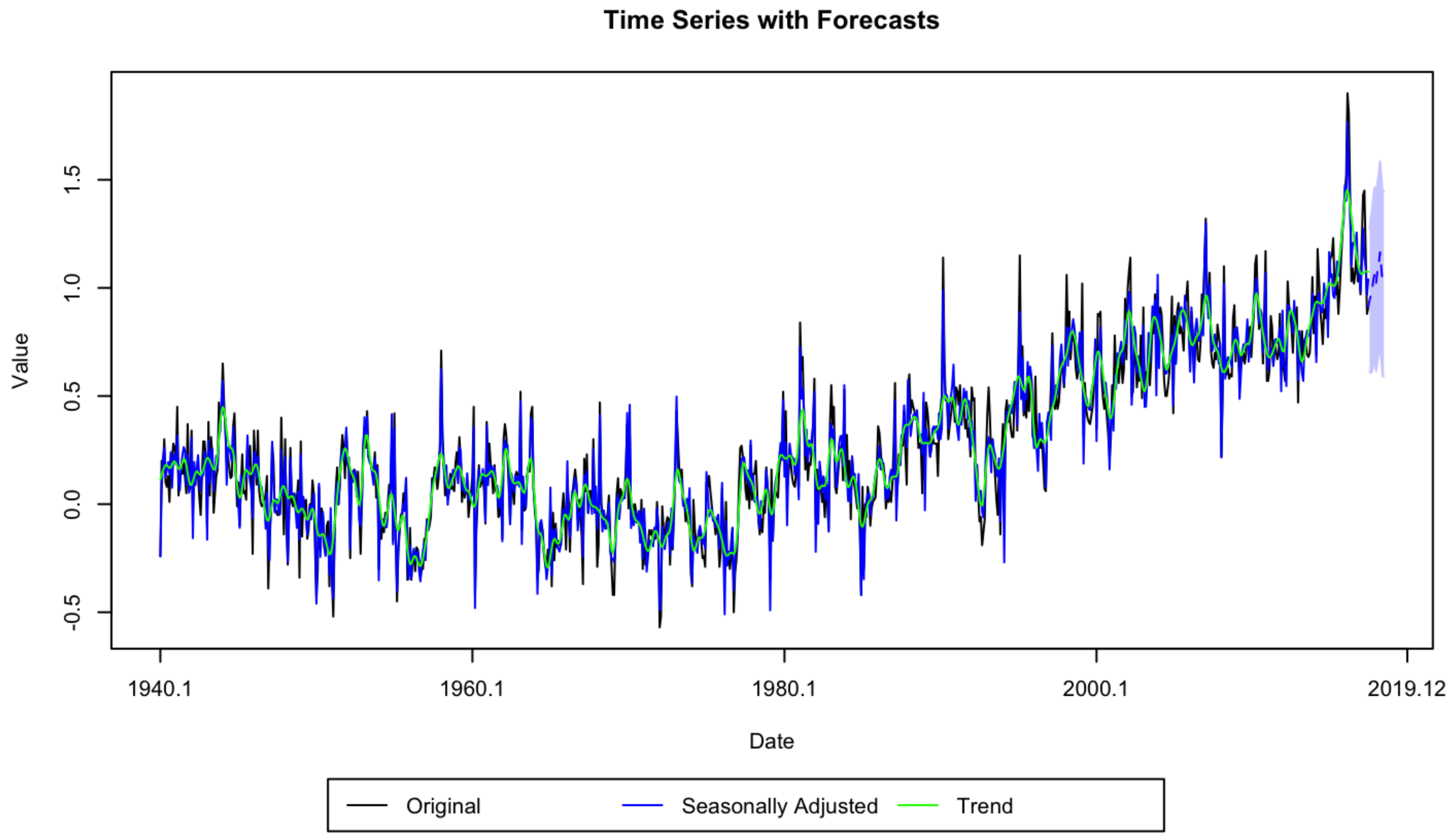
```
##
##  Augmented Dickey-Fuller Test
##
## data:  landSeaTemp.diff
## Dickey-Fuller = -17.958, Lag order = 11, p-value = 0.01
## alternative hypothesis: stationary
```

Because we got a p-value less thant 0.05, we conclude that the differenced series is stationary at 5% level of significance.

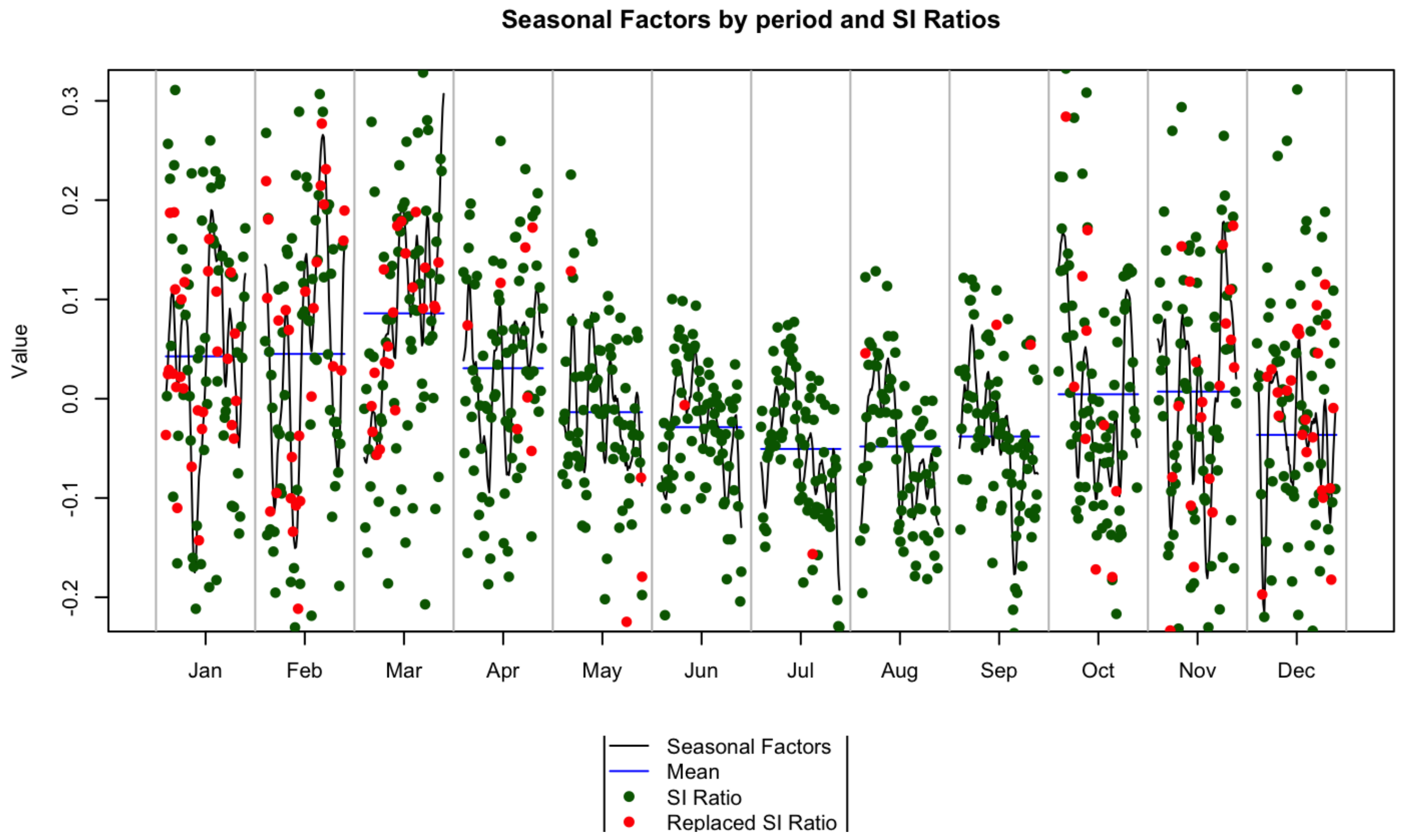
Now we will apply decomposition methods to separate the effect of trend and seasonality. This will let us infer the patterns of trend and seasonality separately. The X12 decomposistion is applied by the following code chunk:

```r
landSeaTemp.window = window(landSeaTemp,start = c(1940,1))
# Notice that x12() function has a limitation on the number of
# data points we can send to it. So, I'm subsetting the data
# before calling the x12() function
landSeaTemp.decom.x12 = x12(landSeaTemp.window)
```

```
plot(landSeaTemp.decom.x12 , sa=TRUE , trend=TRUE , forecast = TRUE)
```



```
plotSeasFac(landSeaTemp.decom.x12)
```



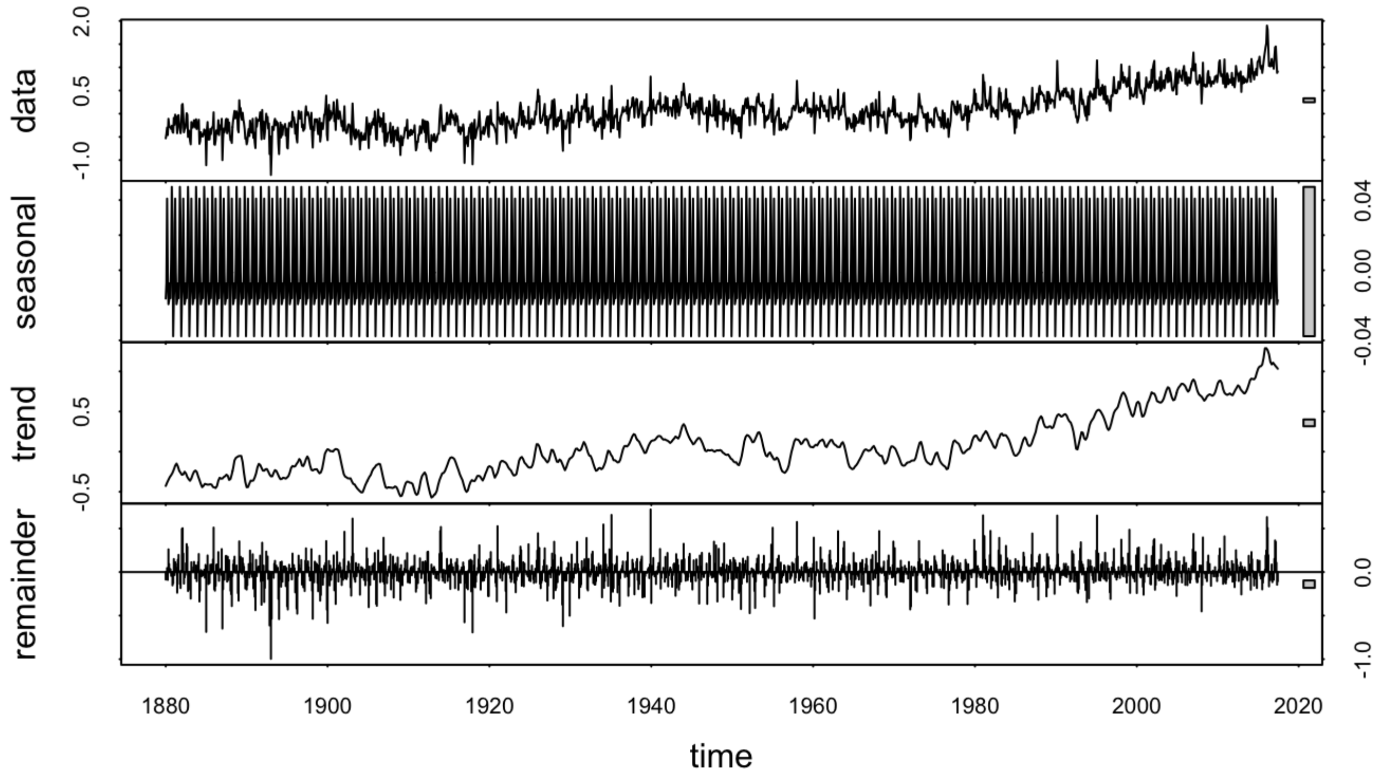
When adjusted for seasonality, we observe that the effect of large fluctuations after 1980 is not that much and those large fluctuations in land and sea temperature are due to the seasonality effect.

However, the downward fluctuations are not directly related to the seasonal effect.

We have lots of extreme values in the months of January - April and October - December.

We will also apply the STL decomposition. The following code chunk applies the STL decomposition and displays the results.

```r
landSeaTemp.decom <- stl(landSeaTemp, t.window=15,
                         s.window="periodic", robust=TRUE)
plot(landSeaTemp.decom)
```

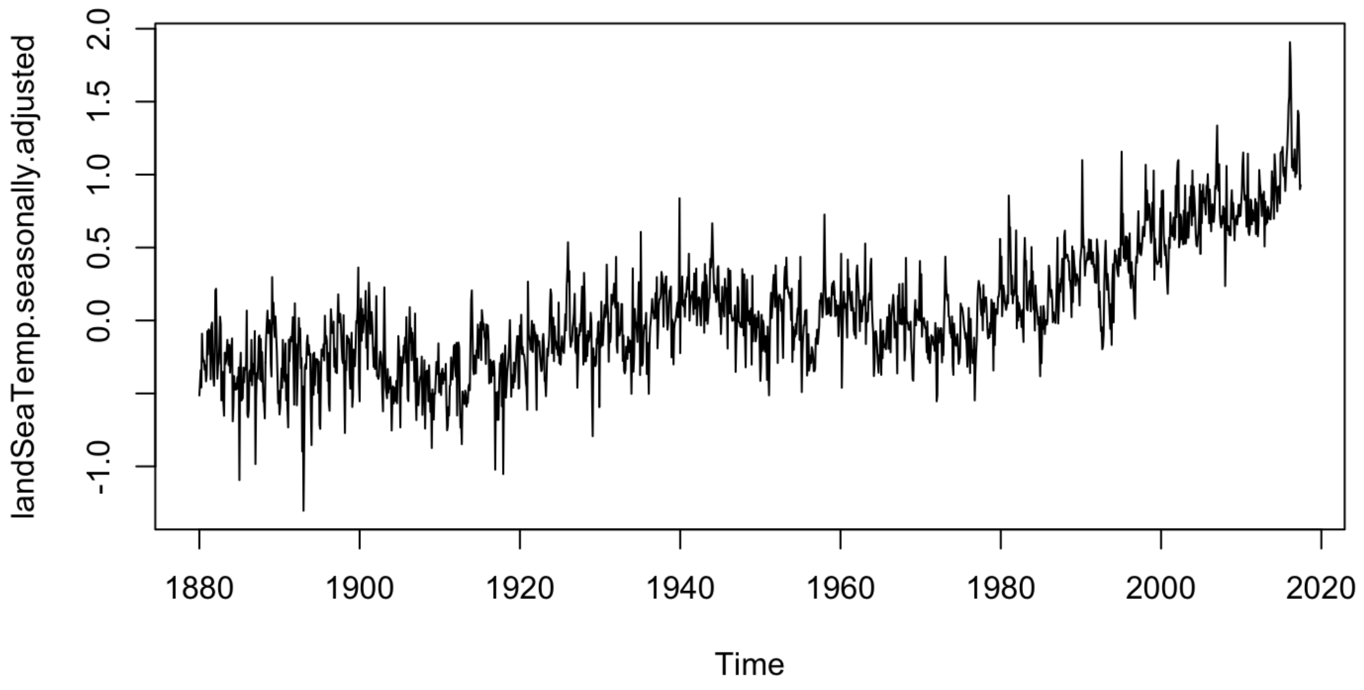


When the effect of seasonality is filtrated out, the trend component becomes more apparent.

In the remainder part, there are some significant fluctuations and changing variance is present in the land and sea temperature series.

The next visualisation displays the sesonally adjusted land and sea temperatures series.

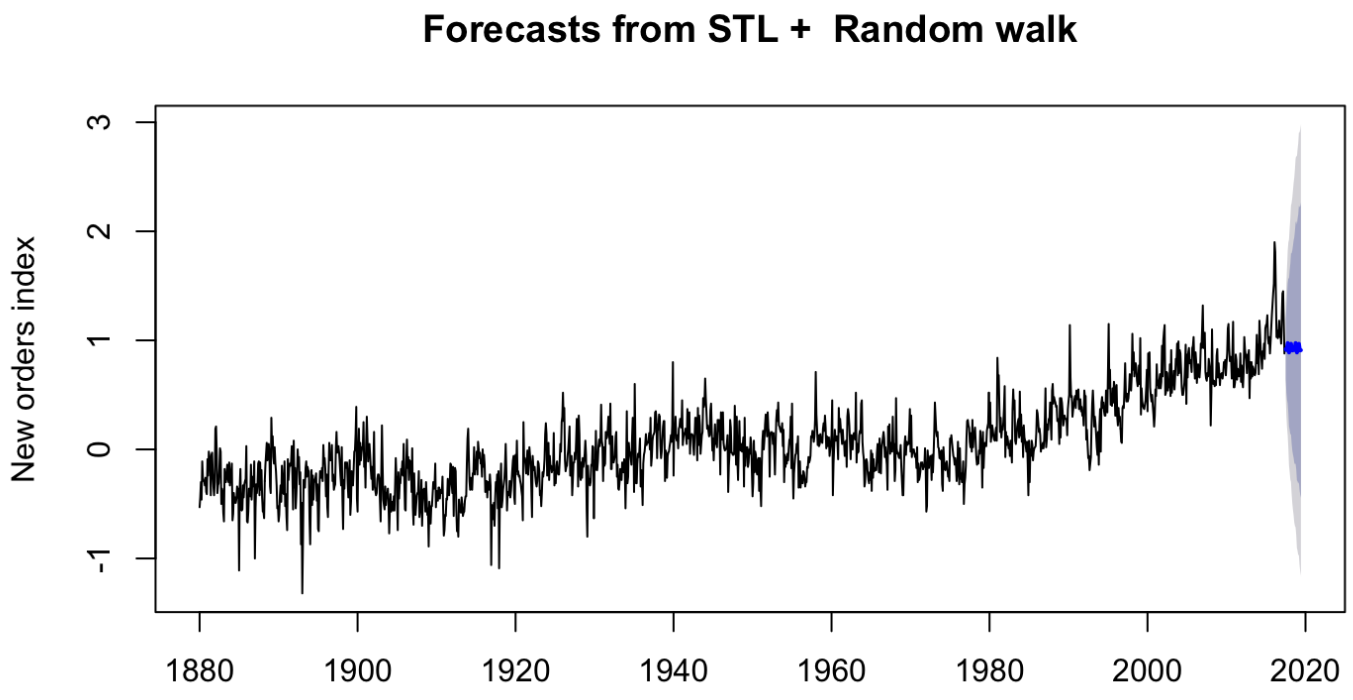
```
landSeaTemp.seasonally.adjusted = seasadj(landSeaTemp.decom)
plot(landSeaTemp.seasonally.adjusted)
```



Large fluctuations and changing variance are more visible in the seasonally adjusted series.

We will display forecasts from STL methods.

```
forecasts = forecast(landSeaTemp.decom, method="naive")
plot(forecasts, ylab="New orders index")
```



Naive forecasts represent the mean level for future land and sea temperatures.

Summary

In this module, we studied basic operations including lag, differencing, and Box-Cox transformations.

Then, we covered decomposing a time series data into this trend and seasonal components.

By this, we get detailed information about the nature of trend and seasonality in the data.

One point that needs to be remembered is that we should not use decomposition techniques to evaluate the existence of these components.

Once we decide that there are these components, we can use decomposition techniques to visualise and get more detailed information about the components.

It is also possible to get future forecasts using decomposition models.

What’s next?

In the next module, we will focus on

- distributed lag models
  - polynomial distributed lags,
  - the geometric lag model,
  - the Koyck transformation, and
  - autoregressive distributed lag model
- forecasting with distributed lag models/

References

Cleveland R.B., Cleveland, W.S., McRae, J.E., Terpenning I. (1990). STL: Seasonal - Trend decomposition based on Losses. *Journal of Official Statistics*, 6, 1, 3-73.

Cryer, J.D., Chan, K-S. (2008). *Time Series Analysis with Applications in R*. Springer.

Hyndman, R.J. Athanasopoulos, G. (2014). *Forecasting: Principles and Practice*. OTEXTS.

Thanks for your attendance! Please use Socrative.com with room *FORECASTINGPG* to give feedback!