6 - Time Series Decomposition

Rob Hyndman (with Deppa edits/additions)

March 4th, 2019

Table of Contents

[6.1 Time Series Components 2](#_Toc2587857)

[Example 6.1 - Electrical Equipment Manufacturing (Euro Area) 2](#_Toc2587858)

[Seasonally Adjusted Time Series 6](#_Toc2587859)

[Example 6.1 - Electrical Equipment Manufacturing (cont’d) 6](#_Toc2587860)

[6.2 - Moving Averages (see earlier handout - won’t use these much going forward) 8](#_Toc2587861)

[6.3 - Classical Decomposition 9](#_Toc2587862)

[Additive Decomposition 9](#_Toc2587863)

[Step 1 9](#_Toc2587864)

[Step 2 9](#_Toc2587865)

[Step 3 9](#_Toc2587866)

[Step 4 9](#_Toc2587867)

[Multiplicative Decomposition 9](#_Toc2587868)

[Step 1 9](#_Toc2587869)

[Step 2 9](#_Toc2587870)

[Step 3 9](#_Toc2587871)

[Step 4 10](#_Toc2587872)

[Example 6.2 - Monthly U.S. Liquor Sales 10](#_Toc2587873)

[Comments on Classical Decomposition 16](#_Toc2587874)

[6.4 - X11 Decomposition 16](#_Toc2587875)

[Example 6.1 - Electrical Equipment Manufacturing Index (cont’d) 17](#_Toc2587876)

[Example 6.2 - Monthly U.S. Liquor Sales (cont’d) 24](#_Toc2587877)

[6.5 - SEATS Decompostion 26](#_Toc2587878)

[(X-13 Seasonal Extraction in ARIMA Time Series) 26](#_Toc2587879)

[Example 6.1 - Electrical Equipment Manufacturing Index (cont’d) 27](#_Toc2587880)

[Example 6.2 - Monthly U.S. Liquor Sales (cont’d) 28](#_Toc2587881)

[6.6 - STL Decomposition 29](#_Toc2587882)

[Example 6.2 - Monthly U.S. Liquor Sales (cont’d) 32](#_Toc2587883)

[6.7 - Forecasting with Decomposition 33](#_Toc2587884)

[Example 6.1 - Electrical Equipment Manufacturing Index (cont’d) 33](#_Toc2587885)

[Example 6.2 - Monthly U.S. Liquor Sales (cont’d) 38](#_Toc2587886)

# 6.1 Time Series Components

If we assume an additive model, then we can write

where is the data at time period , is the seasonal component at time , is the trend-cycle component at time , and is the remainder component at time . Sometimes these components are multiplicative and thus the additive model above becomes,

where , , and are defined as above.

The **additive model** is the most appropriate if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle, does not vary with the level of the time series. When the variation in the seasonal pattern, or the variation around the trend-cycle, appears to be proportional to the level of the time series, then a **multiplicative model** is more appropriate. Multiplicative models are common with economic time series.

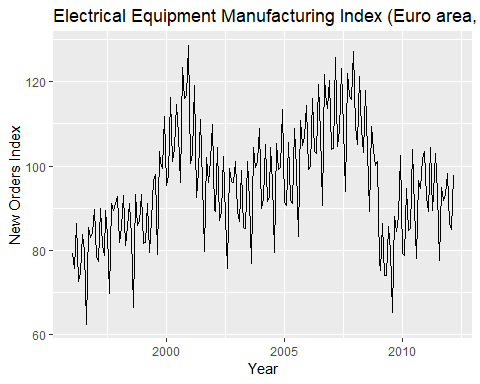
An alternative to using a multiplicative model is to first transform the data until the variation in the series appears to be stable over time, then use an additive model. When a log transformation has been used, this is equivalent to using a multiplicative decomposition because,

### Example 6.1 - Electrical Equipment Manufacturing (Euro Area)

From the elecequip help file:  
Monthly manufacture of electrical equipment: computer, electronic and optical products. January 1996 - March 2012. Data adjusted by working days; Euro area (17 countries). Industry new orders index, 2005=100.

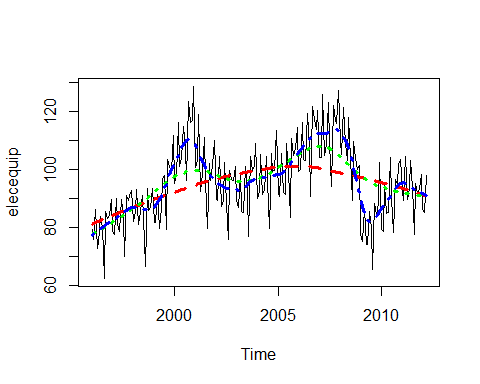
What does mean? It means that all values have been divided by the original mean of the electrical equipment time series value for 2005 and then multiplied by 100. Thus the mean for 2005 in the indexed scale is 100. The indexed value at time is given by .

require(fpp2)  
autoplot(elecequip) + xlab("Year") + ylab("New Orders Index") + ggtitle("Electrical Equipment Manufacturing Index (Euro area, 2005 = 100)")



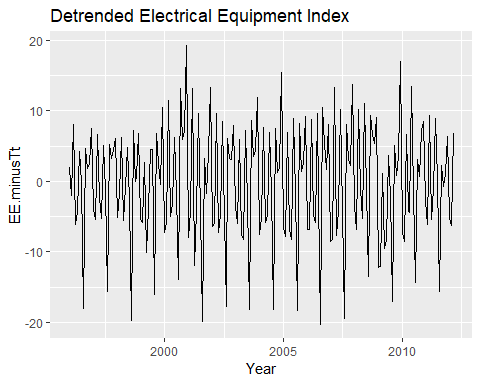
We will now try to use different tools to decompose this series into the three components . In order to estimate the long-term/cyclic trends in the time series we could use tools such as regression, moving averages, and local regression fitting. We will use local weighted regression smoothing (lowess).

low1 = lowess(elecequip,f=2/3)  
plot(elecequip)  
lines(low1,lty=2,lwd=3,col="red")  
low2 = lowess(elecequip,f=1/3)  
lines(low2,lty=3,lwd=3,col="green")  
low3 = lowess(elecequip,f=.1)  
lines(low3,lty=4,lwd=3,col="blue")



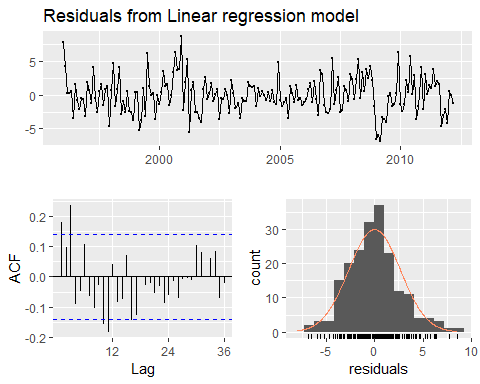
Which lowess smoother seems to capture the long-term/cyclical trend the best? We will assume that the lowess smooth with f=.10 is a satisfactory estimate of .

Tt = low3$y  
# Subtract the Tt from the original series leaving behind St+Rt  
EE.minusTt = elecequip - Tt  
autoplot(EE.minusTt) + xlab("Year") + ggtitle("Detrended Electrical Equipment Index")



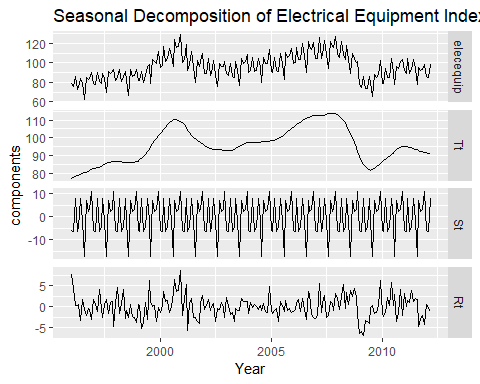
We will now use regression with seasonal dummy variables (i.e. monthly dummy variables) to extract the seasonal trend . We can then find by subtracting the seasonal fit, , from the detrended time series (EE.minusTt).

seasonfit = tslm(EE.minusTt~season)  
St = fitted(seasonfit)  
Rt = EE.minusTt - St  
# Rt = residuals(seasonfit) is the same result  
checkresiduals(seasonfit)



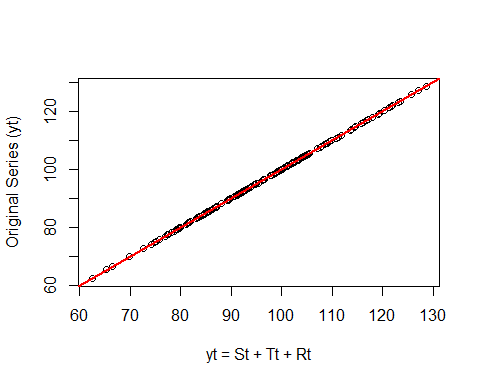
##   
## Breusch-Godfrey test for serial correlation of order up to 24  
##   
## data: Residuals from Linear regression model  
## LM test = 42.449, df = 24, p-value = 0.01149

components = cbind(elecequip,Tt,St,Rt)  
autoplot(components,facet=T) + xlab("Year") + ggtitle("Seasonal Decomposition of Electrical Equipment Index")



This was a pretty tedious process, but it demonstrates the idea behind additive decomposition of a time series . We will check this below.

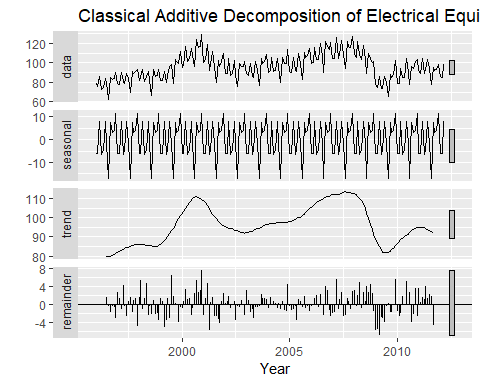
yt = St + Tt + Rt  
plot(yt,elecequip,xlab="yt = St + Tt + Rt",ylab="Original Series (yt)")  
abline(0,1,lwd=2,col="red")



We can see the match perfectly.

There must be an easier way to do this, and there is. The function decompose is the fpp2/forecast library does this automatically.

ee.decomp = decompose(elecequip)  
autoplot(ee.decomp) + xlab("Year") + ggtitle("Classical Additive Decomposition of Electrical Equipment Index")



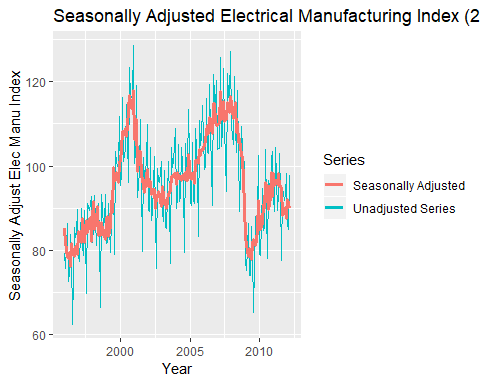
## Seasonally Adjusted Time Series

If the seasonal component () is removed from the original data, the resulting values are called the “seasonally adjusted” data. For an additive model, the seasonally adjusted data are given by , and for multiplicative data, the seasonally adjusted values are obtained using .

### Example 6.1 - Electrical Equipment Manufacturing (cont’d)

In the example below we will deseasonalize or seasonally adjust the electrical manufacturing index using our results from above.

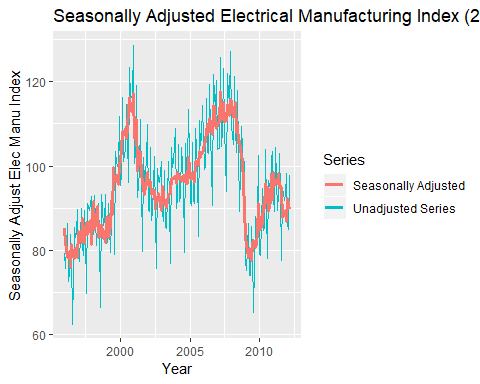
seasonfit = tslm(EE.minusTt~season)  
St = fitted(seasonfit)  
# Subtract St from yt to obtain seasonally adjusted time series.  
elec.seasadj = elecequip - St  
autoplot(elecequip) + xlab("Year") + ylab("Seasonally Adjust Elec Manu Index") + ggtitle("Seasonally Adjusted Electrical Manufacturing Index (2005 = 100)") +  
 autolayer(elecequip,series="Unadjusted Series") +  
 autolayer(elec.seasadj,lwd=1.2,series="Seasonally Adjusted") +  
 guides(colour=guide\_legend(title="Series"))



# We can also extract the seasonal component (St) from the results of decompose().  
attributes(ee.decomp)

## $names  
## [1] "x" "seasonal" "trend" "random" "figure" "type"   
##   
## $class  
## [1] "decomposed.ts"

St.decompose = ee.decomp$seasonal  
# Subtract St from yt to obtain seasonally adjusted time series.  
elec.seasadj2 = elecequip - St.decompose  
autoplot(elecequip) + xlab("Year") + ylab("Seasonally Adjust Elec Manu Index") + ggtitle("Seasonally Adjusted Electrical Manufacturing Index (2005 = 100)") +  
 autolayer(elecequip,series="Unadjusted Series") +  
 autolayer(elec.seasadj2,lwd=1.2,series="Seasonally Adjusted") +  
 guides(colour=guide\_legend(title="Series"))



If the variation due to seasonality is not of primary interest, the seasonally adjusted series can be useful. For example, monthly unemployment data are usually seasonally adjusted in order to highlight variation due to the underlying state of the economy rather than the seasonal variation. An increase in unemployment due to school leavers seeking work is seasonal variation, while an increase in unemployment due to large employers laying off workers is non-seasonal. Most people who study unemployment data are more interested in the non-seasonal variation. Consequently, employment data (and many other economic series) are usually seasonally adjusted.

Seasonally adjusted series contain the remainder component as well as the trend-cycle. Therefore, they are not “smooth”, and “downturns” or “upturns” can be misleading. If the purpose is to look for turning points in the series, and interpret any changes in the series, then it is better to use the trend-cycle component () rather than the seasonally adjusted data.

# 6.2 - Moving Averages (see earlier handout - won’t use these much going forward)

Read section 6.2 in Forecasting Principles and Practice (<https://otexts.org/fpp2/moving-averages.html>).

# 6.3 - Classical Decomposition

The classical decomposition method originated in the 1920s. It is a relatively simple procedure, and forms the starting point for most other methods of time series decomposition. There are two forms of classical decomposition: an additive decomposition and a multiplicative decomposition. These are described below for a time series with seasonal period (e.g. for quarterly data, for monthly data, and for weekly data, for daily data with a weekly pattern).

In classical decomposition, we assume that the seasonal component is constant from year to year. For multiplicative seasonality, the values that form the seasonal component are sometimes called the “seasonal indices”.

## Additive Decomposition

The above example illustrates the basics of additive decomposition. The function decompose achieves the decomposition using a slight different algorithm, which is outlined below.

### Step 1

If is an even number, compute the trend-cycle component using a -MA to obtain . If is an odd number, compute the trend-cycle component using an -MA to obtain .

### Step 2

Calculate the detrended series .

### Step 3

To estimate the seasonal component for each season, simply average the detrended values for that season. For example, with monthly data, the seasonal component for March is the average of all the detrended March values in the data. These seasonal component values are then adjusted to ensure that they add to zero. The seasonal component is obtained by stringing together these values for each year of data. This gives .

### Step 4

The remainder component is calculated by subtracting the estimated seasonal and trend-cycle components from the original time series (), i.e. .

## Multiplicative Decomposition

A classical multiplicative decomposition is very similar, except that the subtractions are replaced by divisions.

### Step 1

If is an even number, compute the trend-cycle component using a -MA to obtain . If is an odd number, compute the trend-cycle component using an -MA to obtain .

### Step 2

Calculate the detrended series: .

### Step 3

To estimate the seasonal component for each season, simply average the detrended values for that season. For example, with monthly data, the seasonal index for March is the average of all the detrended March values in the data. These seasonal indexes are then adjusted to ensure that they add to . The seasonal component is obtained by stringing together all of the seasonal indices for each year of data. This gives .

### Step 4

The remainder component is calculated by dividing out the estimated seasonal and trend-cycle components:

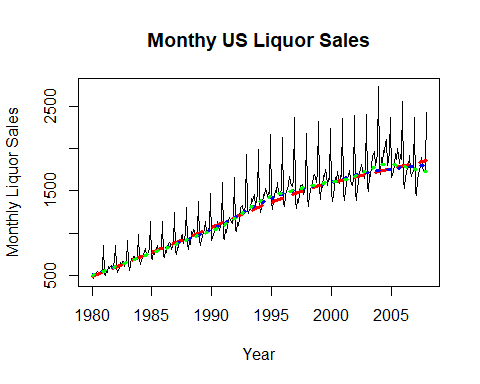
### Example 6.2 - Monthly U.S. Liquor Sales

As an example of multiplicative decomposition consider the monthly liquor sales in the US.

Liquor = read.csv(file="http://course1.winona.edu/bdeppa/FIN%20335/Datasets/US%20Liquor%20Sales.csv")  
names(Liquor)

## [1] "Time" "Month" "Year" "Liquor.Sales"

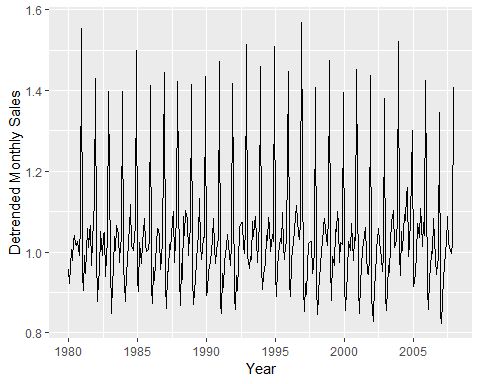
LiqSales = ts(Liquor$Liquor.Sales,start=1980,frequency=12)  
# First we will use lowess to fit the long-term/cyclic trend (Tt)  
plot(LiqSales,xlab="Year",ylab="Monthly Liquor Sales",main="Monthy US Liquor Sales")  
low1 = lowess(LiqSales,f=2/3)  
low2 = lowess(LiqSales,f=1/3)  
low3 = lowess(LiqSales,f=.20)  
lines(low1,lty=2,lwd=3,col="red")  
lines(low2,lty=3,lwd=3,col="blue")  
lines(low3,lty=3,lwd=3,col="green")



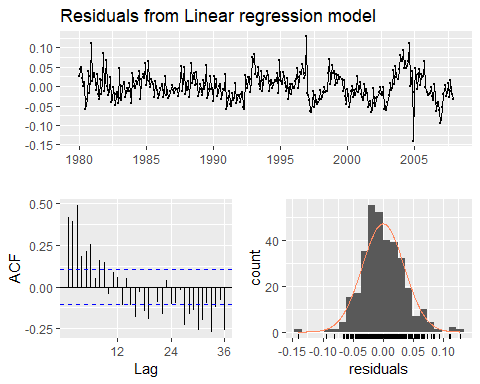
# The last lowess smooth in green appears to capture the long-term/cyclic trend the best  
  
Tt = low3$y

We now divide the original series by and then extract the seasonal trend .

LS.detrend = LiqSales/Tt  
autoplot(LS.detrend) + xlab("Year") + ylab("Detrended Monthly Sales")

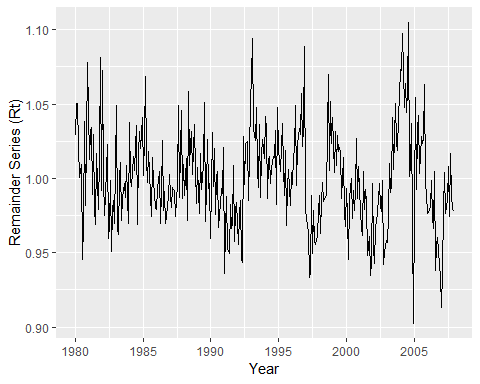


fit = tslm(LS.detrend~season)  
checkresiduals(fit)

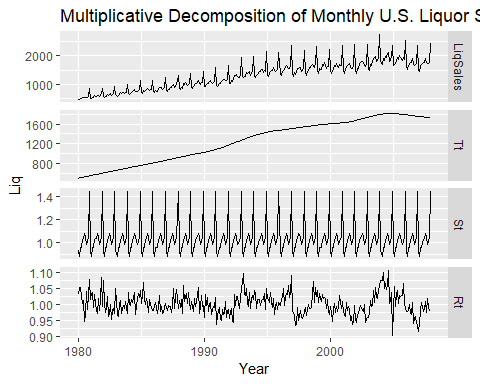


##   
## Breusch-Godfrey test for serial correlation of order up to 24  
##   
## data: Residuals from Linear regression model  
## LM test = 143.69, df = 24, p-value < 2.2e-16

# Residuals still show some time structure, i.e. they are NOT white noise.  
St = fitted(fit)  
Rt = LiqSales/(Tt\*St)  
autoplot(Rt) + xlab("Year") + ylab("Remainder Series (Rt)")

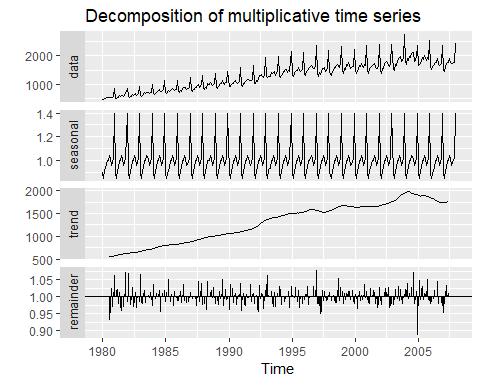


Liq = cbind(LiqSales,Tt,St,Rt)  
autoplot(Liq,facet=T) + xlab("Year") + ggtitle("Multiplicative Decomposition of Monthly U.S. Liquor Sales")



This process is made considerably easier by again using the decompose function with argument type="multiplicative".

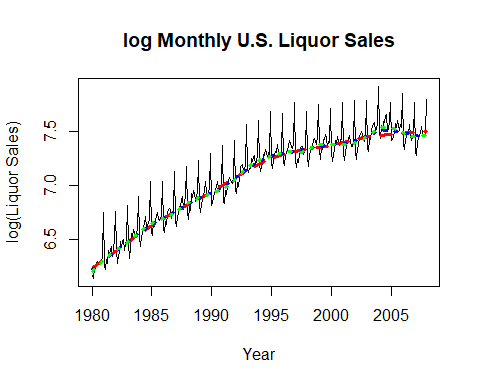
LS.decomp = decompose(LiqSales,type="multiplicative")  
autoplot(LS.decomp)



Comparison of the results from decompose with those we did by applying the steps reveals very similar results with exception of the long-term trend estimate . The estimate obtained from decompose is considerably “wigglier” than our estimate. We could have obtained a more flexible estimate of using lowess by choosing a smaller value for the fraction of observations in each local fitting window. Here we would have to choose a value for smaller than the smallest value we used which was .

When considering these data in earlier chapters one of the approaches we took was to log-transform the monthly liquor sales. This had the effect of stabilizing the seasonal swings in this time series, thereby making an additive model more appropriate. Below we will take this approach by first doing the step-by-step decomposition ourselves and by using the decompose function applied to the with type="additive".

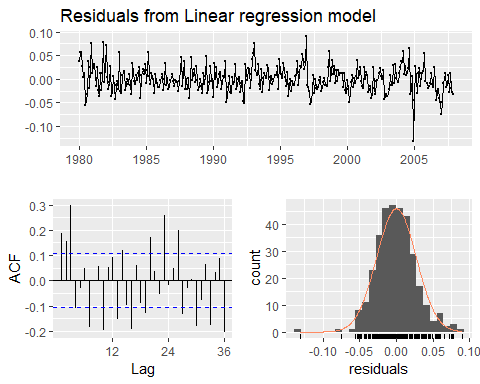
logLS = log(LiqSales)  
plot(logLS,xlab="Year",ylab="log(Liquor Sales)",main="log Monthly U.S. Liquor Sales")  
low1 = lowess(logLS,f=1/3)  
low2 = lowess(logLS,f=.20)  
low3 = lowess(logLS,f=.10)  
lines(low1,lty=2,lwd=3,col="red")  
lines(low2,lty=3,lwd=3,col="blue")  
lines(low3,lty=3,lwd=3,col="green")



# We will use the last lowess smooth (f = .10) to estimate the long-term trend (Tt).  
Tt = low3$y

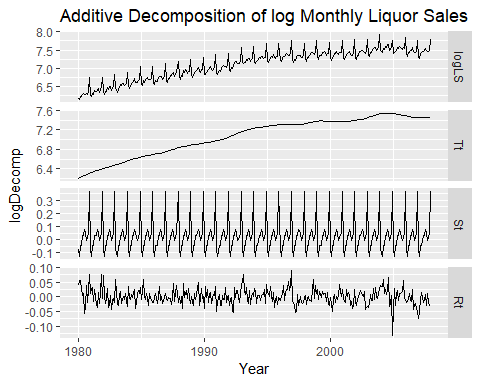
Next we remove the long-term trend and then estimate the seasonal trend assuming an additive approach is appropriate.

logLS.detrend = logLS - Tt  
fit = tslm(logLS.detrend~season)  
St = fitted(fit)  
Rt = logLS - Tt - St  
checkresiduals(fit)



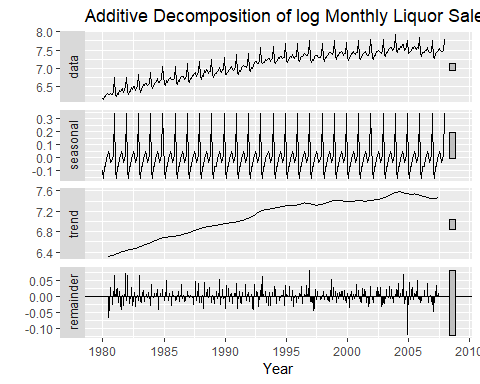
##   
## Breusch-Godfrey test for serial correlation of order up to 24  
##   
## data: Residuals from Linear regression model  
## LM test = 100.87, df = 24, p-value = 2.131e-11

# As with the multiplicative fit our residuals still show some time structure.  
logDecomp = cbind(logLS,Tt,St,Rt)  
autoplot(logDecomp,facet=T) + xlab("Year") + ggtitle("Additive Decomposition of log Monthly Liquor Sales")



We can simplify the above process by using decompose with type="additive" with log-liquor sales.

logLS.decomposition = decompose(logLS,type="additive")  
autoplot(logLS.decomposition) + xlab("Year") + ggtitle("Additive Decomposition of log Monthly Liquor Sales from decompose()")



Again the main difference between the two approaches is the degree of local-fitting, i.e. wigglyness, of the long-term trend estimates. Comparing the results from the decomposition using the decompose function with type="multiplicative" on the untransformed liquor sales to those obtained by using the decompose function with type="additive" on the log-transformed liquor sales we see very little, if any, difference between the two. That is because the multiplicative model is equivalent to the additive model .

## Comments on Classical Decomposition

While classical decomposition is still widely used, it is not recommended, as there are now several much better methods. Some of the problems with classical decomposition are summarized below.

* The estimate of the trend-cycle is unavailable for the first few and last few observations. For example, if , there is no trend-cycle estimate for the first six or the last six observations. Consequently, there is also no estimate of the remainder component for the same time periods.
* The trend-cycle estimate tends to over-smooth rapid rises and falls in the data.
* Classical decomposition methods assume that the seasonal component repeats from year to year. For many series, this is a reasonable assumption, but for some longer series it is not. For example, electricity demand patterns have changed over time as air conditioning has become more widespread. Specifically, in many locations, the seasonal usage pattern from several decades ago had its maximum demand in winter (due to heating), while the current seasonal pattern has its maximum demand in summer (due to air conditioning). The classical decomposition methods are unable to capture these seasonal changes over time.
* Occasionally, the values of the time series in a small number of periods may be particularly unusual. For example, the monthly air passenger traffic may be affected by an industrial dispute, making the traffic during the dispute very different from usual. The classical method is not robust to these kinds of unusual values or shocks in the time series.

# 6.4 - X11 Decomposition

Another popular method for decomposing quarterly and monthly data is the **X11** method which originated in the U.S. Census Bureau and Statistics Canada.

This method is based on classical decomposition, but includes many extra steps and features in order to overcome the drawbacks of classical decomposition that were discussed in the previous section. In particular, trend-cycle estimates are available for all observations including the end points, and the seasonal component is allowed to vary slowly over time. X11 also has some sophisticated methods for handling trading day variation, holiday effects and the effects of known predictors. It handles both additive and multiplicative decomposition. The process is entirely automatic and tends to be highly robust to outliers and level shifts (*shocks*) in the time series.

The details of the X11 method are described in Dagum and Bianconcini (2016). Here we will only demonstrate how to use the automatic procedure in R. Recent advances in the approach have led to X12 and X13 decompositions. We will examine X-13 SEATS decomposition in section 6.5 below.

The X11 method is available as an option using the seas() function from the seasonal package for R. You will need to install the seasonal package in R-Studio and then load it.

We will demonstrate the use of the X11 method with the two time series we worked with in examples 6.1 and 6.2 above.

### Example 6.1 - Electrical Equipment Manufacturing Index (cont’d)

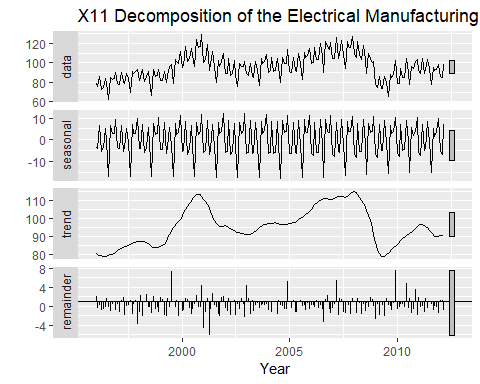
require(seasonal)

## Loading required package: seasonal

ee.x11 = seas(elecequip,transform.function="none",x11="")  
summary(ee.x11)

##   
## Call:  
## seas(x = elecequip, transform.function = "none", x11 = "")  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## Easter[8] -2.08863 0.83316 -2.507 0.0122 \*   
## AR-Nonseasonal-01 -0.11742 0.17175 -0.684 0.4942   
## AR-Nonseasonal-02 0.04078 0.08525 0.478 0.6324   
## AR-Nonseasonal-03 0.38949 0.06731 5.787 7.17e-09 \*\*\*  
## MA-Nonseasonal-01 0.22396 0.18661 1.200 0.2301   
## MA-Seasonal-12 0.82220 0.04863 16.907 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## X11 adj. ARIMA: (3 1 1)(0 1 1) Obs.: 195 Transform: none  
## AICc: 976, BIC: 997.8 QS (no seasonality in final): 0   
## Box-Ljung (no autocorr.): 22.67 Shapiro (normality): 0.9893

autoplot(ee.x11,facet=T) + xlab("Year") + ggtitle("X11 Decomposition of the Electrical Manufacturing Index")



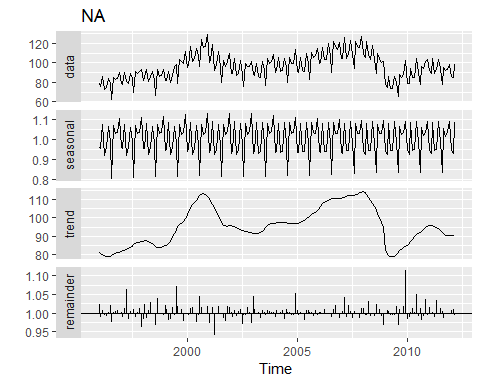
The results look similar to what we have seen previously in terms of the decomposition of this time series into the three components , , and . Notice that when calling the function seas above we specified two optional arguments. The first, transform.function="none" means we do NOT want a transformation to be applied to the time series. The default setting is transform.function="auto" which means the algorithm will automatically decide whether or not a transformation should be applied to the time series (e.g. ). The second optional argument is x11="" which means we want to use X11-decomposition vs. X13-decomposition which is the default.

In code below we again use X11-decomposition, but allow the algorithm to decide whether or not a transformation should be applied to the time series.

ee.x11.auto = seas(elecequip,x11="")  
summary(ee.x11.auto)

##   
## Call:  
## seas(x = elecequip, x11 = "")  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## LS2009.Jan -0.13348 0.02638 -5.059 4.22e-07 \*\*\*  
## AO2009.Dec 0.10497 0.02463 4.262 2.03e-05 \*\*\*  
## AR-Nonseasonal-01 0.79731 0.09064 8.796 < 2e-16 \*\*\*  
## MA-Nonseasonal-01 1.23304 0.09573 12.881 < 2e-16 \*\*\*  
## MA-Nonseasonal-02 -0.48547 0.06420 -7.561 3.99e-14 \*\*\*  
## MA-Seasonal-12 0.82471 0.05042 16.355 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## X11 adj. ARIMA: (1 1 2)(0 1 1) Obs.: 195 Transform: log  
## AICc: 942.4, BIC: 964.2 QS (no seasonality in final): 0   
## Box-Ljung (no autocorr.): 34.97 . Shapiro (normality): 0.9922

autoplot(ee.x11.auto,facet=T)

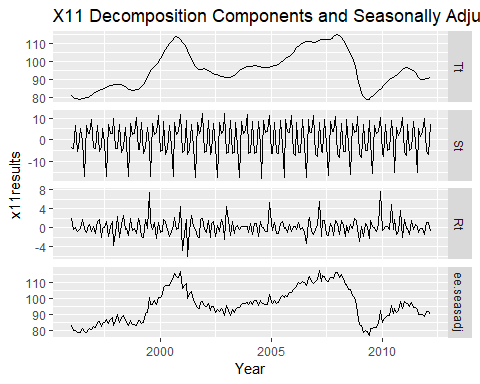


Notice in the summary of X11-decomposition is says that Transform: log was used, i.e. a multiplicative approach to the decomposition was employed.

We can extract the various components of our time series from the results of running seas() command. Given the output of the seas() function, e.g. ee.x11 above, we can use seasonal(ee.x11) to extract the seasonal component , trendcycle(ee.x11) to extract the long-term/cyclic trend component , and remainder(ee.x11) to extract the remainder component . The command seasadj(ee.x11) will extract the seasonally adjusted time series.

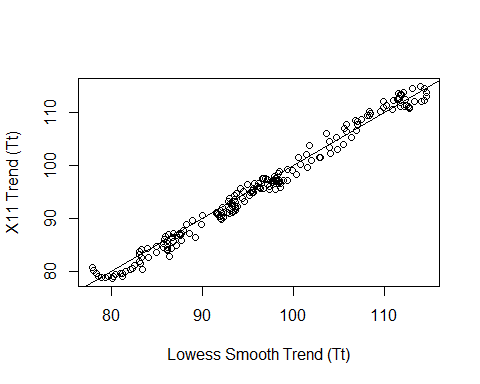
Tt = trendcycle(ee.x11)  
St= seasonal(ee.x11)  
Rt = remainder(ee.x11)  
ee.seasadj = seasadj(ee.x11)  
x11results = cbind(Tt,St,Rt,ee.seasadj)

autoplot(x11results,facet=T) + xlab("Year") + ggtitle("X11 Decomposition Components and Seasonally Adjusted Time Series")

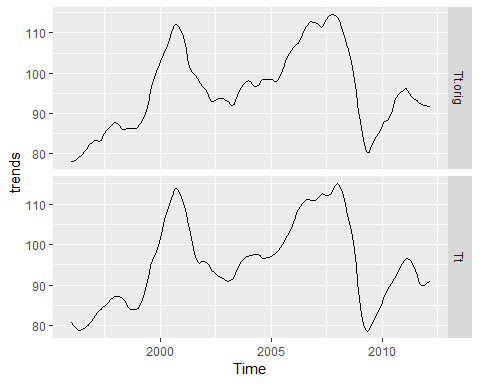


We can compare these to our earlier estimates obtained in our very first example of decomposing this time series.

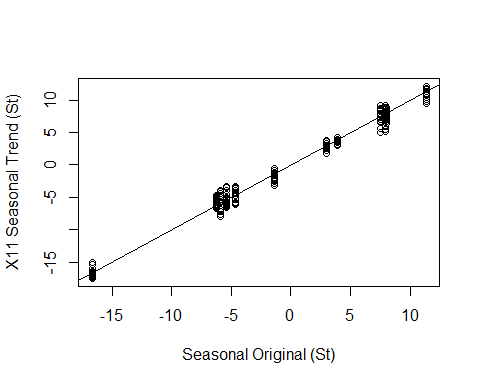
low3 = lowess(elecequip,f=.075)  
Tt.orig = low3$y  
EE.minusTt = elecequip - Tt  
seasonfit = tslm(EE.minusTt~season)  
St.orig = fitted(seasonfit)  
Rt.orig = EE.minusTt - St.orig  
plot(Tt.orig,Tt,xlab="Lowess Smooth Trend (Tt)",ylab="X11 Trend (Tt)")  
abline(0,1)



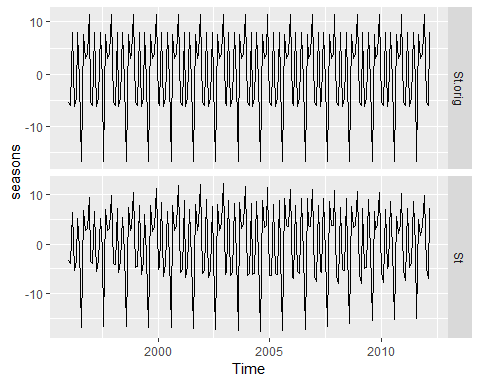
trends = cbind(Tt.orig,Tt)  
autoplot(trends,facet=T)



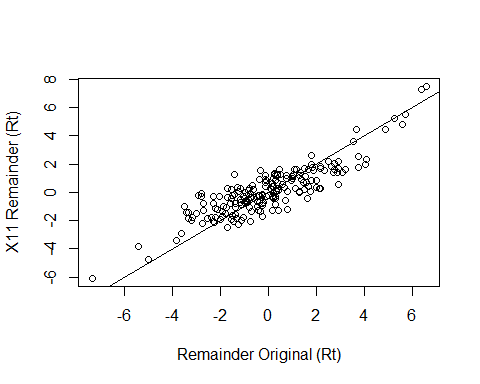
plot(St.orig,St,xlab="Seasonal Original (St)",ylab="X11 Seasonal Trend (St)")  
abline(0,1)



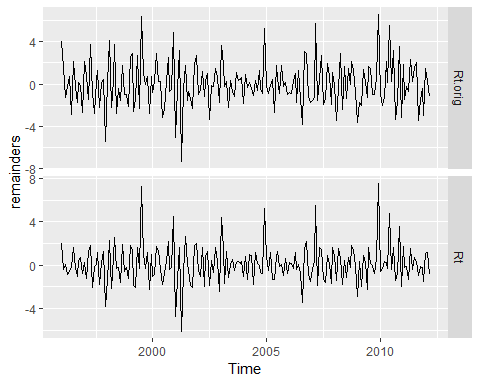
seasons = cbind(St.orig,St)  
autoplot(seasons,facet=T)



plot(Rt.orig,Rt,xlab="Remainder Original (Rt)",ylab="X11 Remainder (Rt)")  
abline(0,1)



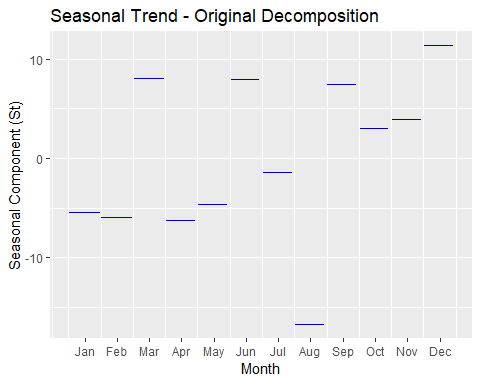
remainders = cbind(Rt.orig,Rt)  
autoplot(remainders,facet=T)



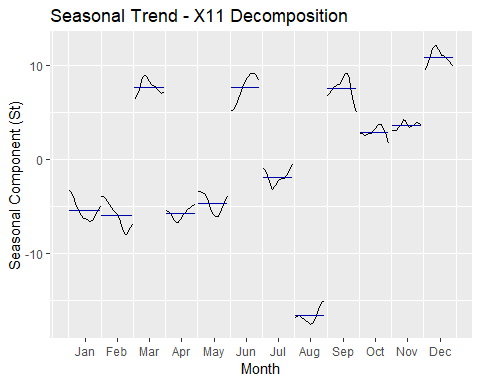
There are definitely differences in the estimated components with the most notable being in the long-term/cyclic trend estimate . As a result of these differences, the other components though not obviously differing visually, must necessarily differ as well due to the fact the long-term/cyclic trend () is removed from the time series in order to estimate the other two components.

The X11 method allows for the seasonal patterns to change over time. This can be seen by using the ggsubseriesplot command to examine them.

ggsubseriesplot(St.orig) + ylab("Seasonal Component (St)") + ggtitle("Seasonal Trend - Original Decomposition")



ggsubseriesplot(St) + ylab("Seasonal Component (St)") + ggtitle("Seasonal Trend - X11 Decomposition")

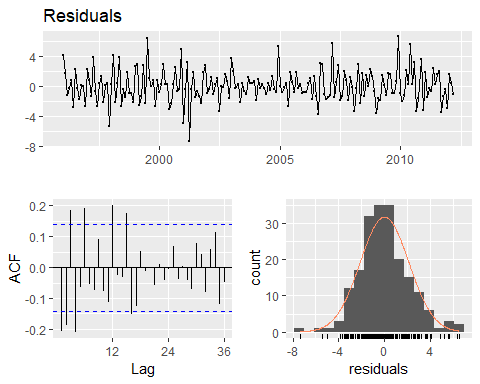


Notice how the season/month effect is constant in our original “by-hand” decomposition, whereas the X11 seasonal component changes over time (across years) for each an every month.

We can use the checkresiduals() command to compare the distributive properties of the remainder terms.

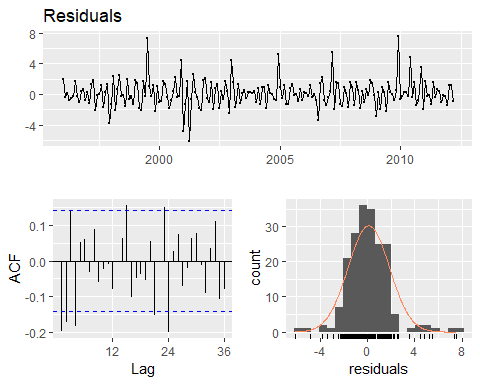
checkresiduals(Rt.orig)

## Warning in modeldf.default(object): Could not find appropriate degrees of  
## freedom for this model.



checkresiduals(Rt)

## Warning in modeldf.default(object): Could not find appropriate degrees of  
## freedom for this model.



The residuals are smaller for the X11 decomposition, but I would argue the residuals from our first “by-hand” decomposition have slightly less correlation structure.

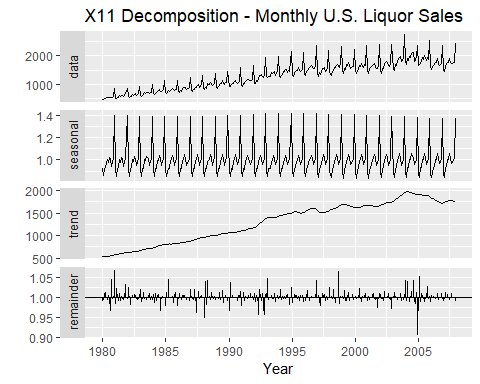
### Example 6.2 - Monthly U.S. Liquor Sales (cont’d)

We will now use X11 decomposition to decompose the monthly U.S. liquor sales times series and obtain a seasonally-adjusted version of this time series.

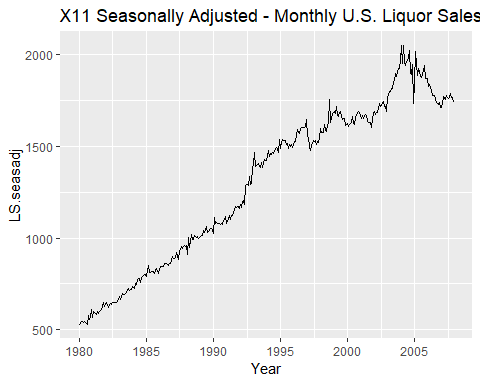
LS.X11 = seas(LiqSales,x11="")  
summary(LS.X11)

##   
## Call:  
## seas(x = LiqSales, x11 = "")  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## Mon 0.0111006 0.0026699 4.158 3.22e-05 \*\*\*  
## Tue -0.0050011 0.0026717 -1.872 0.0612 .   
## Wed -0.0067356 0.0026673 -2.525 0.0116 \*   
## Thu -0.0068152 0.0026605 -2.562 0.0104 \*   
## Fri -0.0012697 0.0026672 -0.476 0.6341   
## Sat -0.0008859 0.0026681 -0.332 0.7399   
## Constant -0.0003022 0.0001300 -2.325 0.0201 \*   
## AO2004.Dec -0.1021728 0.0206405 -4.950 7.42e-07 \*\*\*  
## MA-Nonseasonal-01 0.4687727 0.0480344 9.759 < 2e-16 \*\*\*  
## MA-Seasonal-12 0.8687013 0.0301654 28.798 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## X11 adj. ARIMA: (0 1 1)(0 1 1) Obs.: 336 Transform: log  
## AICc: 3163, BIC: 3203 QS (no seasonality in final): 0   
## Box-Ljung (no autocorr.): 22.99 Shapiro (normality): 0.9798 \*\*\*

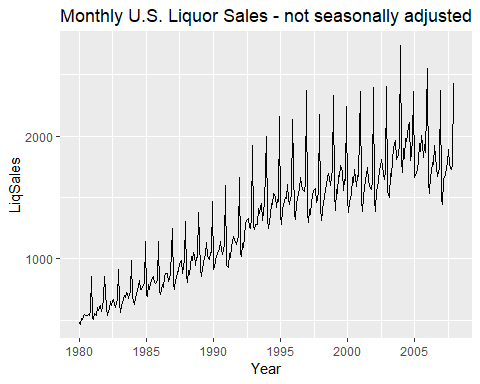
autoplot(LS.X11) + xlab("Year") + ggtitle("X11 Decomposition - Monthly U.S. Liquor Sales")



LS.seasadj = seasadj(LS.X11)  
autoplot(LS.seasadj) + xlab("Year") + ggtitle("X11 Seasonally Adjusted - Monthly U.S. Liquor Sales")



autoplot(LiqSales) + xlab("Year") + ggtitle("Monthly U.S. Liquor Sales - not seasonally adjusted")



Some things to notice about from results above:

* The X11 algorithm automatically recognized the need to log-transform the series, i.e. use a multiplicative approach when decomposing the series.
* It estimated a weekly trend, adjusting sales by day of week. I assume this achieved by counting the number of each day (M,T,W,…) that occur during the month that the sales figure is for.
* It estimated a separate coefficient for a downward shock in sales in December 2004. This shock is clearly evident in the seasonally adjusted time series.
* Moving Averages (MA) were used to estimate both the long-term and seasonal trends.

# 6.5 - SEATS Decompostion

## (X-13 Seasonal Extraction in ARIMA Time Series)

“SEATS” stands for “Seasonal Extraction in ARIMA Time Series” (ARIMA models are discussed in Chapter 8). This procedure was developed at the Bank of Spain, and is now widely used by government agencies around the world. The procedure works only with quarterly and monthly data. So seasonality of other kinds, such as daily data, or hourly data, or weekly data, require an alternative approach.

The details are beyond the scope of the FPP book. However, a complete discussion of the method is available in Dagum and Bianconcini (2016). Here we will only demonstrate how to use it via the seasonal package.

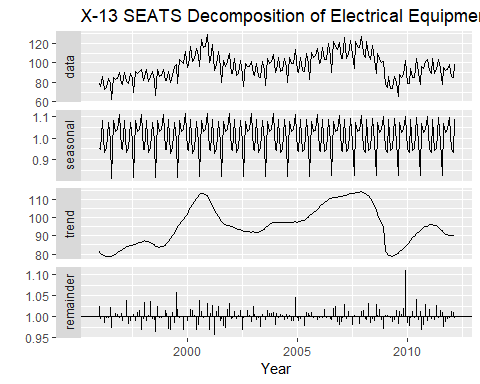
We will revisit the previous two examples using X-13 SEATS decomposition instead.

### Example 6.1 - Electrical Equipment Manufacturing Index (cont’d)

ee.X13 = seas(elecequip)  
summary(ee.X13)

##   
## Call:  
## seas(x = elecequip)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## LS2009.Jan -0.13348 0.02638 -5.059 4.22e-07 \*\*\*  
## AO2009.Dec 0.10497 0.02463 4.262 2.03e-05 \*\*\*  
## AR-Nonseasonal-01 0.79731 0.09064 8.796 < 2e-16 \*\*\*  
## MA-Nonseasonal-01 1.23304 0.09573 12.881 < 2e-16 \*\*\*  
## MA-Nonseasonal-02 -0.48547 0.06420 -7.561 3.99e-14 \*\*\*  
## MA-Seasonal-12 0.82471 0.05042 16.355 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## SEATS adj. ARIMA: (1 1 2)(0 1 1) Obs.: 195 Transform: log  
## AICc: 942.4, BIC: 964.2 QS (no seasonality in final): 0   
## Box-Ljung (no autocorr.): 34.97 . Shapiro (normality): 0.9922

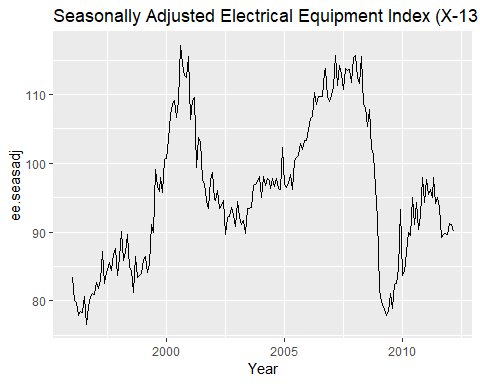
autoplot(ee.X13) + xlab("Year") + ggtitle("X-13 SEATS Decomposition of Electrical Equipment Index")



The result is quite similar to the X11 decomposition shown in Figure 6.9.

As with the X11 method, we can use the seasonal(), trendcycle() and remainder() functions to extract the individual components, and seasadj() to compute the seasonally adjusted time series. Below we extract the X-13 SEATS seasonally adjusted version of the electrical equipment manufacturing index.

ee.seasadj = seasadj(ee.X13)  
autoplot(ee.seasadj) + xlab("Year") + ggtitle("Seasonally Adjusted Electrical Equipment Index (X-13 SEATS)")



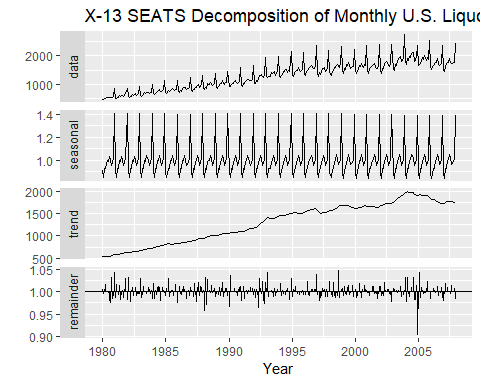
The seasonal package has many options for handling variations of X11 and SEATS. See the package website for a detailed introduction to the options and features available. (<http://www.seasonal.website/seasonal.html>)

### Example 6.2 - Monthly U.S. Liquor Sales (cont’d)

LS.X13 = seas(LiqSales)  
summary(LS.X13)

##   
## Call:  
## seas(x = LiqSales)  
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## Mon 0.0111006 0.0026699 4.158 3.22e-05 \*\*\*  
## Tue -0.0050011 0.0026717 -1.872 0.0612 .   
## Wed -0.0067356 0.0026673 -2.525 0.0116 \*   
## Thu -0.0068152 0.0026605 -2.562 0.0104 \*   
## Fri -0.0012697 0.0026672 -0.476 0.6341   
## Sat -0.0008859 0.0026681 -0.332 0.7399   
## Constant -0.0003022 0.0001300 -2.325 0.0201 \*   
## AO2004.Dec -0.1021728 0.0206405 -4.950 7.42e-07 \*\*\*  
## MA-Nonseasonal-01 0.4687727 0.0480344 9.759 < 2e-16 \*\*\*  
## MA-Seasonal-12 0.8687013 0.0301654 28.798 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## SEATS adj. ARIMA: (0 1 1)(0 1 1) Obs.: 336 Transform: log  
## AICc: 3163, BIC: 3203 QS (no seasonality in final): 0   
## Box-Ljung (no autocorr.): 22.99 Shapiro (normality): 0.9798 \*\*\*

autoplot(LS.X13) + xlab("Year") + ggtitle("X-13 SEATS Decomposition of Monthly U.S. Liquor Sales")



The differences between the X-11 and X-13 approaches for this time series are VERY minimal.

# 6.6 - STL Decomposition

STL is a very versatile and robust method for decomposing time series. STL is an acronym for “Seasonal and Trend decomposition using Loess”, while Loess is a method for estimating nonlinear relationships. The STL method was developed by Cleveland et al. (1990).

STL has several advantages over the classical, SEATS and X-11 decomposition methods:

* Unlike SEATS and X-11, STL will handle any type of seasonality, not only monthly and quarterly data.
* The seasonal component is allowed to change over 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.
* It can be robust to outliers (i.e., the user can specify a robust decomposition), so that occasional unusual observations will not affect the estimates of the trend-cycle and seasonal components. They will, however, affect the remainder component.

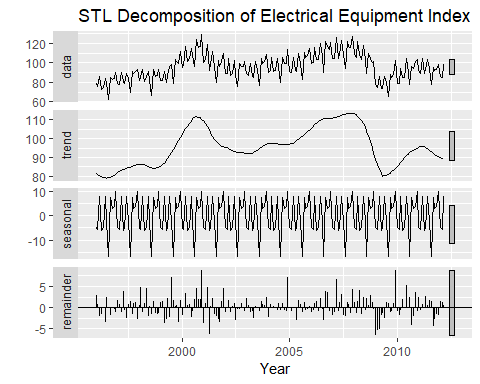
On the other hand, STL has some disadvantages. In particular, it does not handle trading day or calendar variation automatically, and it only provides facilities for additive decompositions.

It is possible to obtain a multiplicative decomposition by first taking logs of the data, then back-transforming the components. Decompositions between additive and multiplicative can be obtained using a Box-Cox transformation of the data with . A value of corresponds to the multiplicative decomposition while is equivalent to an additive decomposition.

In our first decomposition of the electrical equipment manufacturing time series in Section 6.1 we used Loess smoothing to estimate the long-term/cyclic trend component . Thus that example is much more akin to the STL decompostion approach than the X-11 and X-13 SEATS methods.

The best way to begin learning how to use STL is to see some examples and experiment with the settings. In the first example below STL is applied to the electrical equipment orders data.

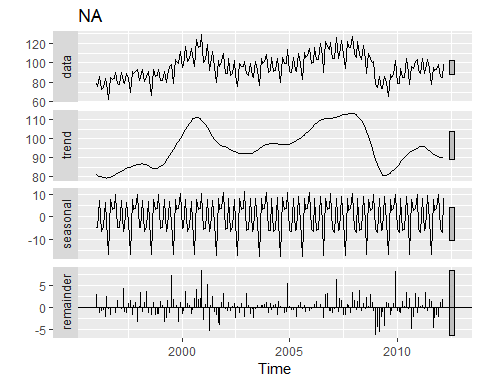
ee.stl = stl(elecequip,t.window=13,s.window="periodic",robust=TRUE)  
autoplot(ee.stl) + xlab("Year") + ggtitle("STL Decomposition of Electrical Equipment Index")



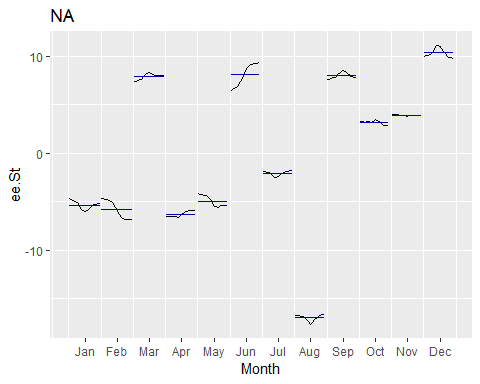
The results from shows an STL decomposition where the trend-cycle is estimated by using 13 years at a time, the seasonal component does not change over time, and the robust option has been used. Robustness in general means “outlier resistant”“, thus a robust fit will be less effected by strange observations in a time series. From the resulting decomposition it is obvious that there has been a down-turn at the end of the series, and that the orders in 2009 were unusually low (corresponding to some large negative values in the remainder component).

When performing an STL decomposition of a time series there are two main tuning parameters that control the flexibility of the fits in determining the long-term/cyclic and seasonal trends. The two main parameters are the width of trend-cycle window (t.window) and the seasonal window (s.window). These control how rapidly the trend-cycle and seasonal components can change or how flexible the fits will be. Smaller values allow for more rapid changes and greater flexibility. Both t.window and s.window should be odd numbers and refer to the number of consecutive years to be used when estimating the trend-cycle and seasonal components respectively. The user must specify s.window as there is no default. Setting it to be infinite is equivalent to forcing the seasonal component to be “periodic” (i.e., identical across years). Specifying t.window is optional, and a default value will be used if it is omitted.

ee.stl2 = stl(elecequip,t.window=13,s.window=13,robust=TRUE)  
autoplot(ee.stl2)



ee.St = seasonal(ee.stl2)  
ggsubseriesplot(ee.St)

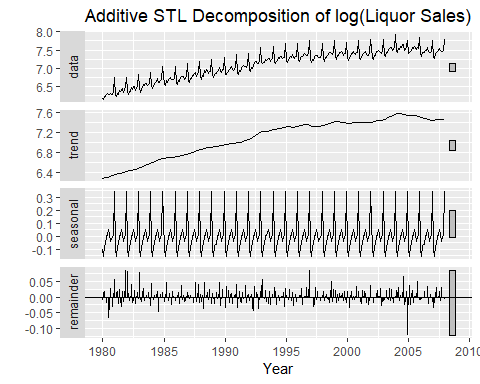


By not using s.window="periodic we can see that the seasonal pattern is NOT constant from year-to-year. For many time series this might be more appropriate.

### Example 6.2 - Monthly U.S. Liquor Sales (cont’d)

We will now consider STL decomposition of the monthly liquor sales in the US. As we have seen before this time series has non-constant seasonal variation, thus a multiplicative decomposition makes sense. In order to do this however, we need to use the fact decomposing the log-transformed time series additively is the same.

logLS = log(LiqSales)  
logLS.stl = stl(logLS,t.window=13,s.window="periodic",robust=TRUE)  
autoplot(logLS.stl) + xlab("Year") + ggtitle("Additive STL Decomposition of log(Liquor Sales)")



# 6.7 - Forecasting with Decomposition

While decomposition is primarily useful for studying time series data, and exploring the historical changes over time, it can also be used in forecasting.

Assuming an additive decomposition, the decomposed time series can be written as

,

where is the seasonally adjusted time series. If multiplicative composition has been used, we can write

where the seasonally adjusted time series.

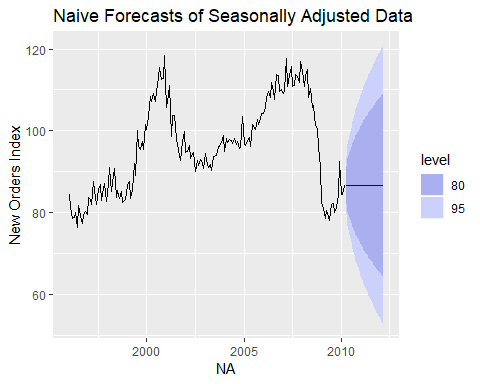
To forecast using a seasonal decomposition, we forecast the seasonal component, , and the seasonally adjusted component , separately. It is usually assumed that the seasonal component is unchanging, or changing extremely, slowly, so it is forecast by simply taking the last year of the estimated component. In other words, a seasonal naive method is used for the seasonal component.

To forecast the seasonally adjusted component, any non-seasonal forecasting method may be used. For example, a drift model, or Holt’s method (discussed next chapter), or a non-seasonal ARIMA model (discussed in Chapter 8), may be used.

### Example 6.1 - Electrical Equipment Manufacturing Index (cont’d)

In this example we will examine the use of stl() in forecasting values of this time series. FOr validation purposes we will use the train and test set approach to choose between rival options for forecasting.

ee.test = tail(elecequip,24)  
ee.train = head(elecequip,171)  
ee.STL = stl(ee.train,t.window=13,s.window="periodic",robust=TRUE)  
ee.SA = seasadj(ee.STL)  
ee.fc1 = naive(ee.SA,h=24)  
# Forecast with No Seasonality Added Back!  
autoplot(ee.fc1) + ylab("New Orders Index") + ggtitle("Naive Forecasts of Seasonally Adjusted Data")



accuracy(ee.fc1,ee.test)

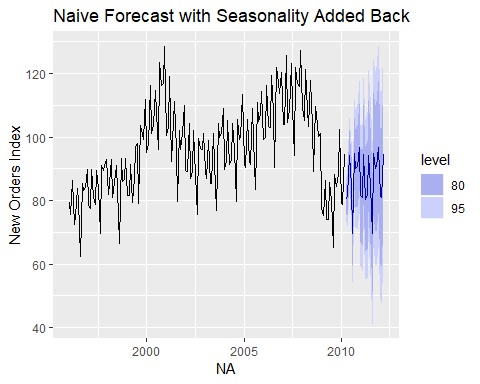
## ME RMSE MAE MPE MAPE MASE  
## Training set 0.01312782 3.556679 2.708036 -0.0527441 2.834883 0.3279942  
## Test set 6.17225503 9.695715 8.109253 6.0173221 8.455577 0.9821835  
## ACF1 Theil's U  
## Training set -0.31870194 NA  
## Test set -0.06728058 0.8821864

The plot above shows naive forecasts using the seasonally adjusted electrical equipment orders index. These need to be re-seasonalized by adding in the seasonal naive forecasts of the seasonal component.

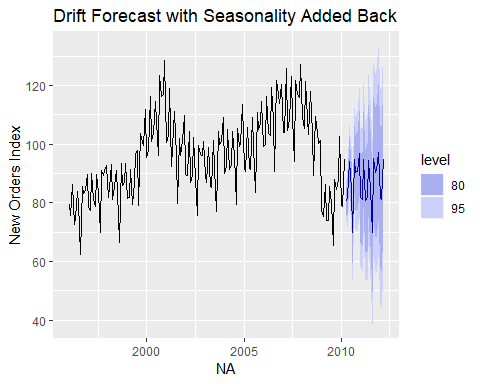
This is made easy with the forecast() function applied directly to the results from the stl() function. Here that would mean the ee.STL object fit above. You also need to specify the method used to forecast the seasonally adjusted data, which in the above example was the naive (last value carried forward) forecast. The forecast function will take care of the reseasonalizing for you.

Below we will compare a naive and drift method forecast for seasonally adjusted data with the reseasonalizing being taken care of the forecast function. We will compare these two methods for forecasting the seasonally adjusted component by comparing the actual test values to their forecasts.

# Add Back in Seasonality  
ee.fc2 = forecast(ee.STL,method="naive",h=24)  
autoplot(ee.fc2) + ylab("New Orders Index") + ggtitle("Naive Forecast with Seasonality Added Back")



ee.fc3 = forecast(ee.STL,method="rwdrift",h=24)   
autoplot(ee.fc3) + ylab("New Orders Index") + ggtitle("Drift Forecast with Seasonality Added Back")



accuracy(ee.fc2,ee.test) # Naive Method

## ME RMSE MAE MPE MAPE MASE  
## Training set 0.01312782 3.556679 2.708036 -0.07546459 2.844520 0.3279942  
## Test set 6.17225502 7.053658 6.172255 6.65908059 6.659081 0.7475765  
## ACF1 Theil's U  
## Training set -0.3187019 NA  
## Test set 0.4211218 0.6163662

accuracy(ee.fc3,ee.test) # Drift Method

## ME RMSE MAE MPE MAPE  
## Training set -2.340719e-15 3.556654 2.707109 -0.08940223 2.843716  
## Test set 6.008157e+00 6.923570 6.009174 6.48153228 6.482602  
## MASE ACF1 Theil's U  
## Training set 0.3278820 -0.3187019 NA  
## Test set 0.7278243 0.4292718 0.6048636

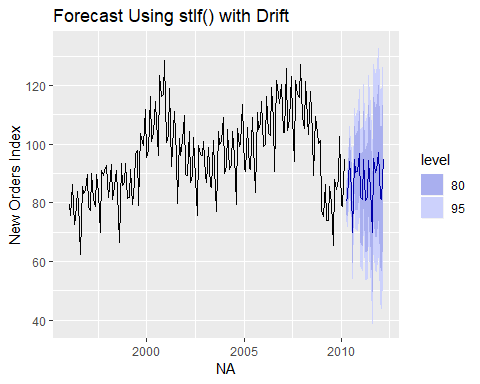
# Which of these forecast approaches for the seasonally adjusted time series is best here?  
#  
# The table below contains these forecasts  
ee.fc3

## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Apr 2010 80.47211 75.88718 85.05703 73.46006 87.48415  
## May 2010 81.72254 75.20067 88.24442 71.74820 91.69689  
## Jun 2010 94.22328 86.18961 102.25695 81.93684 106.50972  
## Jul 2010 84.79611 75.46677 94.12545 70.52811 99.06411  
## Aug 2010 69.83139 59.34210 80.32067 53.78941 85.87337  
## Sep 2010 94.89466 83.34021 106.44911 77.22365 112.56567  
## Oct 2010 90.29914 77.75015 102.84814 71.10711 109.49118  
## Nov 2010 90.79231 77.30376 104.28086 70.16336 111.42127  
## Dec 2010 96.97720 82.59331 111.36109 74.97893 118.97547  
## Jan 2011 81.71301 66.47019 96.95583 58.40113 105.02489  
## Feb 2011 81.13263 65.06146 97.20379 56.55390 105.71136  
## Mar 2011 94.77753 77.90409 111.65098 68.97183 120.58324  
## Apr 2011 80.62964 62.97644 98.28284 53.63140 107.62788  
## May 2011 81.88008 63.46678 100.29337 53.71937 110.04078  
## Jun 2011 94.38081 75.22474 113.53689 65.08412 123.67750  
## Jul 2011 84.95364 65.07016 104.83712 54.54448 115.36280  
## Aug 2011 69.98892 49.39178 90.58606 38.48832 101.48953  
## Sep 2011 95.05220 73.75377 116.35063 62.47906 127.62533  
## Oct 2011 90.45668 68.46814 112.44522 56.82811 124.08525  
## Nov 2011 90.94985 68.28136 113.61833 56.28139 125.61831  
## Dec 2011 97.13473 73.79557 120.47390 61.44056 132.82890  
## Jan 2012 81.87054 57.86920 105.87189 45.16366 118.57743  
## Feb 2012 81.29016 56.63445 105.94587 43.58250 118.99782  
## Mar 2012 94.93507 69.63219 120.23794 56.23766 133.63248

The prediction intervals shown the graphs above are constructed in the same way as the point forecasts. That is, the upper and lower limits of the prediction intervals on the seasonally adjusted data are “reseasonalized” by adding in the forecasts of the seasonal component. In this calculation, the uncertainty in the forecasts of the seasonal component has been ignored. The rationale for this choice is that the uncertainty in the seasonal component is much smaller than that for the seasonally adjusted data, and so it is a reasonable approximation to ignore it.

A short-cut approach is to use the stlf() function. The following code will decompose the time series using STL, forecast the seasonally adjusted series using the method specified, and return reseasonalized the forecasts.

ee.fc2 = stlf(ee.train,t.window=13,s.window="periodic",robust=TRUE,method="naive",h=24)  
ee.fc3 = stlf(ee.train,t.window=13,s.window="periodic",robust=TRUE,method="rwdrift",h=24)  
autoplot(ee.fc3) + ylab("New Orders Index") + ggtitle("Forecast Using stlf() with Drift")



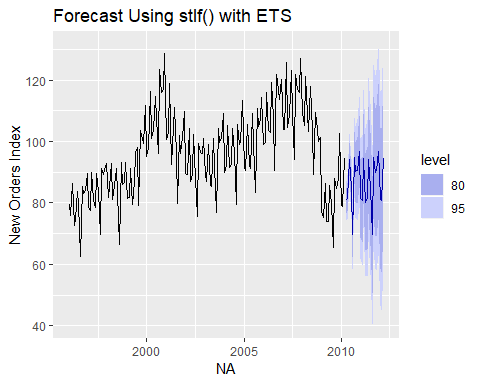
accuracy(ee.fc3,ee.test)

## ME RMSE MAE MPE MAPE  
## Training set -2.340719e-15 3.556654 2.707109 -0.08940223 2.843716  
## Test set 6.008157e+00 6.923570 6.009174 6.48153228 6.482602  
## MASE ACF1 Theil's U  
## Training set 0.3278820 -0.3187019 NA  
## Test set 0.7278243 0.4292718 0.6048636

The stlf() function uses mstl() to carry out the decomposition, so there are default values for s.window and t.window, however in the examples above the settings used previous were chosen for consistency purposes.

As well as the naïve and rwdrift methods, there are two other possible forecasting methods are available with stlf(), as described in the corresponding help file. If NO method is specified, it will use the **ETS** approach (discussed in the next chapter) applied to the seasonally adjusted series. This usually produces quite good forecasts for seasonal time series, and some companies use it routinely for all their operational forecasts. Let’s try the ETS approach with these data.

ee.fc4 = stlf(ee.train,t.window=13,s.window="periodic",robust=TRUE,method="ets",h=24)  
autoplot(ee.fc4) + ylab("New Orders Index") + ggtitle("Forecast Using stlf() with ETS")



accuracy(ee.fc4,ee.test)

## ME RMSE MAE MPE MAPE MASE  
## Training set 0.04899387 3.171031 2.394729 0.01382527 2.530284 0.2900468  
## Test set 6.25412253 7.127832 6.254123 6.74586113 6.745861 0.7574922  
## ACF1 Theil's U  
## Training set 0.02911534 NA  
## Test set 0.42059755 0.6232028

Here the drift method outperforms both the naive and ETS forecasts.

### Example 6.2 - Monthly U.S. Liquor Sales (cont’d)

As we have discussed the log-transformation should be employed when doing an STL decomposition of this time series. How can we make forecasts in the original scale using the stlf() function? We simply include the argument lambda=0 when using the stlf() function.

We will again use the last two years of this time series as a test set and compare forecast accuracy using different methods for forecasting the seasonally adjusted component.

length(LiqSales)

## [1] 336

liq.test = tail(LiqSales,24)  
liq.train = head(LiqSales,312)  
liq.fc1 = stlf(liq.train,t.window=13,s.window="periodic",robust=TRUE,method="naive",h=24,lambda=0)  
liq.fc2 = stlf(liq.train,t.window=13,s.window="periodic",robust=TRUE,method="rwdrift",h=24,lambda=0)  
liq.fc3 = stlf(liq.train,t.window=13,s.window="periodic",robust=TRUE,method="ets",h=24,lambda=0)  
accuracy(liq.fc1,liq.test)

## ME RMSE MAE MPE MAPE MASE  
## Training set 4.051269 48.31927 35.21479 0.332205 2.833776 0.4791345  
## Test set -60.577500 78.92719 62.81199 -3.406061 3.529221 0.8546236  
## ACF1 Theil's U  
## Training set -0.4426322 NA  
## Test set 0.4279976 0.313939

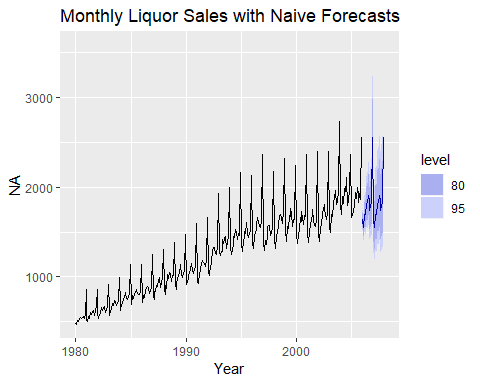
accuracy(liq.fc2,liq.test)

## ME RMSE MAE MPE MAPE  
## Training set -0.9460547 48.32949 34.58192 -0.06172921 2.790402  
## Test set -155.9799304 180.38696 155.97993 -8.69162900 8.691629  
## MASE ACF1 Theil's U  
## Training set 0.4705237 -0.4391227 NA  
## Test set 2.1222722 0.5912656 0.7296247

accuracy(liq.fc3,liq.test)

## ME RMSE MAE MPE MAPE MASE  
## Training set -3.185519 42.49534 28.46804 -0.2422149 2.242624 0.3873379  
## Test set -160.146600 175.43720 160.14660 -8.9864544 8.986454 2.1789641  
## ACF1 Theil's U  
## Training set -0.005534282 NA  
## Test set 0.494292644 0.7086746

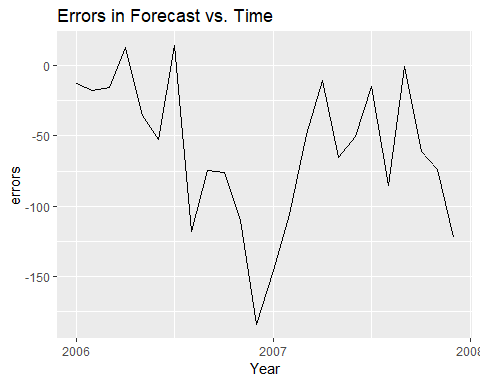
# The naive method appears to produce the best accuracy  
autoplot(liq.fc1) + xlab("Year") + ggtitle("Monthly Liquor Sales with Naive Forecasts")



# Table of forecasts  
liq.fc1

## Point Forecast Lo 80 Hi 80 Lo 95 Hi 95  
## Jan 2006 1636.937 1564.578 1712.642 1527.578 1754.125  
## Feb 2006 1550.951 1454.891 1653.353 1406.472 1710.271  
## Mar 2006 1691.688 1564.271 1829.483 1500.753 1906.914  
## Apr 2006 1693.428 1547.026 1853.685 1474.720 1944.572  
## May 2006 1816.321 1641.679 2009.542 1556.133 2120.014  
## Jun 2006 1824.707 1633.421 2038.393 1540.417 2161.463  
## Jul 2006 1907.758 1692.680 2150.164 1588.822 2290.717  
## Aug 2006 1860.857 1637.487 2114.697 1530.310 2262.803  
## Sep 2006 1743.852 1522.670 1997.163 1417.176 2145.830  
## Oct 2006 1788.692 1550.405 2063.600 1437.397 2225.841  
## Nov 2006 1842.740 1586.147 2140.843 1465.111 2317.702  
## Dec 2006 2553.000 2182.903 2985.844 2009.226 3243.940  
## Jan 2007 1636.937 1390.715 1926.751 1275.741 2100.398  
## Feb 2007 1550.951 1309.580 1836.809 1197.406 2008.883  
## Mar 2007 1691.688 1419.958 2015.416 1294.255 2211.161  
## Apr 2007 1693.428 1413.280 2029.109 1284.259 2232.961  
## May 2007 1816.321 1507.430 2188.509 1365.783 2415.481  
## Jun 2007 1824.707 1506.227 2210.527 1360.795 2446.772  
## Jul 2007 1907.758 1566.527 2323.318 1411.340 2578.783  
## Aug 2007 1860.857 1520.212 2277.832 1365.907 2535.157  
## Sep 2007 1743.852 1417.531 2145.294 1270.286 2393.965  
## Oct 2007 1788.692 1446.908 2211.210 1293.269 2473.899  
## Nov 2007 1842.740 1483.541 2288.908 1322.672 2567.296  
## Dec 2007 2553.000 2045.791 3185.961 1819.455 3582.286

# Calculating Prediction Accuracy Measures "By-Hand"  
errors = liq.test - liq.fc1$mean  
autoplot(errors) + xlab("Year") + ggtitle("Errors in Forecast vs. Time")



RMSEP = sqrt(mean(errors^2))  
RMSEP

## [1] 78.92719

MAE = mean(abs(errors))  
MAE

## [1] 62.81199

MAPE = mean(abs(errors)/liq.test)\*100  
MAPE

## [1] 3.529221

# We can see these agree with those returned by accuracy() function!