Week 4 Topics

1. Chapter 5 – Smoothing Methods

Simple Exponential Smoothing (SES)

We have already considered the naïve and the average as possible methods for forecasting data.

Using the naïve method, all forecasts for the future are equal to the last observed value of the series,

for k = 1, 2, … . Therefore, the naïve method assumes that the most recent observation is the only important one, and all previous observations provide no information for the future. This can be thought of as a weighted average where all of the weight is given to the last observation.

Using the average method, all future forecasts are equal to a simple average of the observed data,

for k= 1, 2, … *.* As we see, the average method assumes that all observations are of equal importance (have the same weight), and gives them equal weights when generating forecasts.

We want something between these two extremes. For example, it may be sensible to attach larger weights to more recent observations than to observations from the distant past. This is exactly the concept behind simple exponential smoothing. Forecasts are calculated using weighted averages, where the weights decrease exponentially as observations come from further in the past — *the smallest weights are associated with the oldest observations.*

Simple exponential smoothing (SES) is a popular forecasting method in business. Its popularity derives from its flexibility, ease of automation, cheap computation, and good performance. SES is similar to forecasting with a moving average, except that instead of taking a simple average over the w most recent values, we take a weighted average of all past values, so that the weights decrease exponentially into the past. The idea is to give more weight to recent information, yet not to completely ignore older information. Like the moving average, SES should only be used for forecasting series that have no trend or seasonality. As we saw earlier, such series can be obtained by removing trend and/or seasonality from raw series, and then applying exponential smoothing to the series of residuals (which are assumed to contain no trend or seasonality). The SES generates a forecast at the time *t+1* (Ft+1) using the following formula:

where  is a constant between 0 and 1 called the smoothing constant. The above formulation displays the exponential smoother as a weighted average of all past observations, with exponentially decaying weights. We can also write the exponential forecaster in another way, which is very useful in practice:

where  is the forecast error at time t. This formulation presents the exponential forecaster as an “active learner”. It looks at the previous forecast () and at its distance from the actual value (), and then corrects the next forecast based on that information. If the forecast was too high in the last period, the next period is adjusted down. The amount of correction depends on the value of the smoothing constant α.

The second formula is also advantageous in terms of data storage and computation time: we need to store and use only the forecast and forecast error from the previous period, rather than the entire series. In applications where real-time forecasting is done, or many series are being forecasted in parallel and continuously, such savings are critical.

Note: forecasting further into the future yields the same forecasts as a one-step-ahead forecast () because the series is assumed to lack trend and seasonality.

# Choosing Smoothing Constant α

The smoothing constant , which is set by the user, determines the rate of learning. A value close to 1 indicates fast learning (that is, only the most recent values influence the forecasts), whereas a value close to 0 indicates slow learning (past observations have a large influence on forecasts). This can be seen by plugging 0 or 1 into the two equations above. Hence, the choice of *α* depends on the required amount of smoothing, and on how relevant the history is for generating forecasts. Default values that have been shown to work well are in the range 0.1-0.2.

Trial and error can also help in the choice of . Examine the time plot of the actual and predicted series, as well as the predictive accuracy (e.g., MAPE or RMSE of the validation period).

The table below shows the weights attached to observations for four different values of when forecasting using simple exponential smoothing. Note that the sum of the weights even for a small value of will be approximately one for any reasonable sample size.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | α=0.2 | α=0.4 | α=0.6 | α=0.8 |
| yt | 0.2000 | 0.4000 | 0.6000 | 0.8000 |
| yt-1 | 0.1600 | 0.2400 | 0.2400 | 0.1600 |
| yt-2 | 0.1280 | 0.1440 | 0.0960 | 0.0320 |
| yt-3 | 0.1024 | 0.0864 | 0.0384 | 0.0064 |
| yt-4 | 0.0819 | 0.0518 | 0.0154 | 0.0013 |
| yt-5 | 0.0655 | 0.0311 | 0.0061 | 0.0003 |

In R, forecasting using SES can be done via the *ets()* function in the forecast package. The three letters in *ets* stand for *error, trend, and seasonality*. Applying this function to a time series will yield forecasts and forecast errors (residuals) for both the training and validation periods. You can use a default value of *α* = 0.2, set it to another value, or choose to find the optimal *α* in terms of minimizing RMSE over the training period. To choose an SES using the *ets* function, we set model = "ANN" (additive error (A), no trend (N), and no seasonality (N)). [For an illustration of using the *ets()* function and comparing default and optimized values, see Section 5.4 in the textbook Practical Time Series Forecasting with R, 2nd edition, by Shmueli & Lichtendahl ]

### Link Between Moving Average and Simple Exponential Smoothing

In both the moving average and simple exponential smoothing methods, the user must specify a single parameter: in moving averages, the window width (w); and in SES, the smoothing constant (*α*). In both cases, the parameter determines the importance of fresh information over older information. In fact, the two smoothers are approximately equal if the window width of the moving average is equal to .

# R-Code for SES

Data.Set.Size <- **window**(Data.ts, start=<start date>, End = <end date>)

*# Estimate parameters*

fc <- **ses**(Data.Set.Size, h=<horizon>)

*# Accuracy of one-step-ahead training errors*

**round**(**accuracy**(fc),2)

Holt’s (Double) Exponential Smoothing

In R, to specify a Holt’s exponential smoothing model with an additive or multiplicative trend, use the ets() function and set the second parameter in model= to A (=Additive) or M (=Multiplicative). As you remember from previous sections, the first character is the model parameter is for error. That is Additive error (A \_ \_ ) and Multiplicative error (M \_ \_ ). Keep the third parameter equal to N (to specify no seasonality). For example, model = “AAN” gives Holt’s exponential smoothing with additive error and additive trend.

Comparing both categories Additive and Multiplicative trend shown in the table below.

|  |  |
| --- | --- |
| HES Additive Trend | HES Multiplicative Trend |
| At the time t, with k-step-ahead: |  |
|  |  |
|  |  |
|  |  |

In R, to specify a Holt’s exponential smoothing model with an additive or multiplicative trend, use the ets() function and set the second parameter in model= to A (=Additive) or M (=Multiplicative). As you remember from previous sections, the first character is the model parameter is for error. That is Additive error (A \_ \_ ) and Multiplicative error (M \_ \_ ). Keep the third parameter equal to N (to specify no seasonality). For example, model = “AAN” gives Holt’s exponential smoothing with additive error and additive trend.

# R-Code for Holt - ES

Data.Set.Size <- **window**(Data.ts, start=<start date>, End = <end date>)

*# Estimate parameters*

fc <- **holt**(Data.Set.Size, h=<horizon>)

*# Accuracy of one-step-ahead training errors*

**round**(**accuracy**(fc),2)

Holt-Winter’s (Triple) Exponential Smoothing

Comparing both categories Additive and Multiplicative seasonality shown in the table below.

|  |  |
| --- | --- |
| HWES Additive Seasonality | WHES Multiplicative Seasonality |
| At the time t, with k-step-ahead: |  |
|  |  |
|  |  |
|  |  |

# R-Code for Holt Winters- ES

Data.Set.Size <- **window**(Data.ts, start=<start date>, End = <end date>)

fit1 <- **hw**(Data.Set.Size,seasonal="additive")

fit2 <- **hw**(Data.Set.Size ="multiplicative")

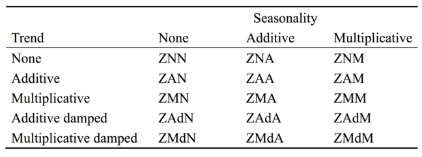
For SES, HES and HWES, I omitted other parameters in forecasting function. Refer to these functions in R and check other parameters.

The *ets*( ) function in R (Just for review)

For the additive seasonality and multiplicative seasonality versions of the exponential smoothing model introduced earlier, we can include either an additive or multiplicative error. We can also specify an additive trend, a multiplicative trend, or no trend at all. This flexibility of the ***ets()***function in R gives us 18 models from which to choose: 2 error types × 3 trend types × 3 seasonality types.

There are two other trend types that the ***ets*** function allows: the additive damped trend (Ad) and the multiplicative damped trend (Md). These advanced trend types apply to time series whose trends will eventually dampen to a flat line in the distant future. With these two other trend types, the number of possible models the ***ets()***function can fit increases to 30.

All these models are summarized in the Table 1 below, where the error (Z) can be either set to additive (A) or multiplicative (M).

Table 1: Possible exponential smoothing models in R

Time series models can be estimated in R using the *ets*( ) function in forecast package. Unlike the ses( ), holt( ), and wh( ) functions, the ets( ) function does not produce forecasts. Rather, it estimates the model parameters and returns information about the fitted model. By default it uses the AICc to select an appropriate model, although other information criteria can be selected.

The R code below shows the most important arguments that this function can take, and their default values. If only the time series is specified, and all other arguments are left at their default values, then an appropriate model will be selected automatically. I explain the arguments below.

**ets**(data.ts, model = "ZZZ", damped = NULL, alpha = NULL, beta = NULL, gamma = NULL, phi = NULL, lambda = NULL, biasadj = FALSE, additive.only = FALSE, restrict = TRUE, allow.multiplicative.trend = FALSE

data.ts

The time series to be forecast

model

A three-letter code indicating the model to be estimated using the ETS classification and notation. The possible inputs are “N” for none, “A” for additive, “M” for multiplicative, or “Z” for automatic selection. If any of the inputs is left as “Z”, then this component is selected according to the information criterion. The default value of ZZZ nsures that all components are selected using the information criterion.

Damped

If damped = TRUE, then a damped trend will be used (either A or M). If damped = FALSE, then a non-damped trend will used. If damped = NULL, (the default), then either a damped or a non-damped trend will be selected, depending on which model has the smallest value for the information criterion.

alpha, beta, gamma, phi

The values of the smoothing parameters can be specified using these arguments. If they are set to Null (the default setting for each of them), the parameters are estimated.

lambda

Box-Cox\* transformation parameter. It will be ignored if lambda = NULL (the default value). Otherwise, the time series will be transformed before the model is estimated. When lambda is not NULL, additive.only is set to TRUE.

biasadj

If TRUE and lambda is not NULL, then the back-transformed fitted values and forecasts will be bias-adjusted.

additive.only

Only models with additive components will be considered if additive.only = TRUE. Otherwise, all models will be considered.

restrict

If restrict = TRUE (the default), the models that cause numerical difficulties are not considered in model selection.

allow.multiplicative.trend

Set this argument to TRUE to allow mutilicative models to be considered.

\* Box-Cox

A Box Cox transformation is a way to transform non-normal dependent variables into a normal shape. Normality is an important assumption for many statistical techniques; if your data isn’t normal, applying a Box-Cox means that you are able to run a broader number of tests.

## Selecting predictors

When there are many possible predictors, we need some strategy for selecting the best predictors to use in a regression model.

A common approach that is not recommended is to plot the forecast variable against a particular predictor and if there is no noticeable relationship, drop that predictor from the model. This is invalid because it is not always possible to see the relationship from a scatterplot, especially when the effects of other predictors have not been accounted for.

Another common approach which is also invalid is to do a multiple linear regression on all the predictors and disregard all variables whose p-value are greater than 0.05. (we saw this in our data mining course. We picked the ones with 3 starts!). To start with, statistical significance does not always indicate predictive value. Even if forecasting is not the goal, this is not a good strategy because the p-value can be misleading when two or more predictors are correlated with each other.

Instead, we will use a measure of predictive accuracy. Five such measures are introduced in this section. They can be calculated using the cv( ) function. This function returns 5 indicators; CV, AIC, AICc, BIC, and AdjR2.

We compare these values against the corresponding values from other models. For the CV, AIC, AICc and BIC measures, we want to find the model with the lowest value; but for *Adjusted R2,* we seek the model with the highest value.

# Adjusted R2

The R2 and SSE (Sum of Squared Error), are not good evaluating measures when it comes to prediction and forecasting. An alternative which is designed to overcome their problems is the adjusted R2 Also called R-bar-squared)

where N is the number of observations and *p* is the number of predictors

# Cross-Validation]

Cross Validation determines the predictive ability of a model. The procedure of calculating CV uses the following steps:

1. Remove observation t from the data set, and fit the model using the remaining data. Then compute the error for the omitted observation. (This is not the same as the residual because the tth observation was not used in estimating the value of
2. Repeat step 1 for t = 1, …, N
3. Compute the MSE from

# Akaike’s Information Criterion

The Akaike’s Information Criterion, AIC, defined as:

where N is the number of observations used for estimation and *p* s the number of predictors in the model.

# Corrected Akaike’s Information Criterion

For small values of N, the AIC tends to select too many predictors, and so a bias-corrected version of the AIC has been developed,

As with the AIC, the AICc should be minimized.

For now we ignore BIC which is “Schwarz’s Bayesian Information Criterion”

# Model selection

A great advantage of the ETS statistical framework is that information criteria can be used for model selection. The AIC, AICs explained in the above section can be used here to determine which of the ETS models is most appropriate for a given time series.

For ETS models, Akaike’s Information Criterion (AIC) is defined as

Where L is the likelihood of the model and p is the total number of parameters and initial states that have been estimated (including the residual variance)

The AIC corrected for small sample bias (AICc) is defined as

And Bayesian Information Criterion, BIC, is

Complex seasonality – Multiple Seasonal Time Series

So far, we have considered relatively simple seasonal patterns such as quarterly and monthly data. However, higher frequency time series often exhibit more complicated seasonal patterns. For example, daily data may have a weekly pattern as well as an annual pattern. Hourly data usually has three types of seasonality: a daily pattern, a weekly pattern, and an annual pattern. Even weekly data can be challenging to forecast as it typically has an annual pattern with seasonal period of 365.25/7 ≈ 52.179 on average.

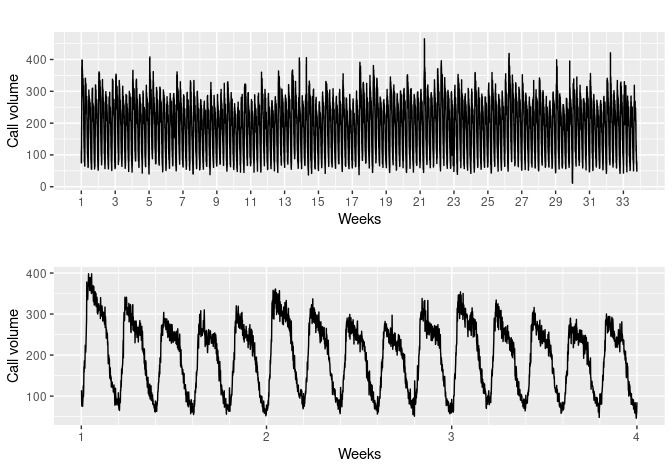
Such multiple seasonal patterns are becoming more common with high frequency data recording systems. Additional examples when multiple seasonal patterns can occur, include call volume in call centers, daily hospital admissions, requests for cash at ATMs, electricity and water usage, and visit to the Internet web sites.

Most of the methods we have considered so far are unable to deal with these seasonal complexities. Even the *ts()* class functions in R can only handle one type of seasonality, which is usually assumed to take integer values (round number of occurrences such as the number of months in a year)

To deal with such series, we will use the *msts()* class functions which handles multiple seasonality time series. This allows you to specify all of the frequencies that might be relevant. It is also flexible enough to handle non-integer frequencies such as number of weeks per year.

Despite this flexibility, we don’t necessarily want to include all of these frequencies — just the ones that are likely to be present in the data. For example, if we have only 180 days of observations, we may ignore the annual seasonality. If the data are measurements of a natural phenomenon (e.g., temperature), we can probably safely ignore any weekly seasonality.

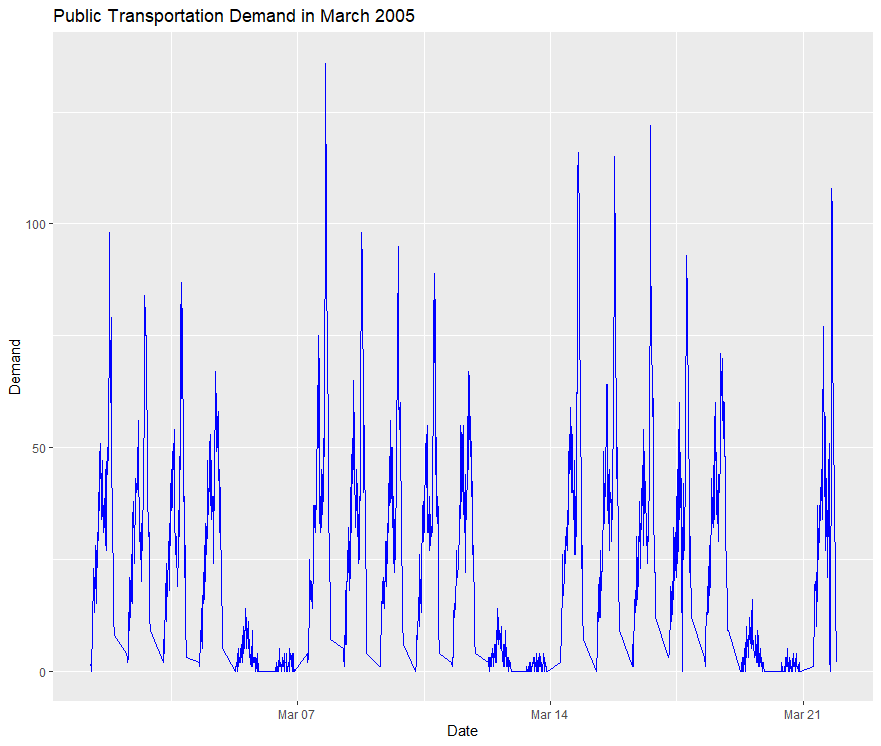
The top panel in the following Figure shows the number of retail banking call arrivals per 5-minute interval between 7:00am and 9:05pm each weekday over a 33 week period. The bottom panel shows the first three weeks of the same time series. There is a strong daily seasonal pattern with frequency 169 (there are 169 5-minute intervals per day), and a weak weekly seasonal pattern with frequency 169 × 5 = 845.

(Call volumes on Mondays tend to be higher than the rest of the week.) If a longer series of data were available, we may also have observed an annual seasonal pattern.

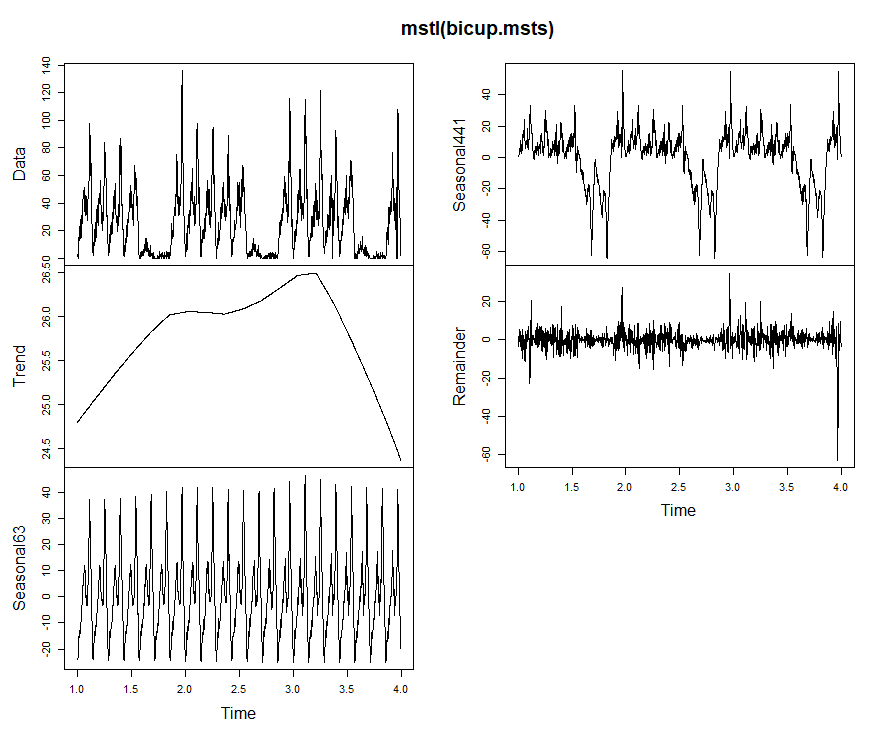
# STL with multiple seasonal periods

The mstl() function is a variation on stl() designed to deal with multiple seasonality. It will return multiple seasonal components, as well as a trend and remainder component.

The following R codes plots are what I used to decompose the two-seasonal data in the public transportation usage recorded 63 times recorded daily. Starting at 6:30AM to 10:00 PM for 21 days (3 weeks). The total of 441 observations in a week.



*bicup.msts <-msts(bicup.df$DEMAND, seasonal.periods = c(63, 441))*

*plot(mstl(bicup.msts))*

AS you see the mstl() allows us clearly the weekly and daily seasonality.

The decomposition can also be used in forecasting, with each of the seasonal components forecast using a seasonal naïve method, and the seasonally adjusted data forecasting using ETS (or some other user-specified method). The stlf() function will do this automatically.

I used msts() for building weekly and daily time series. The seasonal period for weekly was:

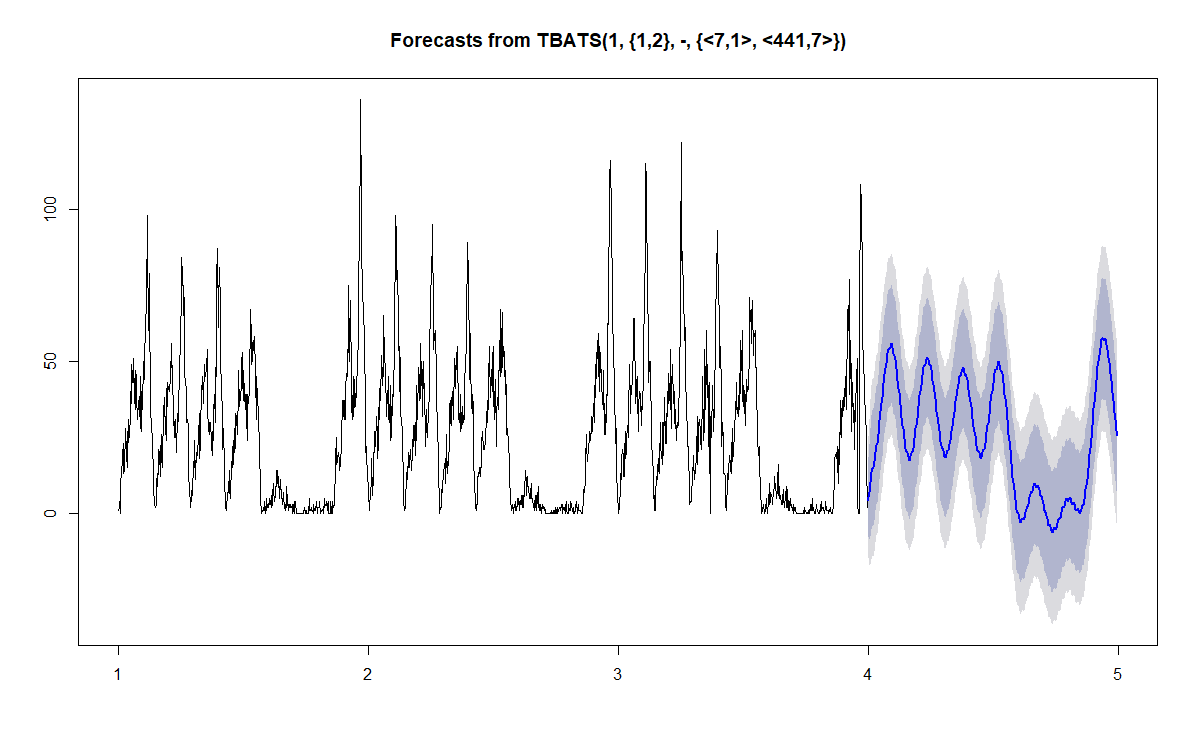
*bicup.msts <-msts(bicup.df$DEMAND, seasonal.periods = c(7, 63))*

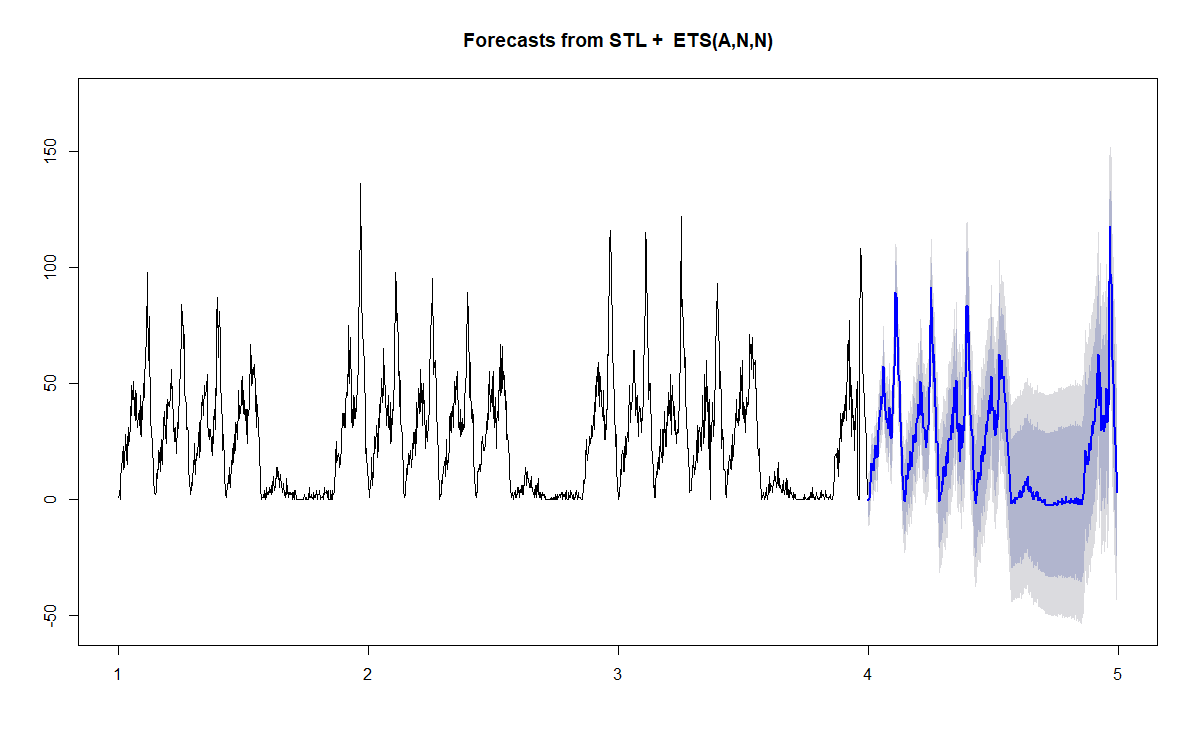
and daily period

*bicup.msts <-msts(bicup.df$DEMAND, seasonal.periods = c(63, 441))*

Then I used tbats() as well as *stlm()* to build my models and forecast for new 7 days.

Figures below shows the forecast with *tbats()* and *stlm()* < (STL+ETS)>





# Multiple Seasonal Dataset: weekly and yearly Forecasting

Below I summarized all said in the above regarding multi-seasonality and the I put codes for textbook exercise in pages 104-105

**Daily Forecasting**

Unless the time series is very long, the easiest approach is to simply set the frequency attribute to 7. The R code for is simply

*y <- ts(x, frequency=7)*

Then any of the usual time series forecasting methods should produce reasonable forecasts. For example

*library(forecast)*

*fit <- ets(y)*

*fc <- forecast(fit)*

*plot(fc)*

When the time series is long enough to take in more than a year, then it may be necessary to allow for annual seasonality as well as weekly seasonality. In that case, a multiple seasonal model such as TBATS is required.

*y <- msts(x, seasonal.periods=c(7,365.25))*

*fit <- tbats(y)*

*fc <- forecast(fit)*

*plot(fc)*

This should capture the weekly pattern as well as the longer annual pattern. The period 365.25 is the average length of a year allowing for leap years. The course textbook has more options for multiple seasonal R forecasting functions such as double seasonal Holt Winter (dshw) or STL+ETS (stml). I used the *tunneltrafic.csv* dataset to generate weekly and daily forecast. In the following pages are my R-codes and some plotting

*#Install required packages*

*install.packages("ggfortify", lib="C:/Users/Public/R-3.6.1/library")*

*install.packages("lubridate", lib="C:/Users/Public/R-3.6.1/library")*

*#activalte required library*

*library(forecast)*

*library(ggfortify)*

*library(lubridate)*

*library(ggplot2)*

*library(readxl)*

*Traffic<-read\_xlsx("TunnelTraffic.xlsx")*

*T.msts <- msts(Traffic$`# Vehicles in tunnel`, seasonal.periods=c(7,365.25))*

*#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ TBATS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*T.msts.tbats<-tbats(T.msts)*

*fc.msts.tbats.y<-forecast(T.msts.tbats, h = 365)*

*plot(fc.msts.tbats.y$model)*

*plot(fc.msts.tbats.y)*

*month<-4\*7*

*fc.msts.tbats.w<-forecast(T.msts.tbats, h = month)*

*plot(fc.msts.tbats.w)*

*##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ STL+ETS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*T.msts.stlm <- stlm(T.msts, s.window = "periodic", method = "ets")*

*fc.msts.stlm.y<- forecast(T.msts.stlm, h = 365)*

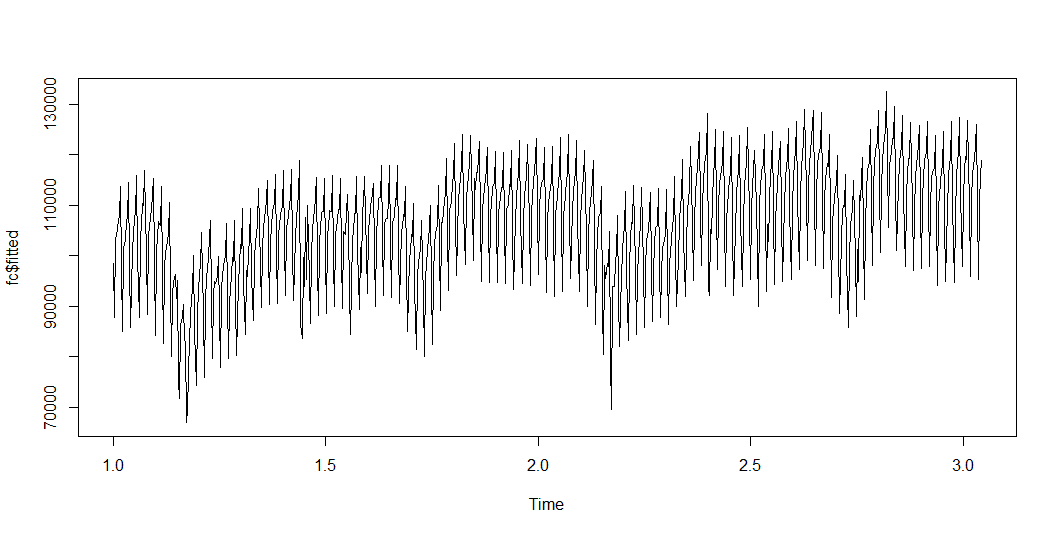
*plot(fc.msts.stlm.y)*

*##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ETS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

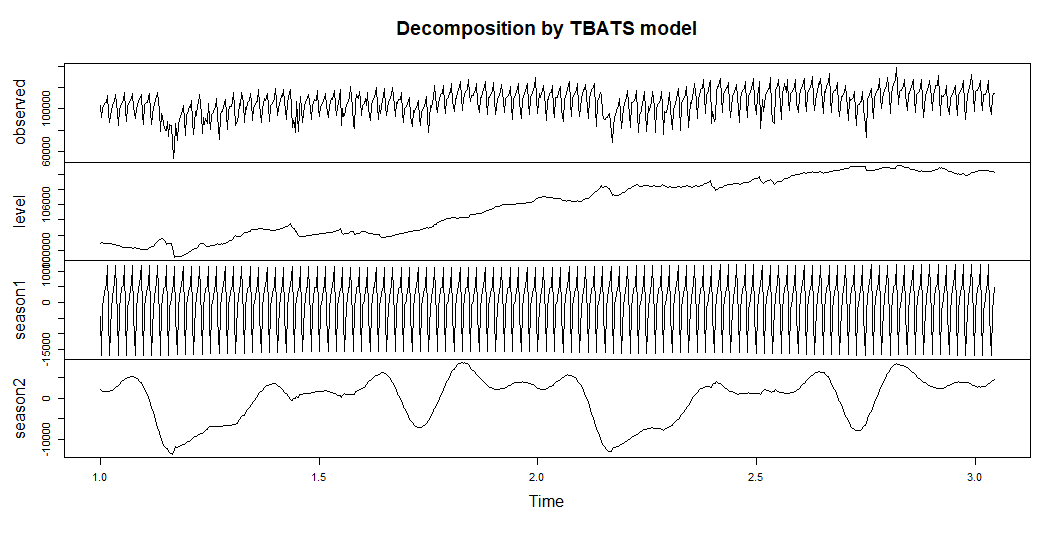
*##using function from lubridate to get the proper start time for a daily series and*

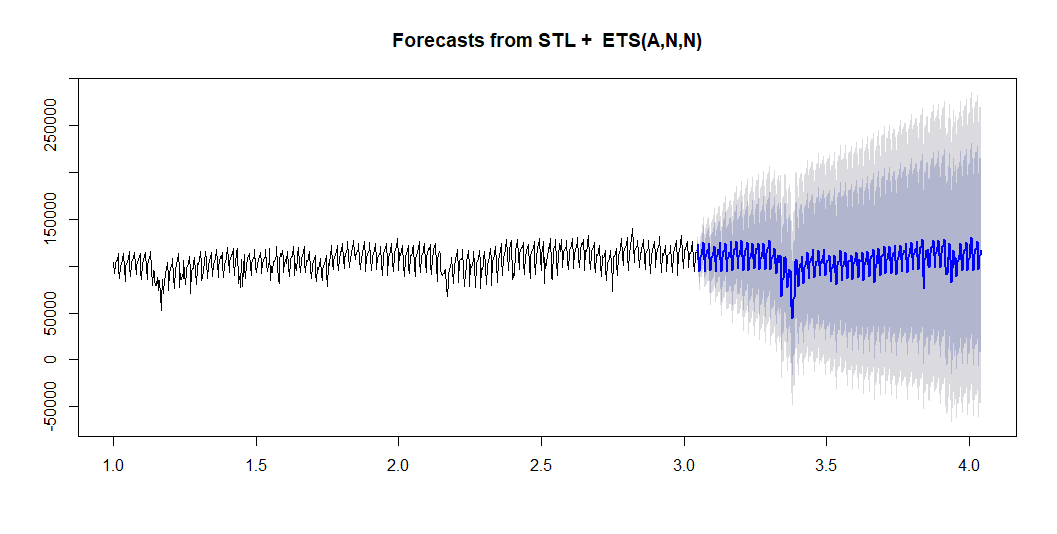
*##assuming that the vector of values you want to index as a ts is called x and is of the proper length*

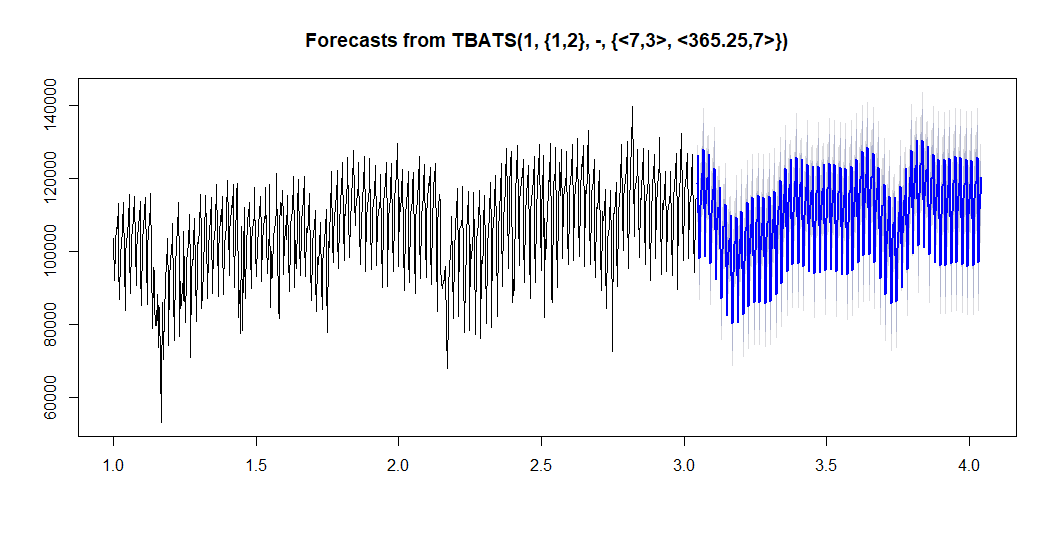
*df <- ts(Traffic$Traffic$`# Vehicles in tunnel`, start = decimal\_date(as.Date("1-11-2003")), frequency = 365)*

*plot(df)*

See the rest of plots of forecasting for year in the next page







1. Chapter 6 – Regression-Based Models

Regression models for capturing trend

Linear regression is a popular forecasting method, using suitable predictors to capture trend and/or seasonality as well as other patterns. This week notes shows how a linear regression model can be set up to capture different types of trend in a time series. The model, which is estimated from the training period, can then produce forecasts on future data by inserting the relevant predictor information into the estimated regression equation.

We describe different types of regression trend models for capturing common trend shapes (linear, exponential, and polynomial). Linear regression can be used to fit a global trend that applies to the entire series and will apply in the forecasting period. A linear trend means that the values of the series increase or decrease linearly in time, whereas an exponential trend captures an exponential increase or decrease. We can also use more flexible functions, such a quadratic functions or higher order polynomials, to capture more complex trend shapes.

## Linear Trend

To create a linear regression model that captures a time series with a global linear trend, the output variable (y) is set as the time series measurement or some function of it, and the predictor (x) is set as a time index t. Let us consider Amtrak ridership example: fitting a linear trend to the Amtrak ridership data. The first step is to create a new column that is a time index t = 1,2,3, … This will serve as our predictor. Here is a snapshot of the first few rows for the two corresponding columns (y and t). As you remember from previous chapters of our textbook, *a seasonal pattern in a time series means that observations that fall in some seasons have consistently higher or lower values than those that fall in other seasons*. Examples are day-of-week patterns, monthly patterns, and quarterly patterns. The Amtrak ridership monthly time series, as can be seen in the figure 1 time plot, exhibits strong monthly seasonality (with highest traffic during summer months).

### Table 1: Amtrak Ridership

|  |  |  |
| --- | --- | --- |
| Month | Ridership | t (Time Index) |
| Jan-91 | 1709 | 1 |
| Feb-91 | 1621 | 2 |
| Mar-91 | 1973 | 3 |
| Apr-91 | 1812 | 4 |
| May-91 | 1975 | 5 |
| Jun-91 | 1862 | 6 |
| Jul-91 | 1940 | 7 |
| Aug-91 | 2013 | 8 |
| Sep-91 | 1596 | 9 |
| Oct-91 | 1725 | 10 |
| Nov-91 | 1676 | 11 |
| Dec-91 | 1814 | 12 |
| Jan-92 | 1615 | 13 |
| Feb-92 | 1557 | 14 |

Before fitting the linear regression, we partition the ridership time series into training and validation periods. Here we keep the last 3 years of data as the validation period. Next, to fit a linear relationship between Ridership and Time, we set the output variable (y) as the Amtrak ridership and the predictor as the time index *t* in the regression model:

Where  is the Ridership at time point *t* and ε is the standard noise term in a linear regression. Thus, we are modeling three of the four time-series components: level (β0), trend (β1), and noise (ε). Seasonality is not modeled.

The next step is to use the linear trend model (also a linear regression model) to make forecasts in the validation period. The next page figure (Figure 1), depicts the actual time series, and the linear model predictions in the training and validation periods.

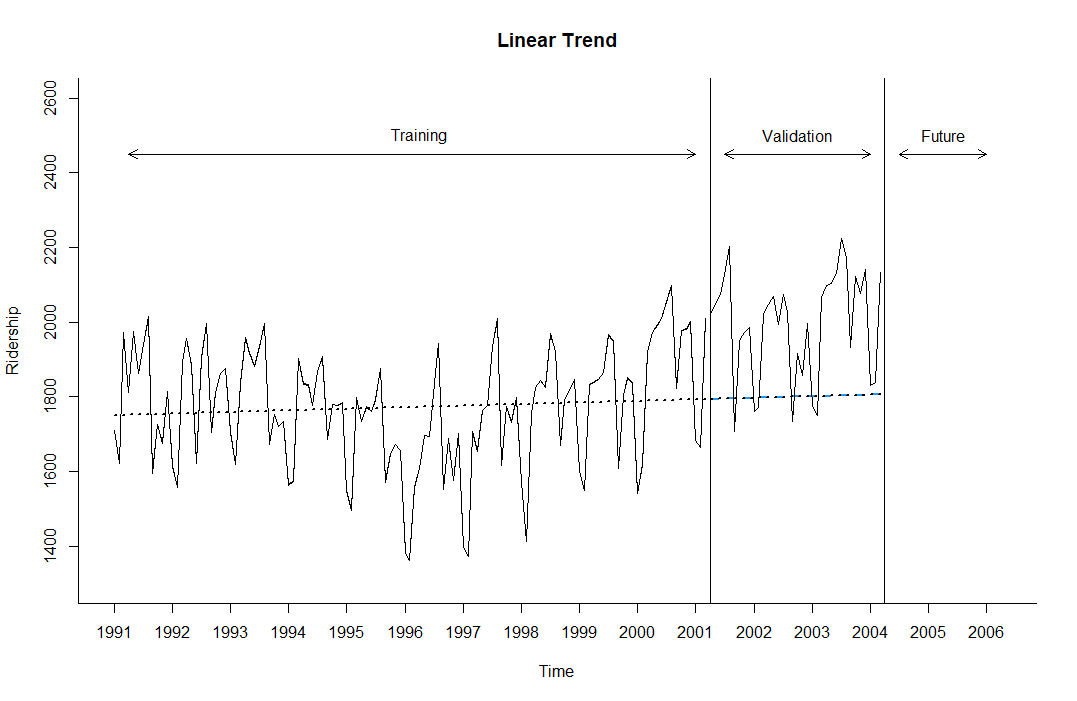


Figure 1: Amtrak ridership series with linear trend model predictions for training (smooth black dashed line) and validation (dashed blue line)

### The following R Codes are used in figure 6.3 on page 120.

### The following R code fits a linear regression model with a linear trend, and plots the predicted values for the training and validation sets overlaid on the original series

Amtrak.df<-read.csv("Amtrak data W date.csv")

library(forecast)

library(zoo)

library(knitr)

Amtrak.ts <- ts(Amtrak.df$Ridership, start = c(1991, 1), end = c(2004, 3), freq = 12)

plot(Amtrak.ts, main = "Amtrak Time Series")

plot(BoxCox(Amtrak.ts, 0))

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#Time Series Data Partitioning

n.valid <- 36

n.train <- length(Amtrak.ts) - n.valid

train.ts <- window(Amtrak.ts, start = c(1991, 1), end = c(1991, n.train))

valid.ts <- window(Amtrak.ts, start = c(1991, n.train + 1), end = c(1991, n.train + n.valid))

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#Linear Trend prediction

train.lm.linear<-tslm(train.ts ~ trend, lambda = 1)

summary(train.lm)

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1750.3595 29.0729 60.206 <2e-16 \*\*\*

trend 0.3514 0.4069 0.864 0.39

#Forecast with linear trend over validation

valid.lm.linear.pred <- forecast(train.lm.linear, h = n.valid, level = 0)

##Full Plot of the Amtrak time series with Liner trend and forecast into validation

plot(valid.lm.linear.pred, ylim = c(1300, 2600), ylab = "Ridership", xlab = "Time", bty = "l",

xaxt = "n", xlim = c(1991,2006.25), main = "Linear Trend", flty = 2)

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

lines(valid.lm.linear.pred$fitted, lwd = 2, col = "black", lty = 3)

lines(valid.lm.linear.pred$mean, lwd = 2, col = "black", lty = 3)

lines(train.ts)

lines(valid.ts)

lines(c(2004.25 - 3, 2004.25 - 3), c(0, 3500))

lines(c(2004.25, 2004.25), c(0, 3500))

text(1996.25, 2500, "Training")

text(2002.75, 2500, "Validation")

text(2005.25, 2500, "Future")

arrows(2004 - 3, 2450, 1991.25, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5 - 3, 2450, 2004, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)arrows(2004.5, 2450, 2006, 2450, code = 3, length = 0.1, lwd = 1, angle = 30)

**Note 1**: beware of examining only the coefficients and their statistical significance for making decisions about the trend, can be misleading. A significant coefficient for trend does not mean that a linear fit is adequate. An insignificant coefficient does not mean that there is no trend in the data. In our example, the slope coefficient (0.3514) is insignificant (p- value=0.39), yet there may be a trend in the data (often once we control for seasonality). To determine suitability of any trend shape, look at the time plot of the (de-seasonalized) time series with the trend overlaid; examine the residual time plot; and look at performance measures on the validation period.

## Exponential Trend

Several alternative trend shapes are useful and easy to fit via a linear regression model. Recall Excel’s Trend line and other plots that help to assess the type of trend in the data. One such shape is an exponential trend. An exponential trend implies a multiplicative increase/decrease of the series over time  . Exponential trends are popular in sales data, where they reflect percentage growth.

which is equal to

Then we call and we know

Therefore, to fit an exponential trend, simply replace the output variable y with log(y) and fit the linear regression:

In the Amtrak example, for instance, we would fit a linear regression of log(*Ridership*) as the dependent variable and the index variable t as the predictor.

The R code for fitting this model is:

train.lm.expo.trend <- tslm(train.ts ~ trend, lambda = 0)

[In R, setting lambda = 0 in the function tslm() indicates an exponential trend. For a linear trend, we set lambda=1, which is the default.]

**Note 2**: We use “log” to denote the natural logarithm (base e = 2.71828…). Excel uses the function =LN. In R, use the function log (). I use mathematical version of it which is

**Note 3**: As in the general case of linear regression, when comparing the predictive accuracy of models that have a different output variable, such as comparing a linear model trend (with y) and an exponential model trend (with log(y)), it is essential to compare forecast or forecast errors on the same scale. An exponential trend model will produce forecasts in logarithmic scale, and the forecast errors reported by many software packages (e.g. XLMiner) will therefore be of the form

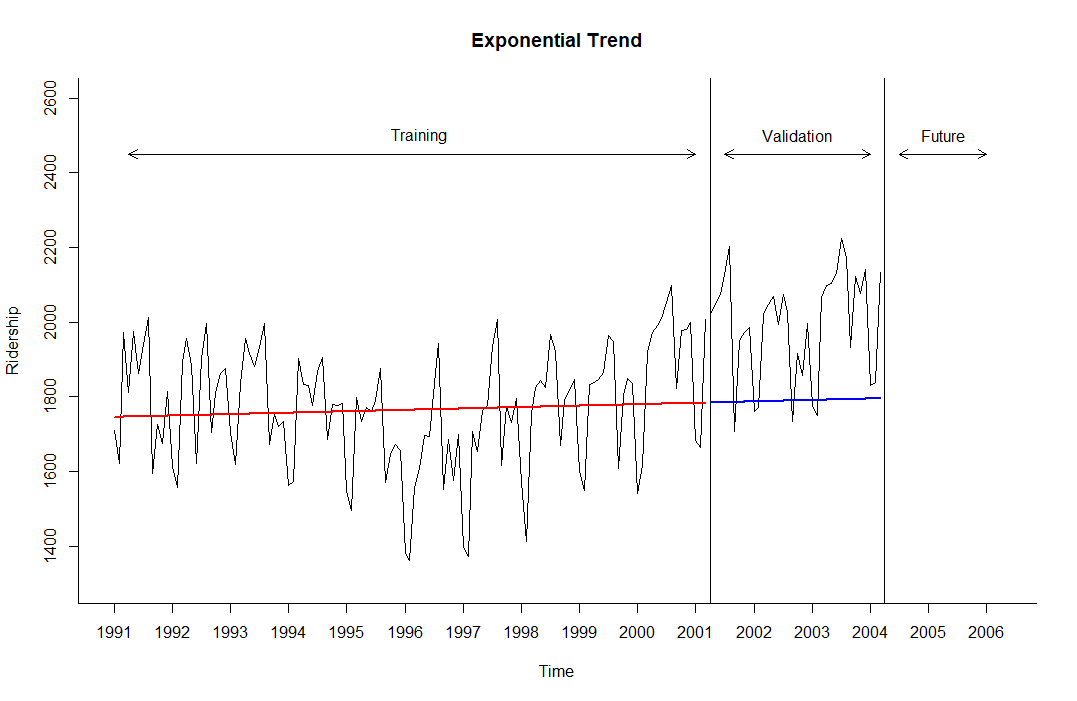
Figure (Figure 2), depicts the actual time series, and the exponential model predictions the training and validation periods.

Figure 2: Amtrak ridership series with exponential trend model (presented as log(y)) predictions for training

(smooth red line) and validation (blue line)

### The following R Codes are used in figure 6.4 on page 123.

### The following R code fits a exponential regression model showed as a linear trend, and plots the predicted values for the training and validation sets overlaid on the original series

# Exponential Trend Prediction

train.lm.expo <- tslm(train.ts ~ trend, lambda = 0)

valid.lm.expo.pred <- forecast(train.lm.expo, h = n.valid, level = 0)

summary(train.lm.expo)

'#

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 7.4646979 0.0168680 442.535 <2e-16 \*\*\*

trend 0.0001783 0.0002361 0.755 0.451

#'

##Full Plot of the Amtrak time series with Exponential trend and forecast into validation

plot(valid.lm.expo.pred, ylim = c(1300, 2600), ylab = "Ridership", xlab = "Time", bty = "l",

xaxt = "n", xlim = c(1991,2006.25), main = "Exponential Trend", flty = 2)

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

lines(valid.lm.expo.pred$fitted, lwd = 2, col = "red") # Training Side

lines(train.ts)

lines(valid.ts)

lines(c(2004.25 - 3, 2004.25 - 3), c(0, 3500))

lines(c(2004.25, 2004.25), c(0, 3500))

text(1996.25, 2500, "Training")

text(2002.75, 2500, "Validation")

text(2005.25, 2500, "Future")

arrows(2004 - 3, 2450, 1991.25, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5 - 3, 2450, 2004, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

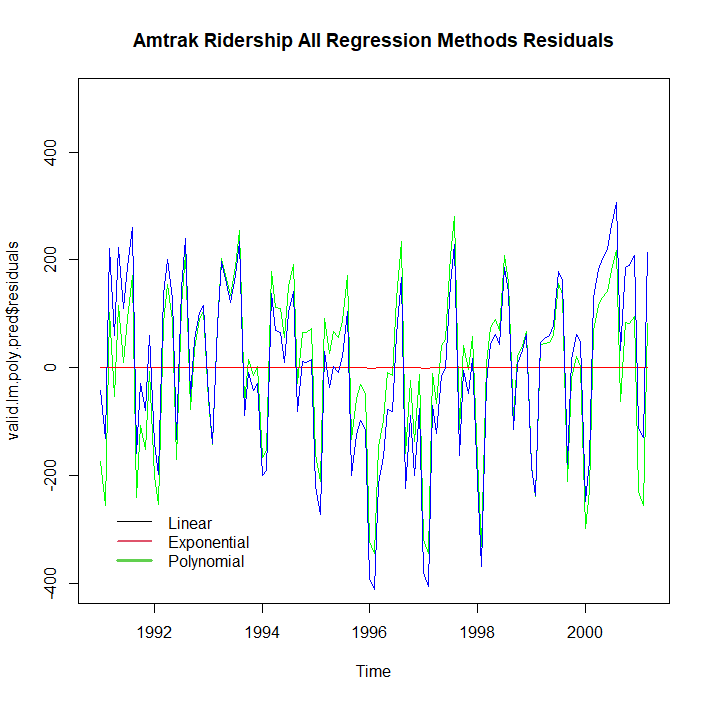
arrows(2004.5, 2450, 2006, 2450, code = 3, length = 0.1, lwd = 1, angle = 30)

## Polynomial Trend

Another nonlinear trend shape that is easy to fit via linear regression is a polynomial trend, and in particular, a quadratic relationship of the form:

This is done by creating an additional predictor *t2* (the square of t) and fitting a multiple linear regression with the two predictors *t* and t2. For the Amtrak ridership data, we have already seen a U-shaped trend in the data. We therefore fit a quadratic model to the training period. In R we can do this using the code:

train.lm.poly.trend <- tslm(train.ts ~ trend + I(trend^2))



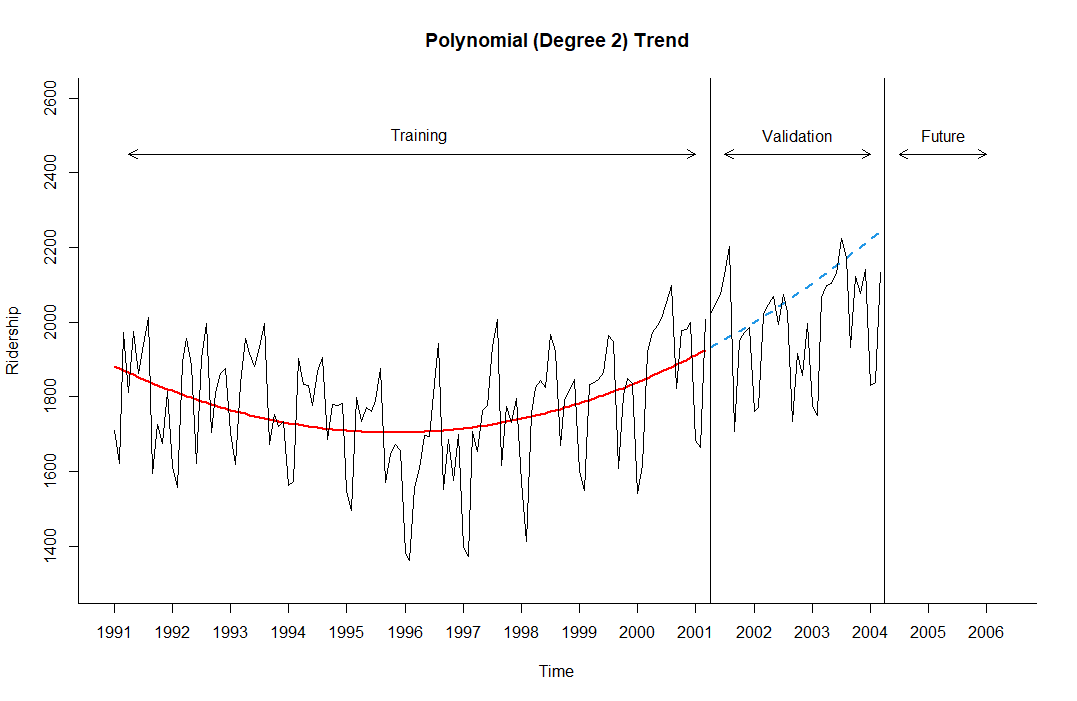
Figure 3 and 4 are the pair of plots for the quadratic fit (Figure 3: actual + forecasts; Figure 3: forecast errors). We can see that this shape captures the pattern in the trend in the training period. The forecast errors now exhibit only seasonality and no trend period. The forecast residuals exhibit only seasonality and no trend

Figure 3: Amtrak ridership series with quadratic Figure 4: forecast residual for linear, exponential and Polynomial

trend model: actual and forecast

The following R Codes are used in figure 3 and 4. (polynomial trend)

# Polynomial Trend Prediction

train.lm.poly <- tslm(train.ts ~ trend + I(trend^2))

valid.lm.poly.pred <- forecast(train.lm.poly, h = n.valid, level = 0)

summary(train.lm.poly)

'#

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1888.88401 40.91521 46.166 < 2e-16 \*\*\*

trend -6.29780 1.52327 -4.134 6.63e-05 \*\*\*

I(trend^2) 0.05362 0.01190 4.506 1.55e-05 \*\*\*

#'

##Full Plot of the Amtrak time series with Exponential trend and forecast into validation

plot(valid.lm.poly.pred, ylim = c(1300, 2600), ylab = "Ridership", xlab = "Time", bty = "l",

xaxt = "n", xlim = c(1991,2006.25), main = "Polynomial (Degree 2) Trend", flty = 2)

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

lines(valid.lm.poly.pred$fitted, lwd = 2, col = "red") # Training Side

lines(train.ts)

lines(valid.ts)

lines(c(2004.25 - 3, 2004.25 - 3), c(0, 3500))

lines(c(2004.25, 2004.25), c(0, 3500))

text(1996.25, 2500, "Training")

text(2002.75, 2500, "Validation")

text(2005.25, 2500, "Future")

arrows(2004 - 3, 2450, 1991.25, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5 - 3, 2450, 2004, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5, 2450, 2006, 2450, code = 3, length = 0.1, lwd = 1, angle = 30)

plot(valid.lm.poly.pred$residuals, ylim = c(-400,500), col = "darkgreen", main = "Amtrak Ridership All Regression Methods Residuals")

lines(valid.lm.expo.pred$residuals, col = "red")

lines(valid.lm.linear.pred$residuals, col = "blue")

legend(1991, -250, c("Linear", "Exponential", "Polynomial"), lty = 1, lwd = c(1,2,3), col = c(1,2,3), bty = "n")

**Note 3**: In general, any type of trend shape can be fit as long as it has a mathematical representation. However, the underlying assumption is that this shape is applicable throughout the period of data that we currently have as well as during the validation period and future. Do not choose an overly complex shape, because although it will fit the training period well, it will likely be overfitting the data. To avoid overfitting, always examine performance on the validation period and refrain from choosing overly complex trend patterns

Regression models for capturing seasonality

A seasonal pattern in a time series means that observations that fall in some seasons have consistently higher or lower values than those that fall in other seasons. Examples are day-of-week patterns, monthly patterns, and quarterly patterns. The Amtrak ridership monthly time series, as can be seen in the time plot, exhibits strong monthly seasonality (with highest traffic during summer months).

### Additive Seasonality

|  |  |  |
| --- | --- | --- |
| Month | Ridership | Season |
| Jan-91 | 1709 | Jan |
| Feb-91 | 1621 | Feb |
| Mar-91 | 1973 | Mar |
| Apr-91 | 1812 | Apr |
| May-91 | 1975 | May |
| Jun-91 | 1862 | Jun |
| Jul-91 | 1940 | Jul |
| Aug-91 | 2013 | Aug |
| Sep-91 | 1596 | Sep |
| Oct-91 | 1725 | Oct |
| Nov-91 | 1676 | Nov |
| Dec-91 | 1814 | Dec |
| Jan-92 | 1615 | Jan |
| Feb-92 | 1557 | Feb |
| Mar-92 | 1891 | Mar |
| Apr-92 | 1956 | Apr |
| May-92 | 1885 | May |

The most common way to capture seasonality in a regression model is by creating a new categorical variable that denotes the season for each observation. This categorical variable is then turned into dummy variables, which in turn are included as predictors in the regression model. To illustrate this, we created a new Month column for the Amtrak ridership data, as shown below (I show the first seventeen rows).

In order to include the season categorical variable as a predictor in a regression model for y (e.g., Ridership), we turn it into dummy variables. For m seasons, we create m-1 dummy variables, which are binary variables that take on the value 1 if the record falls in that particular season, and 0 otherwise. The mth season does not require a dummy, since it is identified when all the m-1 dummies take on zero values.

As with the trend models, before fitting the linear regression, we partition the ridership time series into training and validation periods. Here we keep the last 3 years of data as the validation period.

Next, to fit an additive seasonality model for Ridership. We set the output variable (y) as the Amtrak ridership and use the 11-month dummy variables as predictors. In R, the tslm() function automatically performs this dummy coding so we can directly use the column “season”:

Figure 5 illustrates the regression method model forecast when we consider only seasonality

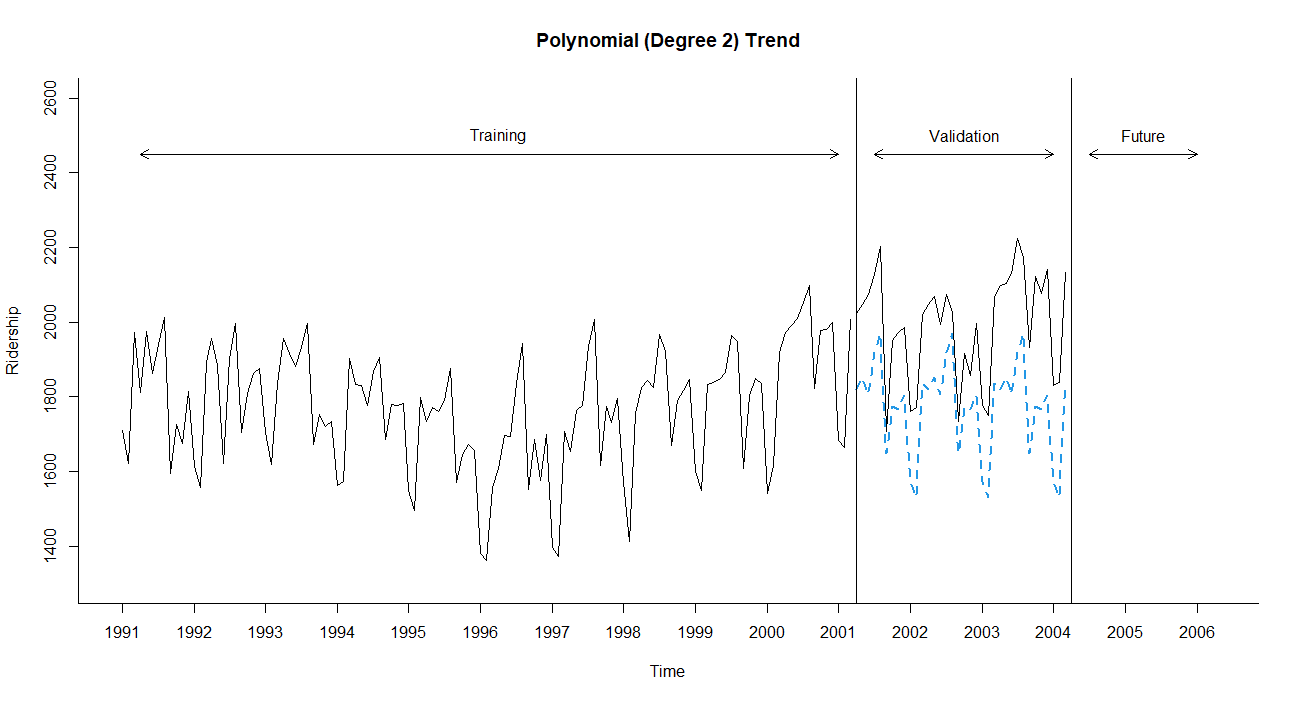


Figure 5: forecast for additive seasonality without considering trend

In the following regression function, we see that R fit the regression model with 11 dummies dropping season1 (January):

and we are modeling three of the four time-series components: level (β0), seasonality (β1 thru β11) and noise ε. Trend is not modeled.

### R-Codes for capturing seasonality without including trend

## Model with Additive seasonality wo trend

train.lm.season<-tslm(train.ts ~ season)

coefficients<-(train.lm.season$coefficients)

kable(coefficients)

'#

| | x|

|:-----------|----------:|

|(Intercept) | 1573.97218|

|season2 | -42.93018|

|season3 | 260.76773|

|season4 | 245.09192|

|season5 | 278.22222|

|season6 | 233.45982|

|season7 | 345.32652|

|season8 | 396.65952|

|season9 | 75.76152|

|season10 | 200.60762|

|season11 | 192.35522|

|season12 | 230.41512|

#'

summary(train.lm.season)

'#

Call:

tslm(formula = train.ts ~ season)

Residuals:

Min 1Q Median 3Q Max

-276.165 -52.934 5.868 54.544 215.081

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1573.97 30.58 51.475 < 2e-16 \*\*\*

season2 -42.93 43.24 -0.993 0.3230

season3 260.77 43.24 6.030 2.19e-08 \*\*\*

season4 245.09 44.31 5.531 2.14e-07 \*\*\*

season5 278.22 44.31 6.279 6.81e-09 \*\*\*

season6 233.46 44.31 5.269 6.82e-07 \*\*\*

season7 345.33 44.31 7.793 3.79e-12 \*\*\*

season8 396.66 44.31 8.952 9.19e-15 \*\*\*

season9 75.76 44.31 1.710 0.0901 .

season10 200.61 44.31 4.527 1.51e-05 \*\*\*

season11 192.36 44.31 4.341 3.14e-05 \*\*\*

season12 230.42 44.31 5.200 9.18e-07 \*\*\*

the code for capturing seasonality provides reasonable result if there is no seasonality but if time series data include a seasonality, then we have to add proper type of season. We already know the polynomial of second degree is the proper trend type for our dataset therefore, we add it to the model. Figure 6 captures this addition.

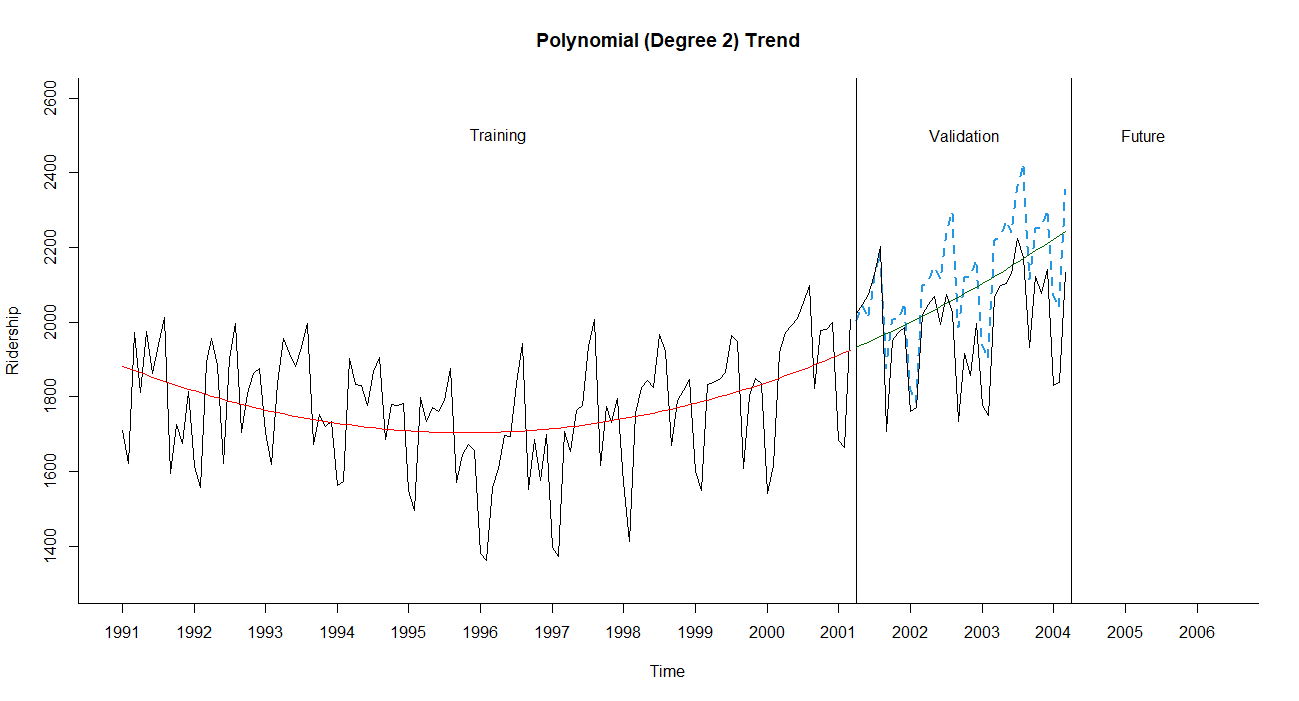


Figure 6: forecast for additive seasonality with polynomial of degree 2 trend

### R-Codes for capturing seasonality with trend

## Model with Additive Seasonaity and Trend

train.lm.trend.seasonality <- tslm(train.ts ~ trend + I(trend^2) + season)

coefficients<-(train.lm.trend.seasonality$coefficients)

kable(train.lm.trend.seasonality$coefficients)

'#

| | x|

|:-----------|-----------:|

|(Intercept) | 1696.979360|

|trend | -7.155851|

|I(trend^2) | 0.060744|

|season2 | -43.245842|

|season3 | 260.014920|

|season4 | 260.617456|

|season5 | 293.796560|

|season6 | 248.961476|

|season7 | 360.634004|

|season8 | 411.651344|

|season9 | 90.316197|

|season10 | 214.603661|

|season11 | 205.671137|

|season12 | 242.929425|

#'

summary(train.lm.trend.seasonality)

'#

tslm(formula = train.ts ~ trend + I(trend^2) + season)

Residuals:

Min 1Q Median 3Q Max

-213.775 -39.363 9.711 42.422 152.187

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.697e+03 2.768e+01 61.318 < 2e-16 \*\*\*

trend -7.156e+00 7.293e-01 -9.812 < 2e-16 \*\*\*

I(trend^2) 6.074e-02 5.698e-03 10.660 < 2e-16 \*\*\*

season2 -4.325e+01 3.024e+01 -1.430 0.15556

season3 2.600e+02 3.024e+01 8.598 6.60e-14 \*\*\*

season4 2.606e+02 3.102e+01 8.401 1.83e-13 \*\*\*

season5 2.938e+02 3.102e+01 9.471 6.89e-16 \*\*\*

season6 2.490e+02 3.102e+01 8.026 1.26e-12 \*\*\*

season7 3.606e+02 3.102e+01 11.626 < 2e-16 \*\*\*

season8 4.117e+02 3.102e+01 13.270 < 2e-16 \*\*\*

season9 9.032e+01 3.102e+01 2.911 0.00437 \*\*

season10 2.146e+02 3.102e+01 6.917 3.29e-10 \*\*\*

season11 2.057e+02 3.103e+01 6.629 1.34e-09 \*\*\*

season12 2.429e+02 3.103e+01 7.829 3.44e-12 \*\*\*

---

#'

### Multiplicative Seasonality

When seasonality is added as described above (create a categorical seasonal variable, then create dummy variables from it, and then regress on *yt*), it captures *additive seasonality*. This means that the average value of y in a certain season is higher or lower by a fixed amount compared to another season. For example, in the Amtrak ridership, the coefficient for August (396.66) indicates that the average number of passengers in August is higher by 396,660 passengers compared to the average in January (the reference category). Using regression models, we can also capture *multiplicative seasonality*, where average values on a certain season are higher or lower by a *fixed percentage* compared to another season. To fit multiplicative seasonality, we use the same model as above, except that we use *log(yt)*as the output variable. To do this in R, we include the argument lambda = 0 in the tslm() function

train.lm.season <- tslm(train.ts ~ season, lambda = 0)

Smoothly Transitioning Seasonality

When the seasonal pattern transitions smoothly from one season to the next, we can use continuous mathematical functions to approximate the seasonal pattern, such as including sinusoidal functions as predictors in the regression model. For example, the Centers for Disease Control and Prevention in the United States use a regression model for modeling the percent of weekly deaths attributed to pneumonia & influenza in 122 cities. The model includes a quadratic trend as well as sine and cosine functions for capturing the smooth seasonality pattern. In particular, they use the following regression model:

The trend terms *t* and *t2* accommodate long-term linear and curvilinear changes in the background proportion of pneumonia & influenza deaths arising from factors such as population growth or improved disease prevention or treatment (CDC data). The sine and cosine terms capture the yearly periodicity of weekly data (with 52.18\* weeks per year). This regression model is then fitted to five years of data to create a “baseline” against which new weekly mortality is compared, called the ’Serfling method’. To fit this type of model with linear, quadratic, sine, and cosine terms to the monthly Amtrak ridership data, we add two predictors to quadratic trend model. In R, the code would be:

\*\*tslm(train.ts ~ trend + I(trend^2) + I(sin(2\*pi\*trend/12)) + I(cos(2\*pi\*trend/12)))

Figure 6.8 (textbook page 131) shows how the seasonality is forecasted using sinusoidal function for the CDC data. I used the same method to forecast the Amtrak ridership. The figure 7 shows the forecast and below is forecast summary.

tslm(formula = train.ts ~ trend + I(sin(2 \* pi \* trend/12)) + I(cos(2 \* pi \* trend/12)))

Residuals:

Min 1Q Median 3Q Max

-318.34 -96.30 8.04 89.73 307.89

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1748.0335 25.0477 69.788 < 2e-16 \*\*\*

trend 0.4229 0.3507 1.206 0.23015

I(sin(2 \* pi \* trend/12)) -48.2666 17.5260 -2.754 0.00681 \*\*

I(cos(2 \* pi \* trend/12)) -106.5245 17.6773 -6.026 1.93e-08 \*\*\*

---

Signif. codes: 0 ë\*\*\*í 0.001 ë\*\*í 0.01 ë\*í 0.05 ë.í 0.1 ë í 1

Residual standard error: 137.9 on 119 degrees of freedom

Multiple R-squared: 0.2762, Adjusted R-squared: 0.2579

F-statistic: 15.14 on 3 and 119 DF, p-value: 2.088e-08

\*In every 4 years, there are 3 years of 365 days and 1 year 366. There are 52.14 weeks in the first three years and 52.28 weeks in the 4th year.

\*\*The R I() function would return a vector of values of the result of math function within parenthesis.

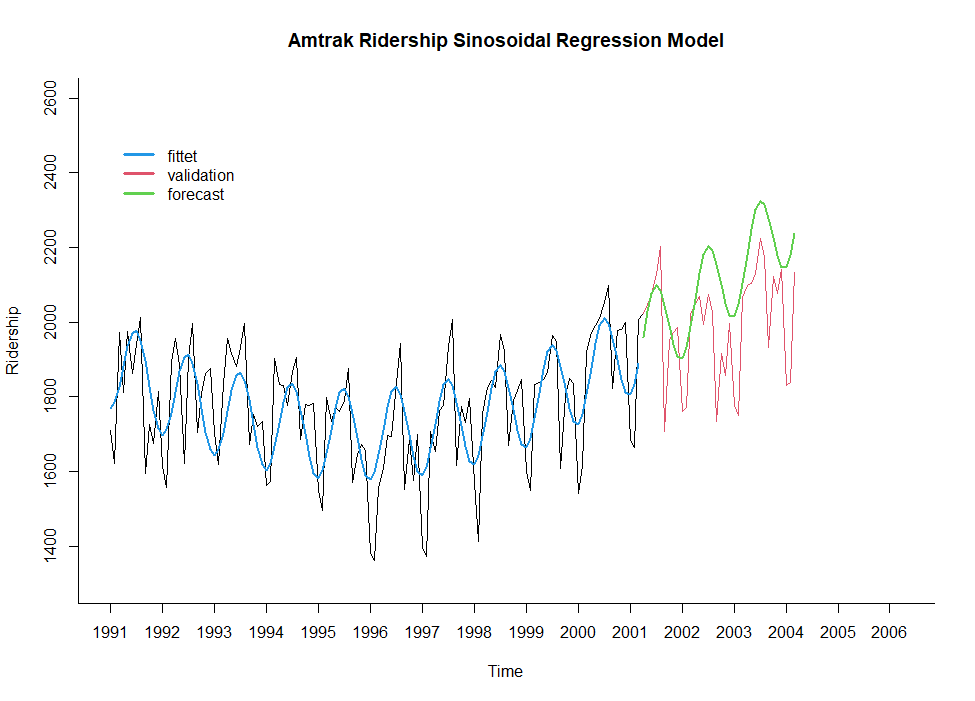


Figure 7: Amtrak forecast of the validation dataset using “Serfling method”.

### R Codes are used to generate figure 6-8

library(forecast)

library(zoo)

Amtrak.data <- read.csv("Amtrak data W date.csv")

Amtrak.ts <- ts(Amtrak.data$Ridership, start = c(1991, 1), end = c(2004, 3), freq = 12)

nValid <- 36

nTrain <- length(Amtrak.ts) - nValid

train.ts <- window(Amtrak.ts, start = c(1991, 1), end = c(1991, nTrain))

valid.ts <- window(Amtrak.ts, start = c(1991, nTrain + 1), end = c(1991, nTrain + nValid))

#Table 6.5

train.lm.trig <- tslm(train.ts ~ trend + I(trend^2) + I(sin(2\*pi\*trend/12)) + I(cos(2\*pi\*trend/12)))

train.lm.trig.pred <- forecast(train.lm.trig, h = nValid, level = 0)

summary(train.lm.trig)

#Similar to the 6-8 but with Amtrak datset

plot(Amtrak.ts, ylim = c(1300, 2600), ylab = "Ridership",

xlab = "Time", bty = "l", xaxt = "n", xlim = c(1991,2006.25),

main = "Amtrak Ridership Sinosoidal Regression Model", lwd = 1)

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

lines(train.lm.trig.pred$fitted, lwd = 2, col = 4)

lines(valid.ts, col = 2)

lines(train.lm.trig.pred$mean, lwd = 2, col = 3)

legend(1991, 2500, c("fittet", "validation", "forecast"), lty = 1, lwd = c(3,3,3), col = c(4,2,3), bty = "n")

'#

lines(c(2004.25 - 3, 2004.25 - 3), c(0, 3500))

lines(c(2004.25, 2004.25), c(0, 3500))

text(1996.25, 2500, "Training")

text(2002.75, 2500, "Validation")

text(2005.25, 2500, "Future")

arrows(2004 - 3, 2450, 1991.25, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5 - 3, 2450, 2004, 2450, code = 3, length = 0.1, lwd = 1,angle = 30)

arrows(2004.5, 2450, 2006, 2450, code = 3, length = 0.1, lwd = 1, angle = 30)

#'

##\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#Forecasting for 2 years into the future

Amtrak.lm.trig <- tslm(Amtrak.ts ~ trend + I(trend^2) + I(sin(2\*pi\*trend/12)) + I(cos(2\*pi\*trend/12)))

two.Years = 24

Amtrak.lm.trig.pred <- forecast(Amtrak.lm.trig, h = two.Years)

plot(Amtrak.lm.trig.pred, ylim = c(1300, 2600), ylab = "Ridership",

xlab = "Time", bty = "l", xaxt = "n", xlim = c(1991,2006.25),

main = "Amtrak Ridership Sinosoidal Regression Model", lwd = 1)

axis(1, at = seq(1991, 2006, 1), labels = format(seq(1991, 2006, 1)))

legend(1991, 2500, c("Time-Series", "forecast"), lty = 1, lwd = c(3,3), col = c(1,4), bty = "n")

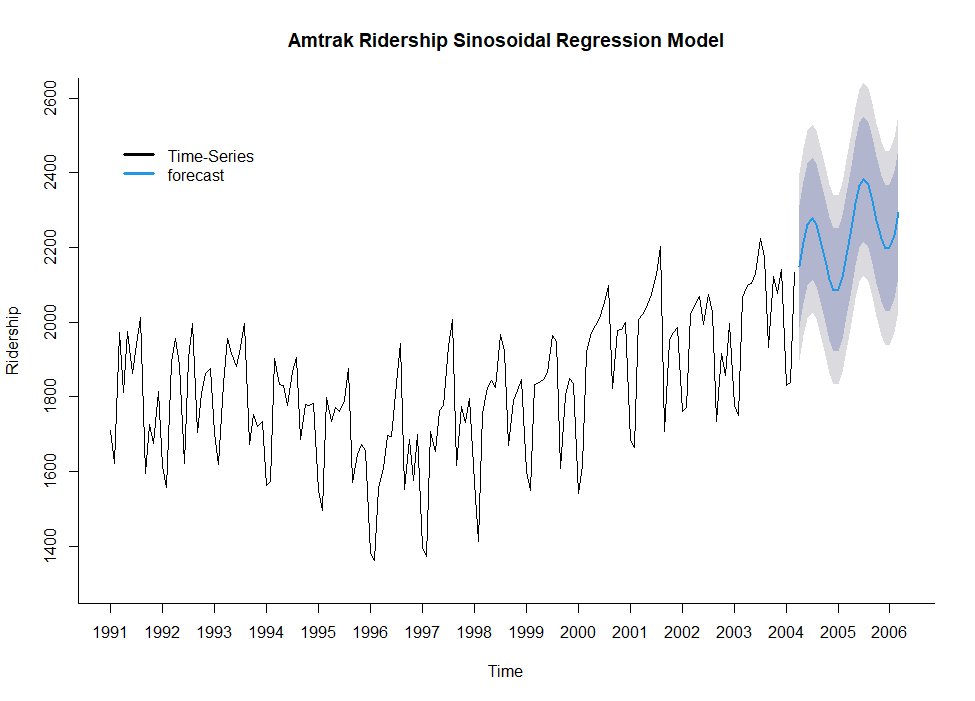


Figure 8: Amtrak future two-Year forecast using “Serfling method”.

### Regression function (regression model)

Regression mode is a mathematical function with predictors as input and estimated observation value as the output. In previous sections of this chapter we saw some of them. I will repeat them and then add those missed. The function has a general form but should be customized based on characteristics of the dataset.

1. Dataset with linear trend and no seasonality

Where

1. Dataset with quadratic trend and no seasonality

1. Dataset with linear trend and additive seasonality (assuming our season is M and we have N seasons)

Note: if we are forecasting an observation in a specific season say, M*i ,* only this season is one and the rest are all zeros

1. Dataset with quadratic trend and additive seasonality
2. Dataset with exponential trend and no seasonality

I will omit all types of multiplicative seasonality in this course.