# Homework 5

Amin Baabol

10/23/2020

## Question 1

The data in Table E4.4 exhibit a linear trend. a. Verify that there is a trend by plotting the data. b. Using the first 12 observations, develop an appropriate procedure for forecasting. c. Forecast the last 12 observations and calculate the forecast errors. Does the forecasting procedure seem to be working satisfactorily?

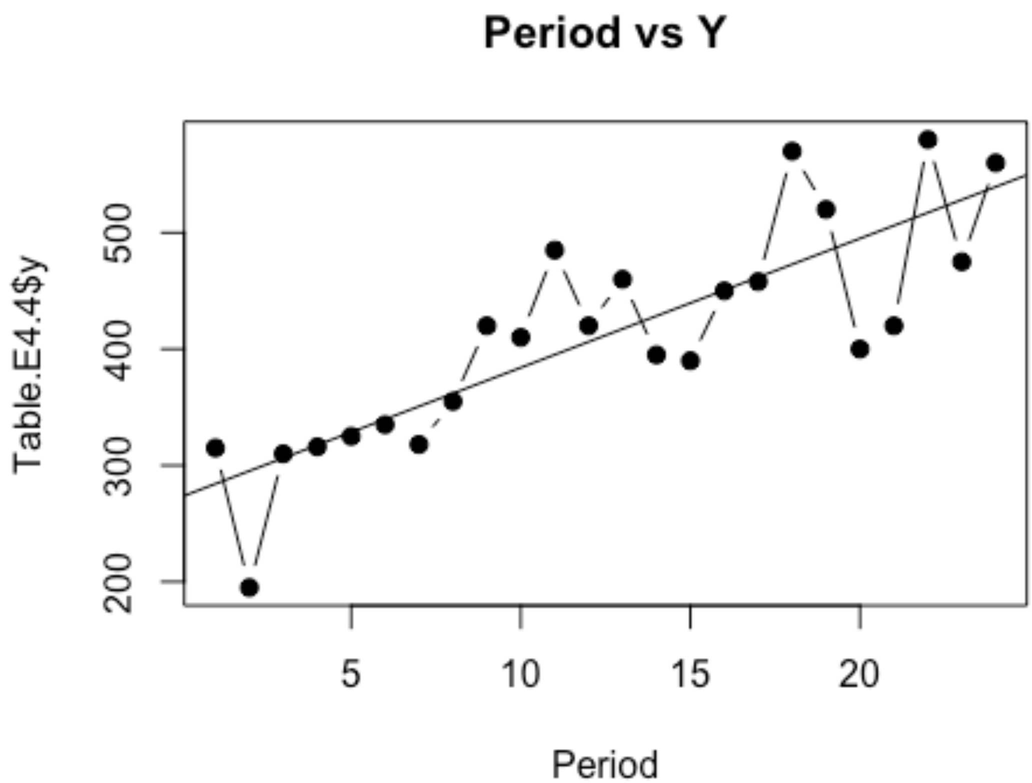
Further instructions from D2L:

For exercise 4.8, choose the optimum lambda by minimizing the SS_E() for a constant process given by Equation (4.29) on page 260 in the text. Also, plot the SS_E ve various similar to Figure 4.19 on page 264 in the text. For a) A time series plot is needed to verify the trend. b) Give and explain your forecast model. c) List the last 12 forecast values and the corresponding forecast errors. Then analyze the forecast errors by drawing a plot like Figure 4.24 on page 269.

## Discussion

a) The plot seems to have an increasing linear-like trend.
b) Since the plot of the data was linear-trend, I used the first order and second order exponential smoothing to find the optimal lambda which was 0.5 because it produced the lowest or minimum sse. I then I assumed that tau was 1 to make my y-hat prediction using the tau-step-ahead prediction method.
c) After forecasting the last 12 observations I created a new data frame with three columns. First column I stored the observed values, second column is the forecasted values and the third column is the error column. I calculated a mean absolute percentage error of 19.997% for my forecast error rate. I verified this error rate by computing again the MAPE using the accuracy function which gave me exactly 19.997 as well. This means my forecast accuracy is 80%, that is relatively satisfactory.

```
library("readxl")
Table.E4.4 <- read_excel("~/Desktop/GradSchool/Fall2020/STAT-
560/Homework/Table_E4_4.xlsx")

#part a
plot.ts(Table.E4.4$y,main = "Period vs Y", xlab = "Period",type = "b", pch =
19)
abline(reg = lm(Table.E4.4$y~time(Table.E4.4$y)))
```

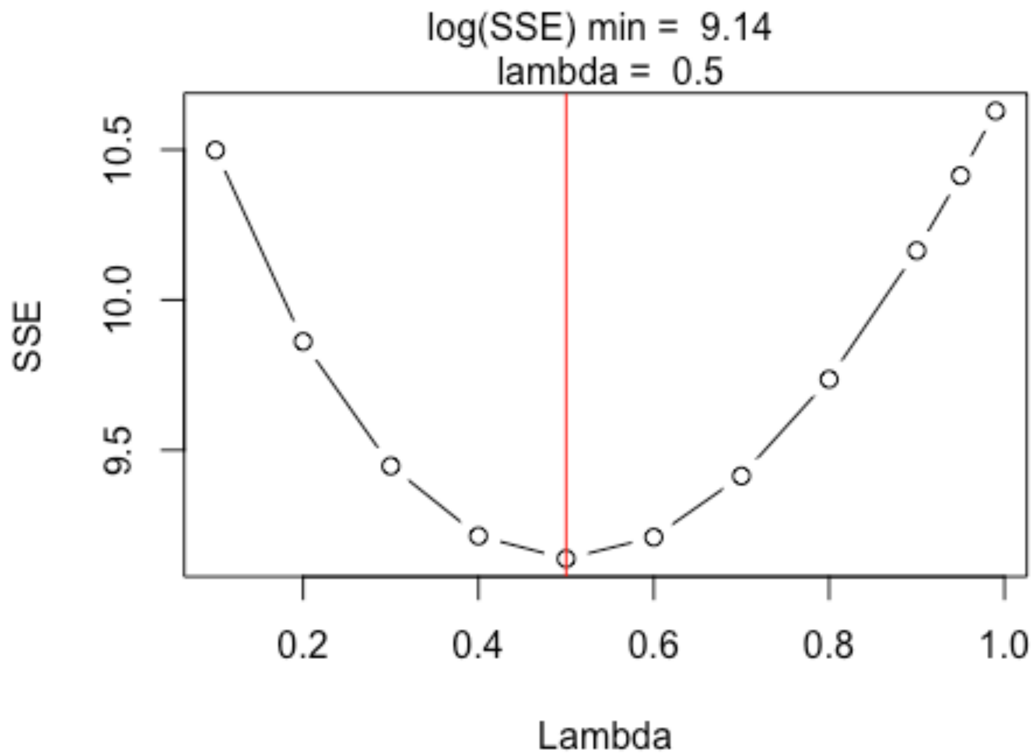



```
#first-order exponential smoothing function
firstsmooth <- function(y, lambda, start = y[1], end = y[12]){
  ytilde <- y
  ytilde[1] <- lambda*y[1] + (1 - lambda)*start
  for (i in 2:length(y)) {
    ytilde[i] <- lambda*y[i] + (1 - lambda)*ytilde[i - 1]
    }
ytilde}


#optimizing lambda
lambda.seq = c(seq(0.1,.9,0.1),.95,0.99)
allSumOFErrors = c()
obsv <- c()
predicted <- c()
difference <- c()

for (i in lambda.seq){
  la <- i
  y.smooth1<-firstsmooth(y=Table.E4.4$y[1:12],lambda=la)
  y.smooth2 <-firstsmooth(y=y.smooth1,lambda=la)
  y.hat <- (2-(la/(1-la)))*y.smooth1-(1-(la/(1-la)))*y.smooth2
  Y.obs <- Table.E4.4$y[1:12]
```

```
  predict.error <- (Y.obs-y.hat)
  sum.sqr.error <- sum((predict.error)^2)
  obsv <- c(obsv,Y.obs)
  predicted <- c(predicted,y.hat)
  difference <- c(difference,predict.error)
  allSumOFErrors <- c(allSumOFErrors, sum.sqr.error)
}

allErrorsDf <- data.frame(
  lambdas = c(lambda.seq),
  sse = c(allSumOFErrors))
allErrorsDf
```

```
##    lambdas       sse
## 1     0.10 36256.635
## 2     0.20 19177.749
## 3     0.30 12669.646
## 4     0.40 10036.118
## 5     0.50  9306.627
## 6     0.60  9996.153
## 7     0.70 12258.954
## 8     0.80 16914.726
## 9     0.90 25947.299
## 10    0.95 33308.934
## 11    0.99 41305.827
```

```
#plotting sse vs lambda
opt.lambda <-allErrorsDf$lambdas[allErrorsDf$sse == min(allErrorsDf$sse)]
plot(log(sse)~lambdas,data = allErrorsDf,type="b",
     ylab='SSE',xlab = 'Lambda')
abline(v = opt.lambda, col = 'red')
mtext(text = paste("log(SSE) min = ", round(min(log(allErrorsDf$sse)),2),
                   "\n lambda = ", opt.lambda))
```



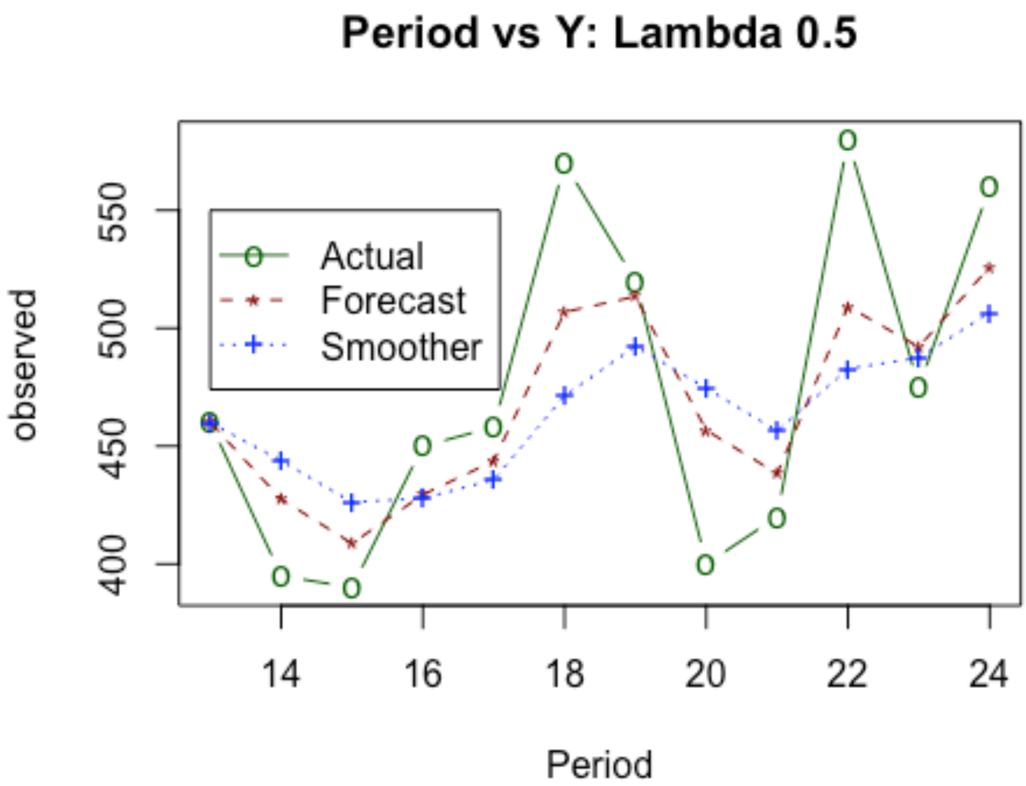

```
#optimum labda is 0.5



#part b

#first-order exponential smoothing function
firstsmooth <- function(y, lambda, start = y[1], end = y[12]){
  ytilde <- y
  ytilde[1] <- lambda*y[1] + (1 - lambda)*start
  for (i in 2:length(y)) {
    ytilde[i] <- lambda*y[i] + (1 - lambda)*ytilde[i - 1]
    }
ytilde}

#Given optimal lambda
y.smooth1<-firstsmooth(y=Table.E4.4$y[13:24],lambda = 0.5)
y.smooth2 <-firstsmooth(y=y.smooth1,lambda = 0.5)
y.hat <- (2-((lambda = 0.5)/(1-(lambda = 0.5))))*y.smooth1-(1-((lambda =
0.5)/(1-(lambda = 0.5))))*y.smooth2

#one-step ahead prediction method
```

```r
observed <- Table.E4.4$y[13:24]
predicted <- y.hat
forecast_error <- (abs((observed-predicted)/observed))*100
partB.table <- data.frame(observed,predicted,forecast_error)
names(partB.table) <- c("Observed","Forecasted","Error(%)")
period <- seq(13,24,1)
```

```r
#forecast table with error column
partB.table
```

```
##    Observed Forecasted  Error(%)
## 1       460   460.0000  0.000000
## 2       395   427.5000  8.227848
## 3       390   408.7500  4.807692
## 4       450   429.3750  4.583333
## 5       458   443.6875  3.125000
## 6       570   506.8438 11.080044
## 7       520   513.4219  1.265024
## 8       400   456.7109 14.177734
## 9       420   438.3555  4.370350
## 10      580   509.1777 12.210735
## 11      475   492.0889  3.597656
## 12      560   526.0444  6.063494
```

```r
#plotting forecast errors
plot(period,observed,main = "Period vs Y: Lambda 0.5",
     xlab = "Period",type = "b", col = "darkgreen",
     pch = "o", lty=1)
points(period, predicted, col="darkred", pch = "*")
lines(period, predicted, col="darkred",lty=2)
points(period, y.smooth2, col="blue", pch="+")
lines(period, y.smooth2, col="blue",lty=3)
legend(13,550,legend=c("Actual","Forecast","Smoother"),
       col=c("darkgreen","darkred","blue"),
       pch=c("o","*","+"),lty=c(1,2,3), ncol=1)
```




## Question 2

TableB.10 contains seven years of monthly data on the number of airline miles flown in the United Kingdom. This is seasonal data. a. Make a time series plot of the data and verify that it is seasonal. b. Use Winters' multiplicative method for the first six years to develop a forecasting method for this data. How well does this smoothing procedure work? c. Make one-step-ahead forecasts of the last 12 months. Determine the forecast errors. How well did your procedure work in forecasting the new data?

Further instructions from D2L:

For exercise 4.27, let all parameters equal to 0.2. For a) It is clearly stated. b) Give the forecast model and draw a plot like Figure 4.31 on page 289. c) List the values of the 12 forecast errors. Then analyze the errors by drawing a plot like Figure 4.32 on page 290.

## Discussion

a) There does seem to be an upward trend and some sort of seasonal cycle.It's worth noting that the seasonal cycles seem to be growing overtime. This method does a good job of capturing the seasonality and variation of the dataset.

b) This smoothing method is easier and less computationally intensive than the tau-step-ahead method. This procedures smooths parameters by minimizing the RMSE while ensuring that the seasonal component of the data is taken into account.

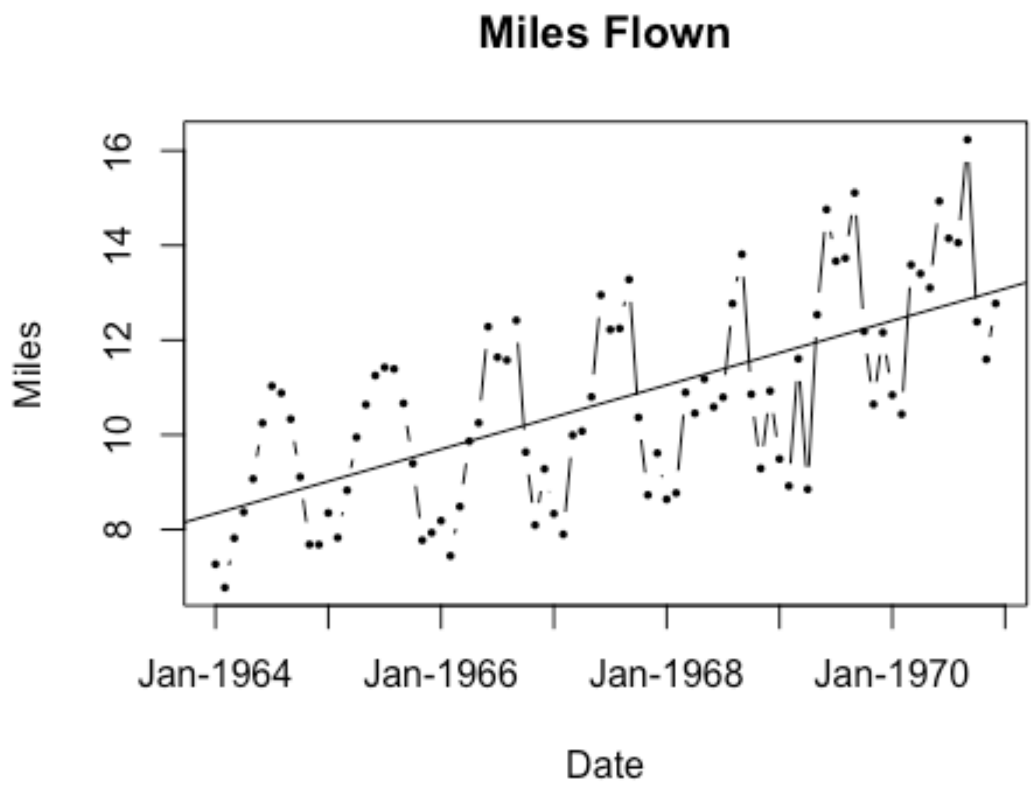
c) Comparing the accuracy measures of both method we get:

One-step-ahead method's error/accuracy:
ME RMSE MAE MPE MAPE 4.413382 4.613956 4.413382 33.22467 33.22467

Holtwinter's method's error/accuracy: ME RMSE MAE MPE MAPE 1.229644 1.72468 1.403504 9.045062 10.32815

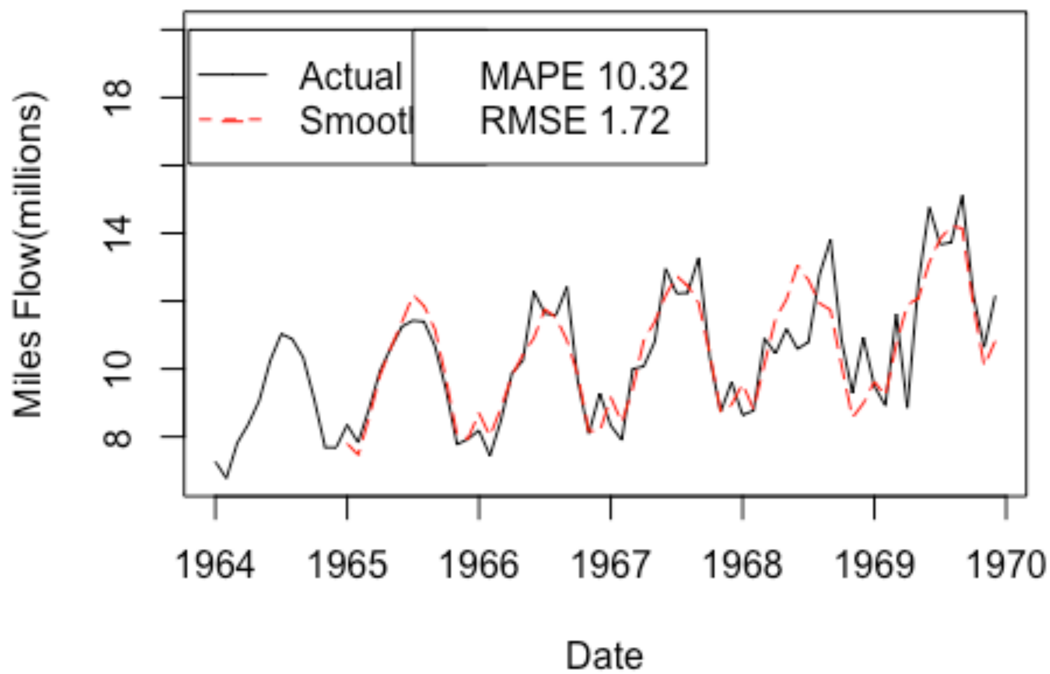
It appears that holtwinter's method is the better and more accurate procedure in forecasting as indicates by the statistics above and the plot below. Having said that, Winter's method is known to overestimate predicted values, so using the damped version of Holtwinter's method might be more appropriate.

```r
library(forecast)

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo

set.seed(1234)
TableB10 = read.csv("~/Desktop/GradSchool/Fall2020/STAT-560/Content/TableB10.csv",
                stringsAsFactors=FALSE)
names(TableB10) <- c("Month", "Miles")

#part a
plot(TableB10[,2],type = "b", pch=16,main = "Miles Flown",
     cex =.5,xlab = 'Date',ylab='Miles',xaxt='n')
axis(1, seq(1,85,12), TableB10[seq(1,85,12),1])
abline(reg = lm(TableB10[,2]~time(TableB10[,2])))
```



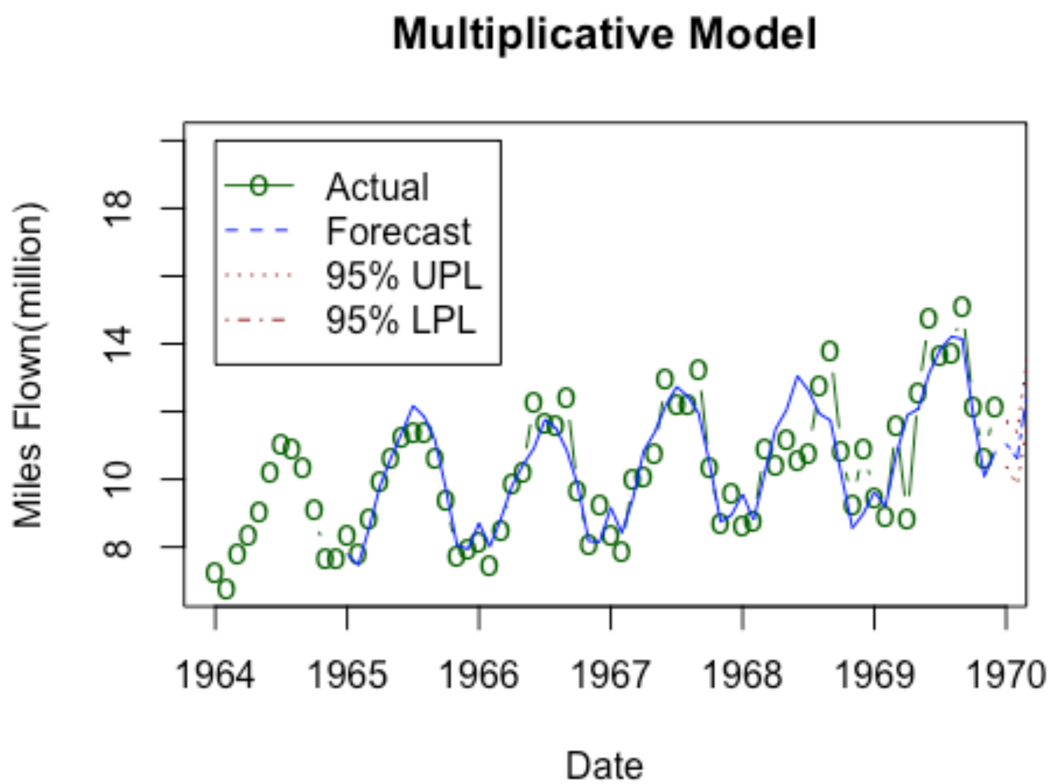

```
#constructing a model using holtwinter's method
#part b
data.timeseries <- ts(TableB10$Miles, start = c(1964,1),frequency = 12)
y <- TableB10$Miles[1:72]

# convert data to ts object
y.ts <- ts(y, start = c(1964,1), frequency = 12)

#multiplicative
miles.hw.mult <- HoltWinters(y.ts,alpha=0.2,beta=0.2,gamma=0.2,
                             seasonal = "multiplicative")

y2 <- TableB10$Miles[73:84]
y2.ts <- ts(y2, start = c(2004,1),frequency = 12)
y2.forecast <- predict(miles.hw.mult, n.ahead=12, prediction.interval = TRUE)


plot(y.ts,type = "l", pch = 19,cex=.5,ylim = range(min(y.ts),20),
     xlab='Date',ylab='Miles Flow(millions)',
     col = "black", lty=1)
lines(miles.hw.mult$fitted[,1],type="l",
      col = "red",pch=19, lty=5)
legend(1963.8,20,legend=c("Actual","Smoothed"),
```

```
       col=c("black","red"),
       pch=c(".","_"),lty=c(1,2), ncol=1)
legend(1965.5,20,c("MAPE 10.32","RMSE 1.72"))
```




```
#part c
#forecasting the last 12 observations using holtwinter's method
y2.forecast

##                fit      upr       lwr
## Jan 1970 11.05085 11.71432 10.387389
## Feb 1970 10.61594 11.40019  9.831684
## Mar 1970 12.76010 13.78349 11.736718
## Apr 1970 13.18773 14.40413 11.971339
## May 1970 15.02865 16.57392 13.483374
## Jun 1970 16.62897 18.53148 14.726465
## Jul 1970 16.83737 18.97988 14.694864
## Aug 1970 17.39488 19.83038 14.959372
## Sep 1970 17.85084 20.58055 15.121126
## Oct 1970 14.78945 17.27369 12.305203
## Nov 1970 12.58429 14.89881 10.269760
## Dec 1970 13.52265 17.29842  9.746887

#plotting forecast errors
plot(y.ts,main = "Multiplicative Model",
```

```
     xlab = "Date",ylab = "Miles Flown(million)",
     type = "b", col = "darkgreen",ylim = range(min(y.ts),20),
     pch = "o", lty=1)
lines(miles.hw.mult$fitted[,1],type = "l", col = "blue")
lines(y2.forecast[,1], col="blue",type="l",lty=6)
lines(y2.forecast[,2], col="darkred",type="l",lty=3)
lines(y2.forecast[,3], col="darkred",type="l",lty=3)
legend(1964,20,legend=c("Actual","Forecast","95% UPL","95% LPL"),
       col=c("darkgreen","blue", "darkred", "darkred"),
       pch=c("o",NA,NA,NA),lty=c(1,2,3,4), ncol=1)
```




```
#calculating error
forecast.error <- data.frame(Actual= y2, Forecasted = y2.forecast,
                             Error = abs((y2-y2.forecast)/y2))
names(forecast.error)

## [1] "Actual"         "Forecasted.fit" "Forecasted.upr" "Forecasted.lwr"
## [5] "Error.fit"      "Error.upr"      "Error.lwr"

#comparing our calculated error to the accuracy function
accuracy1 <- abs(accuracy(y2.forecast, y2))

calcualed.MAPE <- (mean(forecast.error$Error.fit))*100
calcualed.MAPE
```

```
## [1] 10.32815

#Forecast error and accuracy
forecast.error

##    Actual Forecasted.fit Forecasted.upr Forecasted.lwr  Error.fit
Error.upr
## 1  10.840       11.05085       11.71432      10.387389 0.01945137
0.08065652
## 2  10.436       10.61594       11.40019       9.831684 0.01724192
0.09239073
## 3  13.589       12.76010       13.78349      11.736718 0.06099751
0.01431245
## 4  13.402       13.18773       14.40413      11.971339 0.01598757
0.07477466
## 5  13.103       15.02865       16.57392      13.483374 0.14696243
0.26489533
## 6  14.933       16.62897       18.53148      14.726465 0.11357208
0.24097494
## 7  14.147       16.83737       18.97988      14.694864 0.19017273
0.34161892
## 8  14.057       17.39488       19.83038      14.959372 0.23745295
0.41071213
## 9  16.234       17.85084       20.58055      15.121126 0.09959578
0.26774363
## 10 12.389       14.78945       17.27369      12.305203 0.19375622
0.39427625
## 11 11.594       12.58429       14.89881      10.269760 0.08541377
0.28504524
## 12 12.772       13.52265       17.29842       9.746887 0.05877329
0.35440169
##      Error.lwr
## 1  0.041753779
## 2  0.057906884
## 3  0.136307465
## 4  0.106749799
## 5  0.029029519
## 6  0.013830776
## 7  0.038726534
## 8  0.064193761
## 9  0.068552063
## 10 0.006763808
## 11 0.114217696
## 12 0.236855112

accuracy1

##                 ME    RMSE      MAE      MPE     MAPE
## Test set 1.229644 1.72468 1.403504 9.045062 10.32815

calcualed.MAPE
```

```
## [1] 10.32815

#part c using the one-step method with lambda being 0.2 and tau  being 1
y1.smooth1<-firstsmooth(y=TableB10$Miles[1:72],lambda = 0.2)
y2.smooth2 <-firstsmooth(y=y1.smooth1,lambda = 0.2)
y2.hat <- (2-((lambda = 0.2)/(1-(lambda = 0.2))))*y1.smooth1-(1-((lambda =
0.2)/(1-(lambda = 0.2))))*y2.smooth2

#one-step ahead prediction method
observed2 <- TableB10$Miles[73:84]
predicted2 <- y2.hat
forecast2_error <- (abs((observed2-predicted2)/observed2))*100
partc.table <- data.frame(observed2,predicted2,forecast2_error)
names(partc.table) <- c("Observed","Forecasted","Error(%)")

head(partc.table,12)

##    Observed Forecasted Error(%)
## 1    10.840   7.269000 32.94280
## 2    10.436   7.110920 31.86163
## 3    13.589   7.330392 46.05643
## 4    13.402   7.667038 42.79183
## 5    13.103   8.134011 37.92253
## 6    14.933   8.847552 40.75168
## 7    14.147   9.610477 32.06703
## 8    14.057  10.108170 28.09156
## 9    16.234  10.281966 36.66400
## 10   12.389  10.003557 19.25452
## 11   11.594   9.333913 19.49359
## 12   12.772   8.838425 30.79843

accuracy2 <- abs(accuracy(predicted2, observed2))
accuracy2

##                ME     RMSE      MAE      MPE     MAPE
## Test set 4.413382 4.613956 4.413382 33.22467 33.22467
```

## Question 3

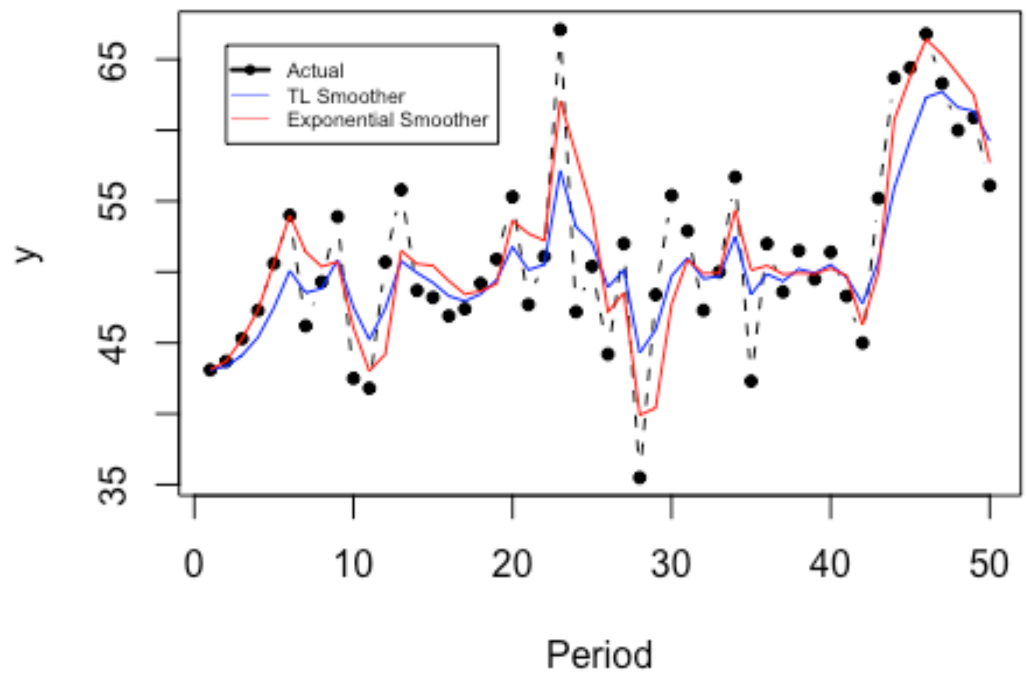
Instructions from D2L:

For example 4.6, show the plot similar to Figure 4.25 by letting the simple exponential smoother with = 0.4 and TL smoother with = 0.4. Print the first 10 calculations in a table that is similar to Table 4.9 on page 276.

## Discussion

```
library("readxl")
Table.E4.2 <- read_excel("~/Desktop/GradSchool/Fall2020/STAT-
560/Content/TableE4_2.xlsx")
```

```r
trigg.leach.smooth <- function(y,delta,y.tilde.start=y[1],lambda.start = 1){
  T <-length(y)
#Initializing the vectors
  Qt <- vector()
  Dt <- vector()
  y.tilde <- vector()
  lambda <- vector()
  err <- vector()
#Setting the starting values for the vectors
  lambda[1] = lambda.start
  y.tilde[1] = y.tilde.start
  Qt[1] <- 0
  Dt[1] <- 0
  err[1] <- 0
  for (i in 2:T){
    err[i] <- y[i]-y.tilde[i-1]
    Qt[i] <- delta*err[i]+(1-delta)*Qt[i-1]
    Dt[i] <- delta*abs(err[i])+(1-delta)*Dt[i-1]
    lambda[i] <- abs(Qt[i]/Dt[i])
    y.tilde[i] = lambda[i]*y[i] + (1-lambda[i])*y.tilde[i-1]
  }
return(cbind(y.tilde,lambda,err,Qt,Dt))
}
#Trigg-Leachh smoother for the data
Trigg.Leach.smooth <- trigg.leach.smooth(Table.E4.2$y,delta = 0.4)

#Simple exponential smoother for the dataset
simple.expo.smooth <- firstsmooth(y=Table.E4.2$y,lambda = 0.4)

simple.expo.smooth <- data.frame(simple.expo.smooth)
Trigg.Leach.smooth <- data.frame(Trigg.Leach.smooth)
colnames(Trigg.Leach.smooth) <- c("y", "Lambda", "Error","Qt", "Dt")
colnames(simple.expo.smooth) <- c("y")


#Plot the data together with TL and exponential smoother like figure 4.25
plot(Table.E4.2, ylab="y",
     xlab="Period", main="",pch=20,
     col="black",type="b", lty =2)
lines(simple.expo.smooth, col="blue", type="l")
lines(Trigg.Leach.smooth$y, col="red", type="l")
legend(2,66,c("Actual","TL Smoother","Exponential Smoother"),
       pch=c(20, NA, NA),lwd=c(2,.5,.5),
       cex=.55,col=c("black","blue","red"))
```




```
head(Trigg.Leach.smooth,10)

##           y     Lambda     Error         Qt       Dt
## 1  43.10000 1.00000000  0.000000  0.0000000 0.000000
## 2  43.70000 1.00000000  0.600000  0.2400000 0.240000
## 3  45.30000 1.00000000  1.600000  0.7840000 0.784000
## 4  47.30000 1.00000000  2.000000  1.2704000 1.270400
## 5  50.60000 1.00000000  3.300000  2.0822400 2.082240
## 6  54.00000 1.00000000  3.400000  2.6093440 2.609344
## 7  51.41244 0.33173798 -7.800000 -1.5543936 4.685606
## 8  50.38543 0.48617279 -2.112444 -1.7776137 3.656341
## 9  50.71667 0.09424834  3.514569  0.3392594 3.599632
## 10 46.06540 0.56607780 -8.216673 -3.0831137 5.446449

#comparing Trigg-Leach and simpler exponential outputs
data.frame(head(Trigg.Leach.smooth$y,10))

##    head.Trigg.Leach.smooth.y..10.
## 1                        43.10000
## 2                        43.70000
## 3                        45.30000
## 4                        47.30000
## 5                        50.60000
## 6                        54.00000
```

```
## 7                            51.41244
## 8                            50.38543
## 9                            50.71667
## 10                           46.06540
```

```r
head(simple.expo.smooth,10)
```

```
##           y
## 1  43.10000
## 2  43.34000
## 3  44.12400
## 4  45.39440
## 5  47.47664
## 6  50.08598
## 7  48.53159
## 8  48.83895
## 9  50.86337
## 10 47.51802
```