Data 624: Week 4 Homework

Angrand, Burke, Deboch, Groysman, Karr

October 12, 2019

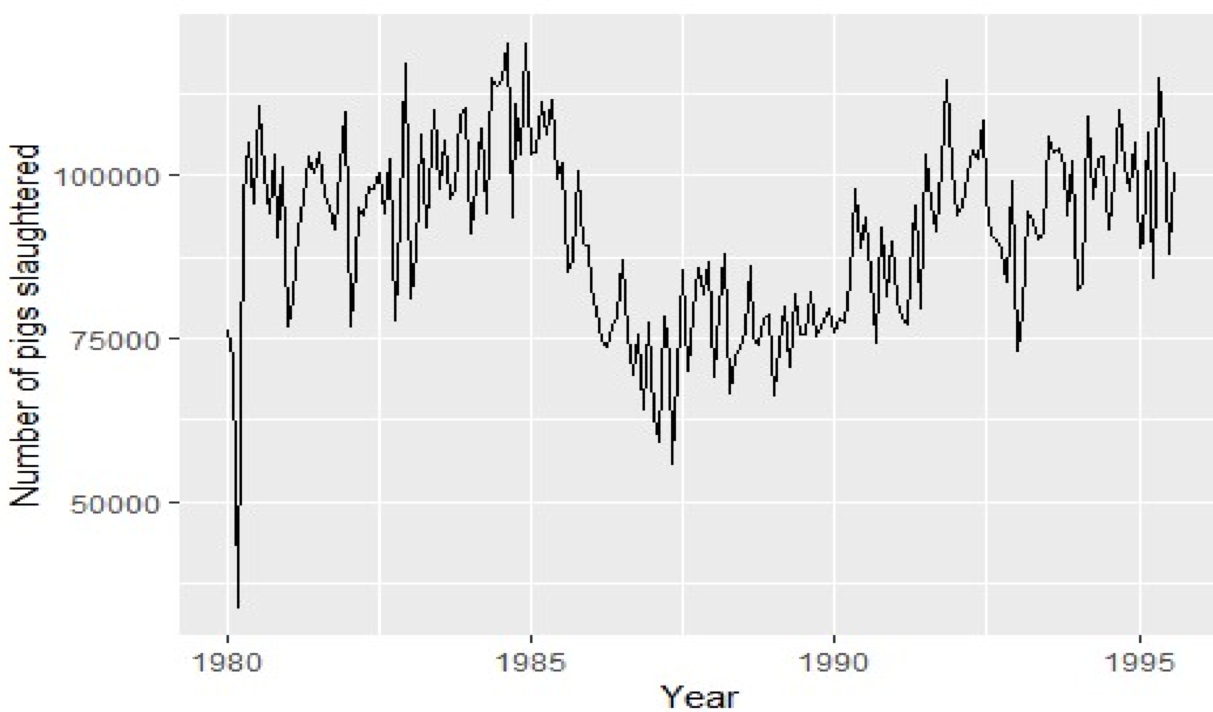
Week 4 Assignment

Chapter 7 HA 7.1, 7.3

*7.1 Consider the **pigs** series - the number of pigs slaughtered in Victoria each month.*

The pigs ts contains monthly total number of pigs slaughtered in Victoria, Australia (Jan 1980 to Aug 1995).

```
#sampling and shape of dataset, preliminary EDA
head(pigs)

##         Jan    Feb    Mar    Apr    May    Jun
## 1980  76378  71947  33873  96428 105084  95741

#autoplot the series
pigsdata<-window(pigs)
autoplot(pigsdata) +
  ylab("Number of pigs slaughtered") +xlab("Year")
```

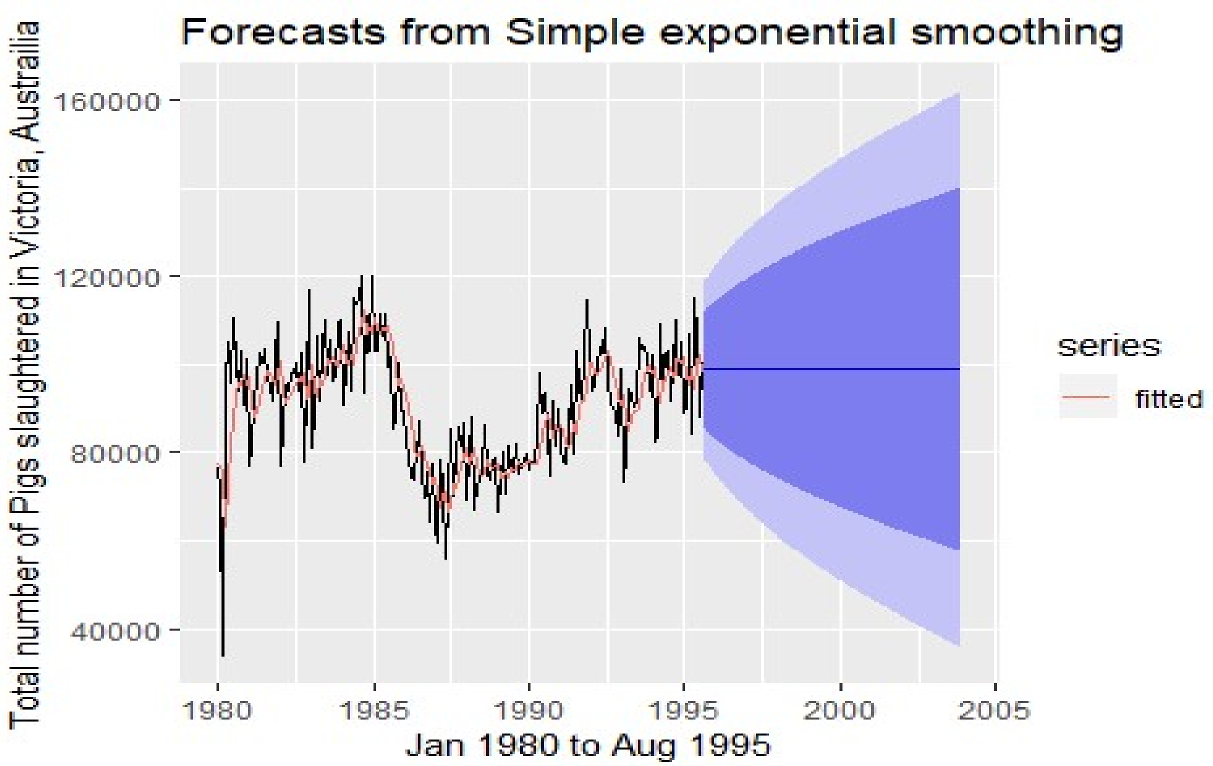


a. Use the **ses()** function in R to find the optimal values of α and ℓ_0, and generate forecasts for the next four months.

```
#Estimate parameters
fc_pigs_ses<-ses(pigsdata, h=4)




#Get forecasted estimate parameters from model -(alpha and l)
round(fc_pigs_ses$model$par[1:2],4)

##      alpha           l
##     0.2971 77260.0561

#generate 4 months of forecasts
data.frame(fc_pigs_ses)

##          Point.Forecast    Lo.80    Hi.80    Lo.95    Hi.95
## Sep 1995       98816.41 85605.43 112027.4 78611.97 119020.8
## Oct 1995       98816.41 85034.52 112598.3 77738.83 119894.0
## Nov 1995       98816.41 84486.34 113146.5 76900.46 120732.4
## Dec 1995       98816.41 83958.37 113674.4 76092.99 121539.8

#Accuracy of one-step-ahead training errors
round(accuracy(fc_pigs_ses),2)

##                  ME    RMSE     MAE   MPE MAPE MASE ACF1
## Training set 385.87 10253.6 7961.38 -0.92 9.27  0.8 0.01

# see how SES model was fitted
fc_pigs_ses$model

## Simple exponential smoothing
##
## Call:
##  ses(y = pigsdata, h = 4)
##
##   Smoothing parameters:
##     alpha = 0.2971
##
##   Initial states:
##     l = 77260.0561
##
##   sigma:  10308.58
##
##      AIC     AICc      BIC
## 4462.955 4463.086 4472.665

# get 1st 4 months of forecasts
tsCV(pigs,ses,h=4)[1:4,]
```

```
##              h=1        h=2       h=3       h=4
## [1,]  -4431.00 -42505.00 20050.00 28706.00
## [2,] -42431.44  20123.56 28779.56 19436.56
## [3,]  62555.00  71211.00 61868.00 76774.00
## [4,]  28706.00  19363.00 34269.00 23953.00
```

```
#Plot (Note that forecast using ses doesn't have a trend component.)
fc_pigs_ses<-ses(pigs, h=100)
autoplot(fc_pigs_ses) +
    autolayer(fitted(fc_pigs_ses), series="fitted") +
    ylab("Total number of Pigs slaughtered in Victoria, Austrailia") + xlab("
Jan 1980 to Aug 1995")
```



b. Compute a 95% prediction interval for the first forecast using $\hat{y} \pm 1.96s$ is the standard deviation of the residuals. Compare your interval with the interval produced by R.

```
# 95% prediction interval for the first forecast

lower.upper <- data.frame( fc_pigs_ses$lower[1, "95%"], fc_pigs_ses$upper[1,
"95%"])
names(lower.upper)<- c("lower.limit", "upper.limit")
lower.upper

##     lower.limit upper.limit
## 95%    78611.97    119020.8
```

```r
# calculate standard deviation of residuals with and without model, s = 10273
.69 vs s (estimated) 10308.58
s <- sd(fc_pigs_ses$residuals)
print(paste("Standard Deviation: ",round(s,2)))

## [1] "Standard Deviation:  10273.69"

# calculate 95% prediction interval with model
pred.interval.model<- data.frame(fc_pigs_ses$mean[1] - 1.96*s,fc_pigs_ses$mea
n[1] + 1.96*s)
names(pred.interval.model)<- c("Lower.Model", "Upper.Model")
pred.interval.model

##   Lower.Model Upper.Model
## 1    78679.97    118952.8

# calculate 95% prediction interval without model
pred.interval.womodel<- data.frame(mean(pigs)  - 1.96*s, mean(pigs) + 1.96*s)
names(pred.interval.womodel)<- c("Lower.NOModel", "Upper.NOModel")
pred.interval.womodel

##   Lower.NOModel Upper.NOModel
## 1         70504      110776.9
```

- R gives an interval of [78611.97, 119020.8] and by computing the standard deviation of the residuals we got [78679.97, 118952.8]. Both are really close.

*7.3 Modify your function from the previous exercise to return the sum of squared errors rather than the forecast of the next observation. Then use the **optim()** function to find the optimal values of α and ℓ_0. Do you get the same values as the **ses()** function?*

- my_ses_func returns an l value that is ~.01% different then the ses function.

```
##      alpha          l
##     0.2971 77269.3253

##      alpha          l
##     0.2971 77260.0561
```