Chapter 2 Solutions

Notice to readers. Unless the R program is written within these solutions, the reader will find all R programs cited in the book and all time series data cited in the book in the code tab at `https://timeseriestime.org`

2.3.2 Exercises

Exercise 2.1

The 3-point moving average smoother for time series y_t would have first term $\frac{1}{3}(y_t + y_{t+1} + y_{t+2})$. The second terms would be $\frac{1}{3}(y_{t+1} + y_{t+2} + y_{t+3})$.

If we use a 3-point moving average, because 3 is an odd number we would not need to center the moving average. As indicated on Page 68, if we average an even number of data points we need to center. But if the number of average terms is odd, we do not center. Thus, the operation that we do next would not be usually done in practice. For the sake of this mathematical exercise, let's assume we do it just to see the mathematical solution.

Upon taking the 2-point moving average of those two terms we obtain:

$\frac{1}{2}\left[\frac{1}{3}(y_t + y_{t+1} + y_{t+2}) + \frac{1}{3}(y_{t+1} + y_{t+2} + y_{t+3})\right] = \frac{1}{6}(y_t + 2y_{t+1} + 2y_{t+2} + y_{t+3})$.

This is equivalent to the smoothing operation $(1/6)[1, 2, 2, 1]$.

Two practical reasons why we would smooth a seasonal time series with a moving average smoother are: (i) to visualize the trend component without the clutter of the seasonal and the random term. (ii) To be able to detrend the time series, i.e., remove the trend for further analysis.

□

Exercise 2.2

The solution to this problem requires program *ch2janacek-smoothers.R* but there are three steps before you apply it:

(i) Obtain data *ch1passengers.csv* from Chapter 1's program and data files. Replace the data statement in *ch2janacek-smoothers.R* with

```
data=read.csv("ch1passengers.csv", header=T)
head(data)
##
dim(data)
```

(ii) Apply the program used in Chapter 1 to remove the commas from the data. Use the relevant statements in *ch1passengersplot.R*

(iii) Extract from the data frame the variable of interest, *domestic* for January 2012 to 2017 using the statements in *ch1passengersplots.R*

Once you do all that, the data is ready to be analyzed with *ch2janacek-smoothers.R* and the Base R function decompose(). Notice that there will have to be some changes in the last program, namely the k=12 for the passengers data, and the plot of the data needs to be told that we lose 6 points of data at the beginning and 6 data points at the end when we do the the centered twelve-point moving average.

Since 12 is an even number, the moving average obtained is centered.

Program *Exercise 2-2.R* illustrates how we put together that code.

By running the program, you will find that decompose() and Janacek's function give exactly the same moving average trend estimate.

□

Exercise 2.3

The 12-point MA values that start between June and July of 2005 are: 17.5175,19.05917, 20.6175, 22.02083, 23.48083, 24.98417, 26.48333,27.88083,29.4125 The 2-point MA of the 12-point MA, starting in July 2005 is: 18.28834, 19.83833, 21.31917, 22.75083, 24.2325, 25.73375, 27.18208, 28.64667

The moving average for the first filter is $(1/12)(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$ and the second filter is $(1/2)(1, 1)$. The filter for the two operations combined is $(1/24)(1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1)$.

□

Exercise 2.4

(a)
```
jj=JohnsonJohnson
jj
class(jj)
jj=ts(jj,start=1960, frequency=4)
plot(jj, ylab="Earnings per Share", main="J & J")
start(jj); end(jj);frequency(jj)
summary(jj)
length(jj)
```

(b) Add the following line to the code to obtain the plot.

```
plot.ts(jj, ylab="Earnings (dollars)",
main="Quarterly earnings (dollars) \n per Johnson & Johnson share 1960{80")
```

(c) You will see the first observations of the time series by typing in R:

```
jj=JohnsonJohnson
jj
```

We could do the calculations requested by hand (using calculator or R) or we could use a program. First, by hand, we could take first a 4-point moving average.

(0.71+0.63+0.85+0.44)/4 = 0.6575

(0.63+0.85+0.44+0.61)/4 = 0.6325

(0.85+0.44+0.61+0.69)/4 = 0.6475

(0.44+0.61+0.69+0.92)/4 = 0.665

(0.61+0.69+0.92+0.55)/4 = 0.6925

(0.69+0.92+0.55+0.72)/4 = 0.72

(0.92+0.55+0.72+0.77)/4= 0.74

(0.55+0.72+0.77+0.92)/4 = 0.74

(0.72+0.77+ 0.92+0.6)/4 = 0.7525

Then, we can take a 2 point moving average of the 4 point moving average

(0.6575+0.6325)/2= 0.645

(0.6325+0.6475)/2 = 0.64

(0.6475+0.665)/2 = 0.65625

(0.665+0.6925)/2 = 0.67875

(0.6925+0.72)/2 = 0.70625

(0.72+0.74)/2 = 0.73

(0.74+0.74)/2 = 0.74

(0.74+0.7525)/2 = 0.74625

Alternatively, you can compile the following functions[85], which shows the programming that goes behind smoothing a time series with a moving average. If you like programming, this will be a fun program to run.

```
odd<-function(k)
# utility function
# if k is odd it returns True
# if k is even it returns False
# if k is zero it returns False
{if ((-1)^k<0) (m<-T) else (m<-F)
m}



swindow<-
function (x, k,pt=T)
{
#moving average windowed smoother
#give it k and it will smooth with equally weighted window length k
# scaled to sum to 1
#plots if pt=T
```

```
# output is smoothed seies

if (odd(k)) {
w <- seq(k) * 0 + 1
}
else {
k <- k + 1
w <- seq(k) * 0 + 2
w[1] <- 1
w[k] <- 1
}
w <- w/sum(w)
n <- length(x)
kf <- floor(k/2)
m <- (n - k + 1)
smooth <- seq(1, m)
for (i in 1:m) {
smooth[i] <- w %*% x[i:(k + i - 1)]
}

sm<- c(x[1:kf], smooth, x[(n - kf + 1):n])
if(pt)
{time<-seq(n)
plot(time,x,type="b")
points(time,sm,pch="+")}
sm}
swindow2<-
function (x, k,pt=T)
{
if (odd(k)) {
#moving average windowed smoother
#give it w and it will smooth x with this window
# i like w to be scaled to sum to 1
# two plots one of original one of smoothed non overlapping
w <- seq(k) * 0 + 1
}
else {
k <- k + 1
w <- seq(k) * 0 + 2
w[1] <- 1
w[k] <- 1
}
w <- w/sum(w)
n <- length(x)
kf <- floor(k/2)
m <- (n - k + 1)
smooth <- seq(1, m)
for (i in 1:m) {
smooth[i] <- w %*% x[i:(k + i - 1)]
}

sm<- c(x[1:kf], smooth, x[(n - kf + 1):n])
if(pt)
{time<-seq(n)
par(mfrow=c(2,1))
plot(time,x,type="b")
plot(time,sm,type="b")}
par(mfrow=c(1,1))
sm}



and then run



swindow(jj,4,pt=T)

or  run



smoothed=swindow(jj,4,pt=T)
head(smoothed, 15)
```

Notice that the *swindow* function returns the first two values of the original series and then the third value is the already smoothed one. From then on, they are all smoothed until the last two before the last, which again are taken from the original series.

So the requested numbers nicely formatted and summarized can be seen in the following table.

If you add to the R code for this problem the following command,

```
decompose(jj)$trend
```

you will notice that the first numbers in the resulting trend component are identical to those in the last column of Table **??**, obtained with must less programming but be aware that behind that simple line of code there is the larger program above it.

□

2.3.8 Exercises

Exercise 2.5

Notice that `nottem` is average monthly temperatures at Nottingham, 1920-1939, a total of 20 years of monthly observations. You can find what the time series is by typing in R

Table for Exercise 2.4(c). Centered 4-point moving average of the first 12 observations of JohnsonJohnson in Exercise 2.4

t	y_t	4-point MA	2-point of 4-point MA
1960:1	0.71		
1960:2	0.63		
		0.6575	
1960:3	0.85		0.645
		0.6325	
1960:4	0.44		0.64
		0.6475	
1961:1	0.61		0.65625
		0.665	
1961:2	0.69		0.67875
		0.6925	
1961:3	0.92		0.70625
		0.72	
1961:4	0.55		0.73
		0.74	
1962:1	0.72		0.74
		0.74	
1962:2	0.77		0.74625
		0.7525	
1962:3	0.92		
1962:4	0.6		

```
?nottem
```

Thus it is a monthly time series. We could aggregate the time series to make it quarterly and apply the method of Section 2.3 (a four-term centered moving average), since the number of quarters to average is even. Section 2.2 has the R code to do the decomposition. You may use the following code to do the decomposition after aggregating. The aggregation will result in a ts object with 80 quarters (hence the need to do centered moving average).

```
nottem.quarterly= aggregate(nottem,nfrequency=4,FUN=mean)
length(nottem.quarterly)
class(nottem.quarterly)

#The seasonal effect of all quarters 4 is the same, so we could just look at
decompose(nottem.quarterly)$seasonal[4]  #gives quarter 4's seasonal effect.
# or we could look at
decompose(nottem.quarterly)$seasonal[40]  #gives quarter 4 of 1929

## To see if the assumption of additive decomposition is appropriate do

plot(decompose(nottem.quarterly))
```

The seasonal effect for quarter 4 of 1929 and all quarters 4 is equal to -5.105263.

By looking at the plot of the decomposition, we can see that the random term fluctuates randomly around a constant mean of 0 with no discernible pattern. Hence it looks like the additive decomposition assumption is a good one.

Extra

It is a good exercise to apply a 12-month moving average even though it is not asked and we did not do it in the Chapter's Sections. The generalization of the method of Section 2.3 to monthly data using `decompose` is straightforward.

Keeping it monthly, the seasonal effect is for each month, not quarter. Let's assume we want month 4, April's seasonal effect. Since the seasonal effect of April is the same for all April months, we just look at the first April in 1920.

For the first part of the question, since `nottem` is of class ts(), and R therefore already knows that it is a monthly time series, we simply type in R,

```
decompose(nottem)$seasonal[4]  #gives, April's seasonal effect estimate
```

We will find out that this seasonal effect estimate is $\hat{s}_{April} = -2.7573465$

Notice that the seasonal swings are not constant on April throughout the years but the seasonal effect is constant in every April because it is the average of all April seasonal swings, an estimate.

The plot of the decomposition,

```
plot(decompose(nottem)) #without other arguments in the function,
# this does additive decomp
```

shows that the random term is randomly fluctuating around a constant value, and has no obvious changing variance. Thus it appears that additive decomposition is good enough for the time series?nottem

Exercise 2.6

The following code will show that indeed the sum of the trend, seasonal and random component at time $t = 5$ and time $t = 9$obtained with the `decompose` function equals the value of the time series at time $t = 5$ and $t = 9$, respectively.

```
data=c( 3.3602, -3.1769, 0.3484, 7.469, 4.4963, -0.4621,
0.7218, 6.9484, 5.2374, 2.9242, 4.7006,
11.2793, 5.1637, 1.5441,  12.121,  9.6588, 8.0922,
3.9653, 11.4177, 13.2088 )

y=ts(data, start=c(1960,2), end=c(1965,1),frequency=4)

## Notice that R's decompose() uses by default additive decomposition.
## Thus if you do not specify type, it will do additive.

trend=decompose(y)$trend    #extract trend
seasonal=decompose(y)$seasonal  #extract seasonal
random=decompose(y)$random   #extract random
```

```
trend[5]+seasonal[5]+random[5] # one instance of addition of components, 1961:2
data[5]  #compare with value of the data

trend[9]+seasonal[9]+random[9] # another instance of addition of components 1962:2
data[9] #compare with value of the data at 1962:2
```

□

Exercise 2.7

(a) (b) Run program *ch2mediansmootherexercise.R* to see the plot and to obtain the values needed for part (c). The calculated 4-point medians obtained from the data using the program are:

3.3602, 0.3484, 0.7218, 4.4963, 4.4963, 2.9242, 4.7006, 5.2374,

5.1637, 4.7006, 5.1637, 9.6588, 8.0922, 8.0922, 9.6588, 9.6588.

(c)

The data values are: 3.3602, -3.1769, 0.3484, 7.469, 4.4963, -0.4621, 0.7218, 6.9484, 5.2374, 2.9242, 4.7006, 11.2793, 5.1637, 1.5441, 12.121, 9.6588, 8.0922, 3.9653, 11.4177, 13.2088

The first running median is calculated by the program as follows, using the first five observations.

(i) Sorting $3.3602, -3.1769, 0.3484, 7.469, 4.4963$ from lowest to highest, we obtain the sorted first five numbers as:

$-3.1769, 0.3484, 3.3602, 4.4963, 7.4690$

(ii) Median($-3.1769, 0.3484, 3.3602, 4.4963, 7.4690$)= 3.3602

To obtain the second running median, the program does the following:

(i) Sort $-3.1769, 0.3484, 7.469, 4.4963, -0.4621$ from lowest to highest to obtain $-3.1769 -0.46210.34844.49637.4690$

(ii) Median($-3.1769 - 0.46210.34844.49637.4690$)=0.3484

And so on and so forth.

(d) The median smoother is not as smooth as the moving average smoother. Compare with Figure 2.4 in the chapter.

□

Exercise 2.8

The reader should notice that the third column should say $\widehat{T}$ instead of *ma4*. That column is the result of executing the 4-term centered moving average, as described in Section 2.3 and 2.3.3.

To obtain the seasonally adjusted time series, deduct the $\hat{S}_t$ column from the x_t columns. The first results are

1985 Q4: 472 – 25.5185 = 446.4815

1986 Q1: 821 – 512.58125 = 308.4185

Complete the results up to 1990 Q2.

As an extra exercise, the reader could double check that the $\widehat{S}_t$ was calculated as indicated in Section 2.3.3.

□

Exercise 2.9

(a) Program `chapter2ma4trend.R` is a simple way of doing the job requested in this question. The program can be used to alert students that the random term obtained with the function `decompose()` is the detendred and seasonally adjusted data. Although the plot of the decompose() output already indicates the difference between the raw original time series and the random term by showing them in two different plots, it helps to see both plotted on the same plot. Students can then be made aware that a special kind of plot, a plot with two vertical axes, is needed to be able to see two time series that have very different scale in the same image. Sometimes, students plot time series of very different scales on the same image and then complain that they can not see one of the images. This two-axes plot helps remind them not to do that.

This question, because of the short time series involved, can also be used to show the programming involved in decomposing and obtaining the components. Chapter 2, Section 2.3, explains that in detail, mathematically, but it is interesting for students that are particularly interested in programming, to see the programming involved. To do this, the basic programs of Janacek [85] are very didactic. The programs are not included in program `chapter2ma4trend.R` but they can be copy-pasted from this manual. They can also be accessed from timeseriestime.org, history page.

Actually, using Janacek's program helps assess whether students have understood the additive decomposition of Section 2.3. First, it helps remind them that there are missing values in the MA sequence, since we lose data when smoothing. Second, when the program gives the estimated seasonal effect, which are slightly different from those in Table 2.3 due to rounding, the way Janacek's program produces them as

```
seasonaleffect
1        2        3        4
-26.790 -464.265  -27.515  510.585
```

Students can be prompted to indicate which quarter each of those seasonal effects is for. If they look at Table 2.3, they will see that Janacek's program is producing, labeled as "1" the seasonal effect for quarter 2; labeled as "2" the seasonal effect for quarter 3; labeled as "3" the seasonal effect for quarter 4; labeled as "4" the seasonal effect for quarter 1.

The following is the program that uses Janacek's functions plus the code to produce the plots.

```
# Highlight and compile (run) the functions from Janacek's
## Janacek https://archive.uea.ac.uk/~gj/book/smooth.code

tsplot<-function(x){       # Janacek's. Highlight all and run to compile
# time series plotter - plots time against x
# this is the basic time domain plotter
time<-seq(1,length(x))
plot(time,x,type="b")
}

#Table 2.3, x
data=c(322, 144, 472, 821, 408, 247, 626, 925, 434, 259, 681, 1277,
829, 435, 940, 1639, 1222, 592, 1055, 2000, 1278, 768, 1415, 2417)
```

```
tsplot(data)   # apply Janacek’s function



odd<-function(k)      # Janacek’. Highlight all and run to compile
# utility function
# if k is odd it returns True
# if k is even it returns False
# if k is zero it returns False
{if ((-1)^k<0) (m<-T) else (m<-F)
m}



# highlight the whole function and compile this ma(k) smoother
swindow3 <-
function (x, k, pt = T)
{

#keep seasonal
ks <- k
#moving average windowed smoother
#give it k and it will smooth with equally weighted window length k
# scaled to sum to 1
#plots if pt=T
# output is smoothed series but also gives seasonal estimates
if (odd(k)) {
w <- seq(k) * 0 + 1
}
else {
k <- k + 1
w <- seq(k) * 0 + 2
w[1] <- 1
w[k] <- 1
}
w <- w/sum(w)
n <- length(x)
kf <- floor(k/2)
m <- (n - k+1)
smooth <- seq(1, m)
time <- seq((kf + 1), (n - kf))
for (i in 1:m) {
smooth[i] <- w %*% x[i:(k + i - 1)]
}
sm <- c(x[1:kf], smooth, x[(n - kf + 1):n])
time <- seq(n)
if (pt) {

plot(time, x, type = "b")
points(time[(kf + 1):(n - kf)], smooth, pch = "+")
lines(time[(kf + 1):(n - kf)], smooth)
}
tt <- time[(kf + 1):(n - kf )]
ti <- factor((tt - 1)%%ks + 1)
ss <- tapply((x[(kf + 1):(n - kf)] - smooth), ti, mean)
ss <- ss - sum(ss)/k
list(ma = smooth, seas = ss)
}
# to obtain seasonals= swindow3(x,k, pt=T)$seas

## We now add this code to extract the moving average and the
## estimated seasonal effects
trendplusts=swindow3(data, 4, pt=T) # We see the smoother first

# extract the moving average
trend = swindow3(data, 4, pt=T)$ma
trend
length(trend)
# extract the estimated seasonal effect
seasonaleffect=swindow3(data, 4, pt=T)$seas
seasonaleffect
# add seasonal effect and moving average
length(data)
# create sequence of seasonal effects
Shat=rep(seasonaleffect, 24/4)
Shat

## add seasonal effects to moving average
length(trend)
trendvar=rep(NA,24)
trendvar[3:22]=trend
trendvar


TrendPlusShat=Shat + trendvar
TrendPlusShat

# View raw data and fitted Trend plus Seasonal effect in one plot
plot(time, data, type = "b")
points(time, TrendPlusShat, pch = "+",type="l", col="red")

## Obtain the seasonally adjusted and detrended time series and plot it

SeasAndDet=data- TrendPlusShat
SeasAndDet

### create data frame that will be used when doing the two Y-axes plot
```

```
newdataframe=data.frame(time, data, trendvar, Shat, TrendPlusShat,SeasAndDet)
newdataframe

###  install.packages("latticeExtra")
library(latticeExtra)


## create data frame
time<-seq(1,length(x))
newdataframe=data.frame(time, data, SeasAndDet[3:22]=SeasAndDet)

## We use code from
## https://r-graph-gallery.com/145-two-different-y-axis-on-the-same-plot.html

obj1 <- xyplot(data ~ time, newdataframe,
type = "l" , lwd=2, col="steelblue")
obj2 <- xyplot(SeasAndDet ~ time, newdataframe,
type = "l", lwd=2)

# --> Make the plot with second y axis:
doubleYScale(obj1, obj2,
text = c("rawdata", "detrendedandseasadjusted"),
add.ylab2 = TRUE )
```

(b) As we notice in the `doubleYscale()` plot, clearly the detrended and seasonally adjusted time series fluctuates randomly around zero with not particular pattern and no regular seasonality.

Seasonally adjusting and detrending the data has removed all indications of trend or regular seasonality. This, again, does not mean that the seasonally adjusted and detrended data (the random term of the decomposition or data-trendestimate-seasonaleffect) is devoid of signal. In fact, if we do the ACF of such data, we will see that there is significant autocorrelation at lag 2. But ACF has not yet been explained in Chapter 2. After Chapter 3 is done, the instructor can revisit this example to reinforce these concepts (easily forgotten by students) by adding the following code to the decompose() version of the exercise.

```
data=c(322, 144, 472, 821, 408, 247, 626, 925, 434, 259, 681, 1277,
829, 435, 940, 1639, 1222, 592, 1055, 2000, 1278, 768, 1415, 2417)


#### Using decompose() function

## First, make data a ts()

data.ts = ts(data, start=c(1985, 2), freq=4)
par(mfrow=c(2,1))
acf(data.ts)
acf(decompose(data.ts)$random, na.action=na.omit)
acf(data.ts)
```

□

2.4.1 Exercises

Exercise 2.10

Using the program *ch2sodasales.R* all you have to do is add the following lines at the end of the program to see the answer.

```
## must run program ch2sodasales.R before running the following lines

random=decompose(soda,type="mult")$random
product=trend*seasonal*random  #trend and seasonal obtained earlier in the program
cbind(soda, product)  # You will see that the product equals the time series.
```

You will notice that, except for the NA caused by losing data when we smooth with the MA filter, the product of the random component, the seasonal effect component and the trend component equals the true values of the soda time series.

□

Exercise 2.11

Using the program *ch2sodasales.R* all you have to do is add the following lines at the end of the program to see the answer, which will be 0.9855382.

Alternatively, you could also just look at the output of decompose(soda) in the program, since the output is not large. The additional lines given below will be helpful when the data set is large.

```
## must run program ch2sodasales.R  before running the following lines
# install.packages("TSstudio")
library(TSstudio)

## We use TSstudio to convert the ts data to a data frame that lubridate package
## can help us with later.

df=ts_to_prophet(soda)  # a data frame, where the date is now labeled ds
df   # view the data set now in a data frame format
class(df)

# include the seasonal effect from decompose as variable in the new data frame

df$seasonal=seasonal
df

# install.packages("lubridate)  # uncomment if not installed yet.
library(lubridate)
```

```
## because we put the data in data frame format, lubridate can help extract month, year

df$year=year(df$ds)
df$month=month(df$ds) # 1 is January, 2 is february
df

## so now you may select June 1991 to obtain the answer.
df$seasonal[df$year==1991 & df$month==6]

## since small data set, check all June months by viewing the whole data set
df
```

□

Exercise 2.12

Running the following short Base R program, you can obtain the components of the multiplicative decomposition and the data.

```
#AirPassengers is a ts object
# install.packages("TSstudio")
library(TSstudio)

## We use TSstudio to convert the ts data to a data frame that lubridate package
## can help us with later.

df=ts_to_prophet(AirPassengers)  # a data frame, where the date is now labeled ds
df   # view the data set now in a data frame format
class(df)

# include the trend,  seasonal effect and random effect
# from multiplicative decompose as variable in the new data frame

df$trend=decompose(AirPassengers, type="mult")$trend
df$seasonal=decompose(AirPassengers, type="mult")$seasonal
df$random=decompose(AirPassengers, type="mult")$random
df

# install.packages("lubridate)  # uncomment if not installed yet.
library(lubridate)

## because we put the data in data frame format, lubridate can help extract month, year

df$year=year(df$ds)
df$month=month(df$ds) # 1 is January, 2 is february
df
# you now have the decompose components, the data and the date in the same format as in
# the tables seen in the Chapter for the manual calculations.
```

Now we will do the manual calculations for the moving average trend, the seasonal effect and the random term of July 1949 and compare with the output of the program. The reader can then do the next values.

```
ds      y   trend     seasonal  random  year   month
.........................................
..........................................
6   1949-06-01 135   NA     1.1127758        NA 1949     6
7   1949-07-01 148 126.7917 1.2265555 0.9516643 1949     7
8   1949-08-01 148 127.2500 1.2199110 0.9534014 1949     8
9   1949-09-01 136 127.9583 1.0604919 1.0022198 1949     9
10  1949-10-01 119 128.5833 0.9217572 1.0040278 1949    10
11  1949-11-01 104 129.0000 0.8011781 1.0062701 1949    11
```

To obtain the first trend value in the output (which is a centered moving average), we do the following:

Step 1: $\frac{1}{12}(112 + 118 + 132 + 129 + 121 + 135 + 148 + 148 + 136 + 119 + 104 + 118) = 126.6667$

$\frac{1}{12}(118 + 132 + 129 + 121 + 135 + 148 + 148 + 136 + 119 + 104 + 118 + 115) = 126.9167$

Step 2: Take the average of those two.

$(126.6667 + 126.9167)/2 = 126.7917$

So that matches 126.7917 in the output for `1949-07-01`.

Since it is clear from this and subsequent calculations that the $\hat{T}$ component is the same as that given by trend in R's decompose, we can use R as a calculator now to do what we need to do to obtain the seasonal effect manually. Add to the program given earlier in this program the following:

```
# install.packages(dplyr) #uncomment if package not installed
library(dplyr)

df$swing=df$y/df$trend
df
df$ratio=tapply(df$swing,df$month,mean,na.rm=TRUE)
df
## the seasonal effect S-hat is the adjusted value of ratio.
## since the data starts in January, and we have value of ratio
## for that year, we can calculate the adjustment as
adjust=12/sum(df$ratio[1:12])
## with this we can create column with Shat, the seasonal effect.

df$Shat= df$ratio*adjust
df

## So the random component is obtained as follows:
df$random.manual= df$y/(df$Shat*df$trend)
df
```

and you will notice that the values in column *Shat* are the same as the value in column *seasonal* (from R's decompose) and the values in column *random.manual* are the same as the values in column *random* (from R's decompose)

□

Exercise 2.13

The following Base R program will allow you to find the answer.

```
## Preparing the data
rooms=scan("rooms.txt")
head(rooms); str(rooms) # check data
rooms=ts(rooms, start=c(1977,1), frequency=12)

## part (a)

plot(decompose(rooms, type="mult"))
title("Multiplicative decomposition of the raw rooms data")

plot(decompose(log(rooms)))
title("additive decomposition of the log of rooms data")

## part (b)

par(mfrow=c(2,1))
boxplot(rooms~cycle(rooms),
main="Seasonal boxplots of raw rooms ts",
xlab="Month (1=January, ...., 12=December)")
boxplot(log(rooms)~cycle(rooms),
main="Seasonal boxplots of logged rooms",
xlab="Month (1=January, ...., 12=December)")
dev.off()
```

(a) In both cases, the random term is fluctuating around a constant value with no trend in mean or variance, leading to the conclusion that both approaches (additive decomposition of the logged rooms and multiplicative decomposition of the raw rooms result in similar outcome for the random term. This also leads to tentatively conclude that the seasonal swing in the raw rooms data is proportional to the trend. This makes sense since, as indicated in the chapter, time series concerning human populations tend to increase in variability proportionally to trend due to the increase in the size of the population.

(b) The conclusion reached from both seasonal boxplots is the same. The big seasonal increase occurs in August in both plots, since the median in August is higher than any other median in the plot. Then there is another minor seasonal increase in December and January. This makes sense. Those are the seasons in the Northern hemisphere when most people travel and occupy hotels.

□

2.5.1 Exercises

Exercise 2.14

The seasonal effects for acceleration obtained in Table 2.2 were found by first finding a moving average trend, then subtracting that trend from the data to obtain the seasonal swings, and then averaging the seasonal swings of each month and adjusting to obtain the following seasonal effects for the following quarters:

$\hat{S}_{Q1}$: 3.89895; $\hat{S}_{Q2}$: 0.238375; $\hat{S}_{Q3} = -4.041425$; $\hat{S}_{Q4} = -0.0959$

On the other hand, the seasonal coefficients obtained with the regression program in the fist part of Program *ch2regsmoothers.R*, the coefficients corresponding to Q2, Q3, Q4 are not the seasonal effects. Those coefficients represent by how much (or less) the seasonal of q2, for example (for coefficient with q2), is above (or below) the seasonal of q1. Interpreting the coefficients of q3 and q4 similarly, we can then see that those regression coefficients measure something different from $\hat{S}$ in Table 2.2. In order to obtain the seasonal effects, we would have to add to the coefficient of each quarter the value of the intercept corresponding to the seasonal of q1. Since the regression model has also the time trend as independent variable, that seasonal is not obvious as the constant does not represent in this case just the seasonal of q1.

□

Exercise 2.15

With the first regression approach presented in Program *ch2regsmoothers.R*, where dummies were defined for the seasonality, the forecasts are:

1965Q2 forecast: 12.17343 1965Q3 forecast: 7.81635 1965Q4 forecast: 11.78298 1966Q1 forecast: 13.71518

With the approach recommended in this problem, adding to the estimated regression trend the seasonal effects of Table 2.2, the forecasts are:

1965Q2 forecast: 11.092015 1965Q3 forecast: 7.326765 1965Q4 forecast: 11.786840 1966Q1 forecast: 16.296240

There isn't much difference between the forecasts obtained with these methods but they are not identical.

The following R program was used to obtain the last forecasts, after reading the data with Program *ch2regsmoothers.R*.

```
regmodel = lm(y~time)
summary(regmodel)
That=fitted(regmodel)

#We forecast the trend of four future unknown quarterly values
# Yhat future
forecast.T.hat=c(rep(0,4))
for(j in 1:4){
forecast.T.hat[j] = 0.04809+0.51455*(20+j)
}

# we can then add to forecast the seasonal effects from Table 2.2

forecast.yhat=0
forecast.yhat[1] = forecast.T.hat[1]+ 0.238375 # for 1965Q2 forecast
forecast.yhat[2] = forecast.T.hat[2]-4.041425 # for 1965Q3 forecast
forecast.yhat[3] = forecast.T.hat[3]-0.0959 #for 1965Q4 forecast
forecast.yhat[4]= forecast.T.hat[4]+3.89895 # for 1966Q1 forecast

rbind(forecast.yhat[1], forecast.yhat[2], forecast.yhat[3], forecast.yhat[4])
```

□

Exercise 2.16

(a) The expected values are obtained by plugging 1, 2, 3, 4 in linear trend equation, for example, the prediction for the first quarter of 2020 is $\widehat{T} = 500 + 50 * 1 = 550$. Doing the same for all four, we get.

550, 600, 650, 700

(b) We an obtain the predicted values by adding the seasonal effect to the forecasts based on just the trend. The calculations are

550*0.4=220, 600*1.6=960, 650*1.2=650, 700*0.8=560

(c) $\widehat{T} = 500 + 50 * 5 = 750$

$\widehat{T} = 500 + 50 * 6 = 800$

$\widehat{T} = 500 + 50 * 7 = 850$

$\widehat{T} = 500 + 50 * 8 = 900$

Estimation of the trend plus seasonal can be obtained by multiplying by the seasonal:

$\widehat{y} = 750 * 4$

$\widehat{y} = 800 * 1.6$

$\widehat{y} = 850 * 1.2$

$\widehat{y} = 900 * 0.8$

2.6.2 Exercises

Exercise 2.17

The forecasting equation would be,

$\hat{y}_{t+1} = \hat{\mu}_{t+1}(t) = \hat{\mu}_t(t-1) + 0.2(y_t - \hat{\mu}_t(t-1))$

Consequently, and given that $\hat{\mu}_{23}(22) = a = 321$,

$\hat{y}_{24} = \hat{\mu}_{24}(23) = \hat{\mu}_{23}(22) + 0.2(y_{23} - \hat{\mu}_{23}(22)) = 321 + 0.2(362 - 321) = 329.2$

$\hat{y}_{25} = \hat{\mu}_{25}(24) = \hat{\mu}_{24}(23) + 0.2(y_{24} - \hat{\mu}_{24}(23)) = 329.2 + 0.2(314 - 329.2) = 326.16$

$\hat{y}_{26} = \hat{\mu}_{26}(25) = \hat{\mu}_{25}(24) + 0.2(y_{25} - \hat{\mu}_{25}(24)) = 326.16 + 0.2(365 - 326.16) = 333.928$ □

Exercise 2.18

The reader will recognize that this problem is very similar to Exercise 2.17, but there is something given in that exercise for that exercise that is not given here for this new exercise. Both exercises give the $\hat{\alpha}$, the estimate of α obtained with simple exponential smoother. But we are not given the a_{22}, that is, the prediction for $t = 22$ given data up to $t = 21$. Therefore, we can not proceed with a forecast unless we assume an initial value to start with. Look at Table 2.6 though. We had as initial predicted value there the first observation. So we can pretend that in our new exercise, the first observation $x_{22} = 343$ is the initial value, that is, $a_{22} = x_{22} = 343$. Using this assumption,

$\hat{x}_{23} = \hat{\mu}_{23}(22) = \hat{\mu}_{22}(21) + 0.0344(x_{22} - \hat{\mu}_{22}(21)) = 343 + 0.0344(343 - 343) = 343$

That will be R's a_{23}.

With this, we can predict

$\hat{x}_{24} = \hat{\mu}_{24}(23) = \hat{\mu}_{23}(22) + 0.0344(x_{23} - \hat{\mu}_{23}(22)) = 343 + 0.0344(362 - 343) = 343.6536$

$\hat{x}_{25} = \hat{\mu}_{25}(24) = \hat{\mu}_{24}(23) + 0.0344(x_{24} - \hat{\mu}_{24}(23)) = 343.6536 + 0.0344(245 - 343.6536) = 340.2599$

The reader will notice that once you obtain the forecast of $x$for $t = 25$, steps ahead forecast will not have data, and in that case you will have to use the past forecast as the observation being forecasted. This is an indication that for $t = 26, t = 27$ the forecasts will be the same as the forecast for $t = 25$. That is,

$\hat{x}_{26} = 340.2599 = \hat{x}_{27}$

□

Exercise 2.19

See Example 2.4, and Program and data files used there.

(a) The following R program will do the job

```
###Simulating results with non-optimal alpha
# you do that only to show how rmse
# will be higher if
# not using the optimal smoothing parameter.

# Read the cod data
cod=read.table("cod.txt")
attach(cod)
cod=cod[,2]  # select variable
cod.ts=ts(cod,start=c(1979,1),
end=c(1980,12),freq=12)
#Notice how we enter the value of alpha,
#something we would not do if looking for
# the optimal alpha as in Example 2.4.

cod.hw2=HoltWinters(cod.ts,alpha=0.1,
beta=FALSE,gamma=FALSE)  #change alpha each time
cod.hw2
cod.hw3=HoltWinters(cod.ts,alpha=0.3,
beta=FALSE,gamma=FALSE)
cod.hw3
cod.hw4=HoltWinters(cod.ts,alpha=0.5,
beta=FALSE,gamma=FALSE)
cod.hw4
cod.hw5=HoltWinters(cod.ts,alpha=0.7,
beta=FALSE,gamma=FALSE)
cod.hw5

cod.hw6=HoltWinters(cod.ts,alpha=0.7,
beta=FALSE,gamma=FALSE)
cod.hw6

## To obtain the optimal alpha, run program ch2simpleexpsmooth.R
## or read Example 2.4. Optimal alpha gives cod.hw1$SSE
## Following is the command we use there. Notice that we do not
## specify the alpha, letting R minimize the SSE to obtain it.

cod.hw1=HoltWinters(cod.ts,  # For optimal, recall, you do not put alpha at all
beta=FALSE,gamma=FALSE)

cod.hw1   # gives output

## Calculate RMSE for all the smoothers obtained and create table
SSE.all=c(cod.hw1$SSE,cod.hw2$SSE, cod.hw3$SSE,cod.hw4$SSE,cod.hw5$SSE,cod.hw6$SSE  )
RMSE.all= sqrt(SSE.all/length(cod.ts))
alpha.all=c(cod.hw1$alpha,cod.hw2$alpha,cod.hw3$alpha,cod.hw4$alpha,cod.hw5$alpha,
cod.hw6$alpha )
table=data.frame(alpha.all, RMSE.all)
table
```

After running the program given in this exercise, the reader will see in the table object that the lowest RMSE is the one obtained when we let R select the optimal alpha, the one from cod.hw1. The R software obtains the optimal alpha, which gives RMSE=34.37415. The optimal $\hat{\alpha}$ is the one with the lowest RMSE.

(b) We add to the program given in part (a)

```
par(mfrow=c(3,2))
plot(cod.hw2, main="alpha=0.1")
plot(cod.hw3, main="alpha=0.3")
plot(cod.hw4, main="alpha=0.5")
plot(cod.hw5, main="alpha=0.7")
plot(cod.hw6, main="alpha=0.95")
plot(code.hw1, main="Optimal alpha=0.04627398")  # the optimal one.
```

(c) After completing the plots, the reader will see in the figure that as the α increases, the exponential smoother tries to follow each single movement of the series, that is, it is replicating irregular observations, fitting noise, which is nonsense. The smoother should only replicate what is a signal in the data, the only thing that can be used to predict. That is why we use statistics, to extract signals from noisy data according to some optimality criterion.

□

2.6.4 Exercises

Exercise 2.20

$\hat{x}_5 = \hat{\gamma}_{0,5} + \hat{\gamma}_{1,5} = 191.9221 + 16.08943631 = 208.0115$

$\hat{x}_6 = \hat{\gamma}_{0,6} + \hat{\gamma}_{1,6} = 176.4625 + 9.95799391 = 186.4205$

□

Exercise 2.21

We may use Program *ch2trend-exp-smoother.R* to start, but adapting it to the `LakeHuron` time series.

```
####  R code to do Trend Corrected Exponential Smoothing.

?LakeHuron

LH = LakeHuron
class(LH)  ## already a ts object

#(b)
plot.ts(LH, main="Level of Lake Huron 1875-1972", ylab="Level(feet")

train=window(LH, end=1967)
test=window(LH, start=1968)


# (c)  HoltWinters command now
#will find optimal alpha and Beta. Notice how we do
## not specify those parameters, to let R find them

LH.hw=HoltWinters(train,gamma=FALSE)
LH.hw

LH.hw$SSE

## Find the RMSE
sqrt(LH.hw$SSE/length(train))


fitted=fitted(LH.hw)
fitted
# see the last training set gamma_0 and gamma_1
# (d) Forecast and compare forecast with actual values

forecasts=predict(LH.hw,n.ahead=5, prediction.interval = TRUE,
level = 0.95)
forecasts

#install.packages(TSstudio)
library(TSstudio)

## We reshape the data in order to create a data frame with date as variable in one column.
#The ts_to_prophet function is from library(TSstudio). It is a useful function to put
# the data as data frame, which will be convenient to compare data and forecast.

df.test.data=ts_to_prophet(test)  #ds is date, y is LakeHuron level in this exercise
df.test.data

df.forecast=data.frame(forecasts[,3], forecasts[,1], forecasts[,2])
df.forecast
df.compare=data.frame(df.test.data, df.forecast)
# add the forecasts to the data frame
names(df.compare)=c("Year", "TestSet level", "lwr", "forecast", "upr")
df.compare  # By viewing this you will find the answer to part (d)

### (e)

par(
mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)


plot(LH.hw, forecasts,lwd=1.5,cex=0.5, lty=1,
main="Level of Lake Huron.\n
Trend-corrected exponential smoothing",ylab="Level (feet)")
legend("topleft", legend=c("fitted and forecast"),
lty=1, col="red")
lines(test)
# dev.off()
```

(a) Base R got the data from author Brockwell and Davis. The data appears in two of this author's books.

(b) The time plot reveals that Lake Huron's level is decreasing over time.

(c) There are many predicted or fitted values in the training period. Running the program you will see them. They correspond to object `fitted`.

(d) The forecasts are as given next.

```
Year         data      lwr   forecast      upr
1 1968-01-01        578.52 576.8554 578.5201 580.1849
2 1969-01-01        579.74 576.0852 578.6603 581.2353
3 1970-01-01        579.31 575.3715 578.8004 582.2293
4 1971-01-01        579.89 574.6590 578.9405 583.2221
5 1972-01-01        579.96 573.9297 579.0807 584.2316
```

As we can see, in 1970, the level of Lake Huron was 579.31 feet. The forecasted value using HW exponential smoothing is 578.8004. We also observe that all values of time series data fall between the lower (lwr) and upper (upr) vaiues of the prediction interval, which is a good sign.

(e) Judging by the plot obtained with the code for this Exercise, the fit obtained with trend corrected HW follows the time series very closely. We did not plot the prediction intervals, but the reader is invited to make a separate plot containing only the forecast, the interval and the raw test data, which well labeled and colored with a nice legend will show more detail about the forecast.

□

2.6.6 Exercises

Exercise 2.22

The output for the three coefficients of the Holt-Winters trend-seasonal exponential smoothing are

$$\alpha = 0.77999853, \beta = 0.01999391, \gamma = 0.62089830$$

$$x_t = \gamma_{0t} + \gamma_{1t} + \gamma_{2t} + w_t$$

where γ_{0t}, γ_{1t} and γ_{2t} change slowly over time according to the smoothing equations:

$$\begin{aligned}
\widehat{\gamma}_{0t} &= 0.77999853(x_{t-1} - \hat{\gamma}_{2,t-p}) + (1 - 0.77999853)(\hat{\gamma}_{0,t-1} + \hat{\gamma}_{1,t-1}) \\
\widehat{\gamma}_{1t} &= 0.01999391(\hat{\gamma}_{0t} - \hat{\gamma}_{0,t-1}) + (1 - 0.01999391)\hat{\gamma}_{1,t-1} \\
\widehat{\gamma}_{2t} &= 0.62089830(x_t - \gamma_{0t}) + (1 - 0.62089830)\gamma_{2,t-p},
\end{aligned}$$

where, if the time series is monthly, p=12; if the time series is quarterly, then p=4, etc.

```
myairpassengers=as.numeric(AirPassengers)
head(myairpassengers)


sex4 <-function (x, a, b, d, s, k)
{
s1 <- s + 1
m <- x*0+mean(x[1:s])
bigc <- seq(s)
bigc <- x[1:s] - mean(x[1:s])
```

```
Tr <- m
Tr[1] <- (x[4] - x[1])/3
Tr[s]<-mean(x[2:s1])-mean(x[1:s])
xhat <- Tr
e <- m
for (j in s1:k) {
im <- j%%s
if(im==0) (im<-s)
m[j] <- a * (x[j - 1] - bigc[cyc((im - 1),s)]) + (1 - a) * (m[j -
1] + Tr[j - 1])
Tr[j] <- b * (m[j] - m[j - 1]) + (1 - b) * Tr[j - 1]
bigc[im] <- d * (x[j - 1] - m[j - 1]) + (1 - d) * bigc[im]

xhat[j] <- m[j] + Tr[j] + bigc[im]
e[j] <- x[j] - xhat[j]
}
sse <- crossprod(e[s1:k], e[s1:k])
sse
}


####

eforecast5<-function (yseries,s, k)
#this hooks minimizer to esmoother
# ie Holt Winters no seasonal
{
f <- function(p)
{a<-c(1,1)
a[1]<-exp(p[1])/(1+exp(p[1]))
a[2]<-exp(p[2])/(1+exp(p[2]))
a[3]<-exp(p[3])/(1+exp(p[3]))
sex4(yseries, a[1], a[2], a[3],2,k)
}


p <- c(0.5, 0.5,0.5)


m1<-  nlm(f, c(0.5, 0.5,0.5))
a<-m1$estimate[1]
a<-exp(a)/(1+exp(a))
b<-m1$estimate[2]
b<-exp(b)/(1+exp(b))
d<-m1$estimate[3]
d<-exp(d)/(1+exp(d))
print(c(a,b,d))
mm<-length(yseries)
time<-seq(mm)
```

```
Tr<-yseries
m<-yseries
s1 <- s + 1
m <- m*0+mean(yseries[1:s])
bigc <- seq(s)
bigc <- yseries[1:s] - mean(yseries[1:s])

Tr <- m
Tr[1] <- (yseries[4] - yseries[1])/3
Tr[s]<-mean(yseries[2:s1])-mean(yseries[1:s])
xhat <- Tr
e <- m
for (j in s1:mm) {
im <- j%%s
if(im==0) (im<-s)
m[j] <- a * (yseries[j - 1] - bigc[cyc((im - 1),s)]) + (1 - a) * (m[j -
1] + Tr[j - 1])
Tr[j] <- b * (m[j] - m[j - 1]) + (1 - b) * Tr[j - 1]
bigc[im] <- d * (yseries[j - 1] - m[j - 1]) + (1 - d) * bigc[im]

xhat[j] <- m[j] + Tr[j] + bigc[im]
e[j] <- yseries[j] - xhat[j]
gammahats[j,]=c(j, m[j], Tr[j], bigc[im])  # gives errors...
}
plot(time,yseries,type="b")
fseries<-c(yseries[1:s],xhat[s1:mm])
points(time,fseries,pch="+")
xhat
}


cyc<-function(i,s)
{if (i>0) (y<-i)
else (y<-(s+i))
y
}

### We type our own code now.

n=length(myairpassengers); n

gammahats=matrix(0,nrow=143,ncol=4)

## Note that we added to the function eforecast5 code to enter the
# t,  m (gammahat0t), the Tr (gammahat1t) and the bigc( gammahat2t)
# into the matrix, so that we can answer the exercise.

eforecast5(myairpassengers, 12,n)
```

□

Exercise 2.23

See Section 2.6.1 and Example 2.4 and the program *ch2simpleexpsmooth.R*. The following framed R code is based on that script file

```
# set working directory to CH2-CODE-DATA-BASER folder in timeseriestime.org
cod=read.table("cod.txt") # A very short time series
cod



### (a) Find mean
mean(cod[,2])

### (b)
# make the data a time series object and plot it.
cod.ts=ts(cod[,2],start=c(1979,1),end=c(1980,12),freq=12)
plot(cod.ts,ylab="Monthly Cod Catch (in Tons)")

### (c) Simple exponential smoothing

cod.hw1=HoltWinters(cod.ts,beta=F,gamma=F)
cod.hw1
cod.hw1$SSE
cod.hw1$coef
plot(cod.hw1)

cod.fitted=cod.hw1$fitted
cod.fitted
####(d) Forecast

forecast.cod=predict(cod.hw1,n.ahead=4)
forecast.cod
par( mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)
ts.plot(cod.ts,forecast.cod,lty=1:2, col=c("black", "red"), lwd=c(3,3),
main="Simple Exp smoother forecast of cod")
legend("topright", legend=c("data","forecast"), col=c("black","red"),
lty=c(1,2),lwd=c(3,3))

### (e)
```

```
######  Non-optimal alpha. We should notice that the SEE is
## larger when a nonoptimal alpha is used.

cod.hw2=HoltWinters(cod.ts,alpha=0.3,beta=F,gamma=F)
cod.hw3=HoltWinters(cod.ts,alpha=0.5,beta=F,gamma=F)
cod.hw5=HoltWinters(cod.ts,alpha=0.7,beta=F,gamma=F)

## See how the plot looks when fitting with alpha=0.3
cod.hw2=HoltWinters(cod.ts,alpha=0.3, beta=F,gamma=F)
cod.hw2
cod.hw2$SSE
cod.hw2$coef
plot(cod.hw2)
```

(a) The average monthly cod catch is 351.2917 tons.

(b) There is no visible trend, and the time series fluctuates randomly around its mean, without explicit pattern.

(c) If x_t is the value of cod catch at time t, then the model is

$\hat{\mu}_{t+1}(t) = 0.04627x_t + (1 - 0.04627)m\hat{u}_t(t - 1)$

See page 91

$hat\mu_{24}(t) = 0.04627x_{23} + (1 - 0.04627)m\hat{u}_t(23) = 0.04627(314) + (1 - 0.04627)354.7772 = 352.8904$

$a_{1980,12} = 352.8904$

$MSE = SSE/24 = 1181.582$

(d) The forecasts for the first four months are:

```
Jan      Feb      Mar      Apr
1981 353.4506 353.4506 353.4506 353.4506
```

We obtain them by updating the equation given in part (c). After the first forecast, which is given by the value of a in the output of the statement

```
cod.hw1
```

, the predictions are constant and equal to a as there is no new data to be updated.

(e) With $\alpha = 0.3$ the SSE is much larger (SSE=33177.3) than with the optimal (which is SSE=28357.97). In fact, any alpha that is not the optimal will give larger SSE, which makes sense since the optimal is the one that minimizes the SSE.

In addition to that, we notice that the fit is trying to following the noise, overfitting.

□

Exercise 2.24

The `JohnsonJohnson` data set is a ts object that comes with Base R. Therefore, all we have to do is window it to create the training and test set, and then apply the Base R HW exponential smoothing function for trend and seasonal.

```
? JohnsonJohnson   # find out what is being measures
## Not a large time series so look at it

JJ= JohnsonJohnson
JJ
class(JJ)  # notice that it is a ts object
plot.ts(JJ)  # quick time plot view

start(JJ); end(JJ); frequency(JJ)
training=window(JJ, end=c(1977,4), frequency=4)
test=window(JJ, start=c(1978,1), frequency=4)

### (a)

JJ.hw1= HoltWinters(training)
plot(JJ.hw1)
JJ.hw1$coef


forecast.JJ=predict(JJ.hw1,n.ahead=12)
forecast.JJ

cbind(test, forecast.JJ)
###### (b)
# Produce plot like that in Figure 2.12

JJ.fitted=JJ.hw1$fitted
JJ.fitted
par( mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)
ts.plot(JJ,forecast.JJ, lty=1:2, col=c("black", "blue"), lwd=c(3,3),
main="Simple Exp smoother forecast of JJ")
lines(JJ.fitted[,1], lty=3,col="red", lwd=3)
legend("topleft", legend=c("training and test data","fitted", "forecast"),
col=c("black","red","blue"),
lty=c(1,3,2),lwd=c(3,3, 3))
# calculating RMSE of forecast
RMSE.forecast=sqrt((sum(test-forecast.JJ)^2)/12)
RMSE.forecast

#### (c)

plot(JJ.fitted)  # decomposition.

#### (d)
```

```
## We might be tempted to do the regression using the
## components of exponential smoothing. But we do not know
## what those components are in the future, so we would not
## be able to give the regression the independent variables for
## the future. We can just fit the training set. For that, the
## training length is larger than JJ.fitted, due to the
## calculations done for the latter. So to do regression,
## we window the training data. The following code would fit
## the window of the training data.

train.for.reg=window(training, start=c(1961,1),frequency=4)

regmodel= lm(train.for.reg~JJ.fitted[,1]+JJ.fitted[,2]+JJ.fitted[,3])
summary(regmodel)   # we will not use this model.
# plot(regmodel)


##### Alternative regression that allows us to forecast.

## Create the independent variables

### We take the seasonal effect obtained from decomposition.
## We apply the decomposition to the whole data because
## we want to have the values of the seasonal effect for the test
## period. We use multiplicative because the seasonal increases
## with the trend.


seas=decompose(JJ,type="mult")$seasonal
seas

## We create a variable that will allow us to consider
## an exponential trend: trend=exp(t)

a=seq(1:length(JJ))
t=ts(a, start=c(1960,1), frequency=4)

## We then split both independent variables.

seas.training=window(seas, end=c(1977,4), frequency=4)
seas.test=window(seas, start=c(1978,1), frequency=4)
trend.training=window(t, end=c(1977,4), frequency=4)
trend.test= window(t,start=c(1978,1), frequency=4)

# regression model fitted to training data after some trial and error.
# Note that because the seasonal effect is multiplicative, the
# independent variables would be multiplied. To have linear regression
## we take logs. Log of exp(trend) is equal to trend.
## Raw model is: training=(seasonal effect)*trend
```

```
# In log form: log(training)=(log seasonal effect)+ (log trend)




model5=lm(log((training)) ~(trend.training)
+log((seas.training)))
summary(model5)

resid=ts(model5$residuals, start=c(1960,1), frequency=4)

plot.ts(resid)
abline(h=0)  # residuals have some pattern.

acf(resid) # the model does not capture all the seasonalities. Not great.

# we calculate the fitted value in the original scale of the data
## the fitted are given as log, so we must unlog them.

fitted=ts(exp(model5$fitted), start=c(1960,1), frequency=4)

## Check the fit visually

plot.ts(training, main="Fitted values (red) and training (black)",
ylab="Earnings ($)")
lines(fitted, col="red")

## prepare the independent variables needed for the forecast.
## They are their test set values, but must give the same name
## as in the original regression.

out.of.sample.data=data.frame(trend.training=trend.test,
seas.training=seas.test)
out.of.sample.data

## Obtain the forecasts

forecastlist=predict(model5, 12,
newdata=out.of.sample.data )

forecastlist

forecast=exp(forecastlist$fit) #extract the forecasted values

forecast.ts=ts(forecast, start=c(1978,1), frequency=4)

plot.ts(JJ,lty=1,
col=c("black"),type="l",
xlab="Time", ylab="Earnings ($)",  cex=1.5,pch=c(2),
main="JJ fitted and forecasted with multiple regression")
lines(fitted, col="red", lty=2)
lines(forecast.ts,col="blue",lty=2,cex=1.5,pch=c(2))
legend("topleft", c( "Training and test", "fitted", "forecast"),
col=c("black", "red", "blue"),lty=c(1,2,2))

# Calculate the RMSE
RMSE1.forecast=sqrt((sum(test-forecast.ts)^2)/12)
RMSE1.forecast

RMSE1.forecast=sqrt((sum(as.numeric(test)-as.numeric(forecast.ts))^2)/12)
RMSE1.forecast


#### Calculate the RMSE of the average forecast

average= (forecast.ts + forecast.JJ)/2

check=cbind(average,forecast.ts, forecast.JJ, test )
check

RMSE.average.forecast=sqrt((sum(test-average)^2)/12)
RMSE.average.forecast

#compare forecasts
plot.ts(test,ylab="Earnings ($)", main="Comparing forecasts")
lines(average, col="blue")
lines(forecast.JJ, col="green")
lines(forecast.ts, col="maroon")
legend("topleft", c( "Average forecast", "Exp smoothing forecast",
"Regression forecast"),
col=c("blue", "green", "maroon"),lty=c(1,2,2))
```

(a) Apply trend and seasonal Holt-Winters exponential smoothing to the `JohnsonJohnson` time series. Program *ch2trend-exp-smoother.R* is a good way to start, focusing on the bottom portion of the program. Forecast three years ahead (12 quarters).

The forecasts compared with the test data values are contained in the following table.

```
test    forecast.JJ
1978 Q1 11.88   10.696291
1978 Q2 12.06   11.479833
1978 Q3 12.15   10.904330
1978 Q4  8.91    9.924309
1979 Q1 14.04   11.993735
1979 Q2 12.96   12.777277
1979 Q3 14.85   12.201773
```

```
1979 Q4  9.99   11.221752
1980 Q1 16.20   13.291178
1980 Q2 14.67   14.074720
1980 Q3 16.02   13.499217
1980 Q4 11.61   12.519196
```

(b) Run the R program to see the plot.

With trend and seasonal exponential smoothing, the forecast RMSE is 3.1051

(c) We observe that the level is increasing exponentially, the trend seems to have shifted in 1970.

(d) Fit an appropriate regression model to the training set of data, like the example in the Regression smoothers section of this chapter. Forecast 12 quarters ahead and compare the RMSE of the forecast obtained with the regression with that obtained with the trend and seasonal exponential smoothing. Which one produced the lowest RMSE? Notice that it is very common practice in forecasting to fit a gallery of models to the same time series and then compare the forecasts errors.

The RMSE with the regression model that we fit is: 5.579191, which is larger than the one obtained with exponential smoothing in part (b).

The model fitted used as independent variables an exponential trend and the seasonal effects of a multiplicative decomposition and had the following original form:

$Y = (e^t) * (seasonaleffect)$

where Y is the training data, t is a time variable. We logged all components of this model.

(e) The RMSE of the average forecast is 1.237, much lower that either single forecast.

□

Exercise 2.25

As examples to get started using fpp3 to do classical decomposition and exponential smoothing for cases that we have already studied in Chapter 2 with Base R programs, see programs: *ch2sodasales-fpp3version.R* and *ch2simpleexpsmooth-fpp3version.R*

□

2.7.1 Exercises

Exercise 2.26

We will present here an example of the use of `prophet`.The R program that follows, if run, will illustrate what needs to be done to forecast with Facebook's prophet. Run it function by function to avoid missing details. In the first part of the program, we will not change the default arguments of the function prophet to see what we get. The default fit fits an additive seasonal. In the second part of the program, we change one of the arguments (parameters in ML language) by making seasonality.mode="multiplicative."

```
#install.packages("prophet") # remove # if not installed and run
library(prophet)
#install.packages("TSStudio") # remove # if not installed and run
library(TSstudio) # has the function ts_to_profit() helpful to convert a
# ts object to a data frame like those prophet likes.

?prophet()

# Will forecast AirPassengers with prophet.
AP = AirPassengers
AP # it is small data set, so look at it
start(AP); end(AP)
length(AP)

#Split the data into a training set to fit model and test set to compare with forecast
training=window(AP, end=c(1958,12)) # training 1949:1-1958-12
test=window(AP,start=c(1959,1))  # test 1959:1- 1960:12

#Convert the training and test data to a format that the prophet function accepts
df= ts_to_prophet(training)
df_test=ts_to_prophet(test)

# Invoke prophet() without changing the default arguments of the function
m=prophet::prophet(df)
m

# Creates a data frame where to put prophet's fitted values
# for AP and the forecast
future=make_future_dataframe(m,periods=24, freq="month")
head(future)
tail(future)

#  Fit the model and forecast the 24 months of the test period.
forecast=predict(m, future)    # This is a data frame

# View some of the produced fitted values and the forecast.
# Notice that a prediction interval accompanies the point fitted
#and point forecast
head(forecast[c("ds","yhat","yhat_lower","yhat_upper")], n=12)
tail(forecast[c("ds","yhat","yhat_lower","yhat_upper")], n=12)
class(forecast)

# plot the fitted and the forecast. The dots are the data.
plot(m,forecast,xlab="Time",
ylab="Number of airline passengers" )

#Extract the forecasted values only
last24forecast=forecast$yhat[121:144]
last24forecast
```

```
# Plot just the forecast and the test data
plot(ts(df_test[,2],start=c(1959,1)), ylab="Test set number of passengers")
lines(ts(last24forecast, start=c(1959,1)),col="red")
lines(ts(forecast$yhat_lower[121:144], start=c(1959,1)), col="blue")
lines(ts(forecast$yhat_upper[121:144], start=c(1959,1)), col="blue")

## Calculate the root mean square error of the forecast.
rmse_prophet= sqrt(mean((df_test$y-last24forecast)^2))
rmse_prophet

#####################################
# Since the additive seasonality
#does not give very good forecast, we can try
# now with multiplicative
############################################

## Note if RStudio complaints then close it and
## start again running from now on.

#install.packages("prophet") # remove # if not installed and run
library(prophet)
#install.packages("TSStudio") # remove # if not installed and run
library(TSstudio) # has the function ts_to_profit() helpful to convert a
# ts object to a data frame like those prophet likes.

?prophet()

# Will forecast AirPassengers with prophet.
AP = AirPassengers
AP # it is small data set, so look at it
start(AP); end(AP)
length(AP)

#Split the data into a training set to fit model and test set to compare with forecast
training=window(AP, end=c(1958,12)) # training 1949:1-1958-12
test=window(AP,start=c(1959,1))  # test 1959:1- 1960:12

#Convert the training and test data to a format that the prophet function accepts
df= ts_to_prophet(training)
df_test=ts_to_prophet(test)

# Invoke now prophet() changing only the seasonality.mode argument
m=prophet::prophet(df,
seasonality.mode="multiplicative")
m

# Creates a data frame where to put prophet's fitted values
# for AP and the forecast
future=make_future_dataframe(m,periods=24, freq="month")
```

```
head(future)
tail(future)

#  Fit the model and forecast the 24 months of the test period.
forecast=predict(m, future)    # This is a data frame

# View some of the produced fitted values and the forecast.
# Notice that a prediction interval accompanies the point fitted
#and point forecast
head(forecast[c("ds","yhat","yhat_lower","yhat_upper")], n=12)
tail(forecast[c("ds","yhat","yhat_lower","yhat_upper")], n=12)
class(forecast)

# plot the fitted and the forecast. The dots are the data.
plot(m,forecast,xlab="Time",
ylab="Number of airline passengers" )

#Extract the forecasted values only
last24forecast=forecast$yhat[121:144]
last24forecast

# Plot just the forecast and the test data
plot(ts(df_test[,2],start=c(1959,1)), ylab="Test set number of passengers")
lines(ts(last24forecast, start=c(1959,1)),col="red")
lines(ts(forecast$yhat_lower[121:144], start=c(1959,1)), col="blue")
lines(ts(forecast$yhat_upper[121:144], start=c(1959,1)), col="blue")

## Calculate the root mean square error of the forecast.
rmse_prophet= sqrt(mean((df_test$y-last24forecast)^2))
rmse_prophet
```

Judging by the plot of the forecast and what is going on with the fit observed in the plots for the last year, we can see that an additive model for the AirPassengers data set is not a good idea. A lot of the data points are outside the prediction intervals in the last years of the fit and the two years of the forecast. Following the first principles learned in Chapter 2, learned with classical decomposition and comments on exponential smoothing, we either do a multiplicative prophet model or we log the AirPassengers data set before we apply prophet.

In the code provided, when we change the seasonality.mode argument (called parameter in ML language) to multiplicative and we obtain a RMSE of 31.124, compared to an RMSE of 40 for the additive. However, we still notice that the prediction interval does not contain some of the data points. So tuning the other arguments might be necessary. Each domain of applications know what parameters matter for their situation. With just default, we many not do very well, as the example shown illustrates. The forecast is in line with the data with the default, but perhaps we can do better.

□

Completed Table 2.8. Data. Complete the table

Time	y_t	$ma4$	$\hat{T}$	$d = y_t - \hat{T}$	$\hat{S}$	$\hat{y}_t = \hat{T} + \hat{S}$
1989q1	293				70.5917	
1989q2	392				210.7667	
		263.25				
1989q3	221		275.125	−54.125	−76.95	198.175
		287				
1989q4	147		302	−155	−204.4083	97.5917
		317				
1990q1	388		325.25	62.75	70.5917	395.8417
		333.5				
1990q2	512		338.125	173.875	210.7667	548.8917
		342.75				
1990q3	287		354.125	−67.125	−76.95	277.175
		365.5				
1990q4	184		381.5	−197.5	−204.4083	177.0917
		297.5				
1991q1	479		405	74	70.5917	475.5917
		412.5				
1991q2	640		417.375	222.625	210.7667	628.1417
		422.25				
1991q3	347		435	−88	−76.95	358.05
		447.75				
1991q4	223		462.125	−239.125	−204.4083	257.7167
		476.5				
1992q1	581		484.375	96.625	70.5917	554.9667
		492.25				
1992q2	755		497.625	257.4	210.7667	708.3667
		503				
1992q3	410				−76.95	
1992q4	266				−204.4083	

2.8 Problems

Problem 2.1

We first complete the table manually.

We can see how we obtained the seasonal effects column $\hat{S}$ for Table 2.8 by calculating:

$$\bar{s}_{q1} = 62.75 + 74 + 96.6253 = 77.7917$$

$$\bar{s}_{q2} = 173.875 + 222.625 + 257.43 = 217.9667$$

$$\bar{s}_{q3} = -54.125 - 67.125 - 883 = -69.75$$

$$\bar{s}_{q4} = -155 - 197.5 - 239.1253 = -197.20833$$

$$\sum \bar{s} = 28.8001$$

$$(1/4)\sum \bar{s} = 7.2$$

Subtract this from the $\bar{s}s$.

$$\hat{S}_{q1} = 77.7917 - 7.2 = 70.5917$$

$$\hat{S}_{q2} = 217.9667 - 7.2 = 210.7667$$

$$\hat{S}_{q3} = -69.75 - 7.2 = -76.95$$

$$\hat{S}_{q4} = -197.20833 - 7.2 = -204.4083$$

Those seasonal effects that we obtained manually are the same as those obtained with Base R's function decompose (except for some rounding error). The reader may confirm that by running the R program that follows.

The estimate of the trend that we obtained for Table 2.8 is the same as that obtained with decompose() in Base R. Clearly, R's decompose(), like us, is using a centered 4-point moving average to estimate the trend and obtain the seasonal effects that follow.

The plot of the data and the fitted moving average model can be obtained by running the following R program.

Running the following program will also produce the plot containing the raw data, the seasonal adjusted data and the detrended data. The seasonally adjusted data show only the trend and random term. The detrended data shows only the seasonal effect and the random term.

```
## We enter the data and the yhat from Table 2.8
y=c(293, 392, 221, 147, 388, 512, 287,
184, 479, 640, 347, 223, 581, 755, 410, 266)

y.ts=ts(y, start=c(1989,1), end=c(1992, 4), frequency=4)

yhat=c(198.175, 97.5917, 395.8417, 548.8917, 277.175,
177.0917, 475.5917, 628.1417, 358.05, 257.7167,
544.9667, 708.3667)
yhat.ts=ts(yhat, start=c(1989,3), end=c(1992,2), frequency=4)

min(y); max(y)  # will to be able to see all in plot
min(yhat); max(yhat)

### Plot of the data with the fitted values superimposed

par(
mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)

plot.ts(y.ts,
main="Data and fitted additive decomposion \n
model  in Table 2.8",
ylab="y and yhat",
ylim=c(80, 780),
lwd=2, cex=1.5, lty=1, pch=1
)

lines(yhat.ts, col="red", lwd=3, cex=1.5, lty=2, pch=3)

legend("topleft", c("data (y)",  "fitted"),
col=c("black", "red"),lty=c(1,2), lwd=c(2,3))

# dev.off() If you uncomment and run, the plot will disappear

##### We now use decompose() and compare its
## trend and seasonal estimate

decompose(y.ts)

fitted=decompose(y.ts)$trend+decompose(y.ts)$seasonal
fitted  # The same values as those obtained in Table 2.8

## "seasonally adjusted data"= y-seasonal effect

seas.adjust= y.ts - decompose(y.ts)$seasonal
class(seas.adjust)

## "detrended data"=y-trend (but we lose some observations)

detrended=window(y.ts, start=c(1989,3), end=c(1992,2), frequency=4)-
decompose(y.ts)$trend
class(detrended)
min(detrended); max(detrended)

## Quick plot to visualize the raw original data and the other

par(
mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)

plot.ts(y.ts, main="Raw data", ylim=c(-250, 800), cex=1.5, lwd=2, lty=1)
lines(seas.adjust, main="Seasonally adjusted",
lty=2, col="blue", cex=1.5, lwd=2)
lines(detrended, main="detrended",
lty=2, col="purple", cex=1.5,lwd=2)

legend("topleft",
c("data (y)",  "seasonally adjusted only", "detrended only"),
col=c("black", "blue", "purple"),lty=c(1,2,2), lwd=c(2,2,2))
```

□

Problem 2.2

See also Exercise 2.24.

(a)

The time series starts in 1960, quarter 1 and ends in 1980, quarter 4. A training data set from 1960:1 to 1979:4 is created.

The time plot indicates upward trend with increasing variability and exponential kind of growth. There is a very mild seasonality. The random term of the multiplicative decomposition is stationary, but it shows volatility, indicating that the time series is conditionally heteroscedastic.

(b) R's seasonal effects added to the trend each quarter are:

Quarter 1: 0.2216094;

Quarter 2: 0.2439844

Quarter 3: 0.3087344

Quarter 4: -0.7743281

The seasonal effects we obtain manually are

seasonal effect quarter 1: 0.2245156

seasonal effect quarter 2: 0.2468906

seasonal effect quarter 3: 0.3116406

seasonal effect quarter 4: -0.7714219

As we can see, the manual ones are very close to those that R obtains. The small difference could be due to rounding error.

What I did to find those seasonal effects was to subtract from the time series the trend component first.

Y-T = random +seasonal

Then I took the average of (Y-T) for quarter one, then for quarter 2, etc. After that, I obtained a temporary seasonal effect. To make the seasonal effects add to 0, I calculated the average of the four seasonal effects, and adjusted the temporary seasonals by adding that adjustment.

The R code used to do the calculations requested is next.

```


######(a) ##############

#access it and check what type of R object it is
```

```
y=JohnsonJohnson  ; y
class(y)

### where it starts, where it ends, whether it is seasonal,
start(y); end(y); boxplot(y~cycle(y))

### Make a training data set that excludes the last year's 4 quarters.
y.training=window(y,start=c(1960,1), end=c(1979,4))
class(y.training); start(y.training); end(y.training)

## whether the time plot shows trend, and additive and multiplicative decomposition.

plot(y.training, main="Time plot of time series")
plot(decompose(y.training,type="additive"))
plot(decompose(y.training, type="mult"))

#####(b)###########

trend=decompose(y,type="add")$trend
seasonal=decompose(y,type="add")$seasonal
random=decompose(y,type="add")$random

## Manually, after doing the smoothing (the trend in the decomp)
## we would subtract Y-trend

y.minus.trend= y-trend
y.minus.trend  ## need to view the data to see what we are aggregating


## Then, we would take the average of all the Q1 y values, then the
## average of all the Q2 y values , and do the same for Q3 and Q4

temp.Q1= window(y.minus.trend,start=c(1961,1),end=c(1980,1),freq=TRUE)
temp.Q1
S1.temp=mean(temp.Q1); S1.temp

temp.Q2= window(y.minus.trend,start=c(1961,2),end=c(1980,2),freq=TRUE)
temp.Q2
S2.temp=mean(temp.Q2); S2.temp

temp.Q3= window(y.minus.trend,start=c(1960,3),end=c(1979,3),freq=TRUE)
temp.Q3
S3.temp=mean(temp.Q3); S3.temp

temp.Q4= window(y.minus.trend,start=c(1960,4),end=c(1979,4),freq=TRUE)
temp.Q4
S4.temp=mean(temp.Q4); S4.temp

sum=S1.temp+S2.temp+S3.temp+S4.temp; sum
```

```
adjustment=sum/4


cat("seasonal effect quarter 1 is", S1.temp+adjustment, "\n",
"seasonal effect quarter 2 is",S2.temp+adjustment, "\n",
"seasonal effect quarter 3 is", S3.temp+adjustment, "\n",
"seasonal effect quarter 4 is", S4.temp+adjustment)

#Compare to R's seasonal effect for each quarter

seasonal
```

□

Problem 2.3

We can view the image of the random term by plotting all the components.

```
# add is the default for argument type=, so no need to type it. But
# we will include for the sake of illustration.
plot(decompose(AirPassengers, type="add"))
title("\n Additive decomposition of the AirPassengers time series")
```

The plot of the random term at the bottom indicates that additive decomposition is not appropriate. The random term has very heterogeneous variance over time. In the first interval and last interval the fluctuations around a constant term are much larger than in the middle interval. This suggests that we should either (a) do multiplicative decomposition or (b) log the data before doing additive decomposition. Logging is appropriate when the variability increases or decreases with the trend.

□

Problem 2.4

Dataset `nhtemp` contains the mean annual temperature in degrees Fahrenheit in New Haven, Connecticut, USA, from 1912 to 1971. With a time plot, we can see that there is a slight trend upward in the data. The time series fluctuates randomly around that trend. I would suggest a trend corrected exponential smoothing.

The following program will provide the information needed to answer the questions.

```
?nhtemp
class(nhtemp)  # ts object with annual data
start(nhtemp);  end(nhtemp); frequency(nhtemp)

## it is a short series so you can view
nhtemp
plot.ts(nhtemp)  # quick time plot
```

```
## (a)  create training and test set
training=window(nhtemp,end=1970, frequency=1)
test=window(nhtemp, start=1971, frequency=1)

## Do trend corrected exponential smoothing

hw1= HoltWinters(training, gamma=F)
hw1
hw1$SSE
RMSE= sqrt(hw1$SSE/length(training))
plot(hw1) # View fit


fitted=fitted(hw1)
fitted  # View updated values equation values.


### -Exp smoothing decomposition

plot(fitted, main="HW decomposition of Temperature/n ")



#####
# (b) Forecast for 1971
#####


forecast=predict(hw1,n.ahead=1, prediction.interval = TRUE,
level = 0.95)
forecast  # gives the forecast for the test period, and the 95%
hw# upper and lower
# prediction intervals to indicate uncertainty of the forecast
```

(a) Let x denote temperature. The fitted smoother, starting with year 1914, is

$$\hat{x}_t = \hat{\gamma}_{0t} + \hat{\gamma}_{1t}, \tag{1}$$

and the updating (smoothing) equations are, after substituting for the smoothing (updating) parameters alpha and beta obtained with the program are:

$$\begin{aligned}\hat{\gamma}_{0t} &= \hat{\gamma}_{0,t-1} + 0.6479(x_{t-1} - \hat{\gamma}_{0,t-1}) + \hat{\gamma}_{1,t-1},\\ \hat{\gamma}_{1t} &= 0.3031(\hat{\gamma}_{0t} - \hat{\gamma}_{0,t-1}) + (1 - 0.3031)\hat{\gamma}_{1,t-1}\end{aligned}$$

(b) The forecasted value of level and trend are added to obtain the

$$\hat{x}_t = \hat{\gamma}_{0,1971} + \hat{\gamma}_{1,1971} = 52.007 = 51.8861 + 0.1146, \quad (2)$$

□

Problem 2.5

The information about the data (the metadata) can be found at `https://jse.amstat.org/datasets/bestbuy.txt`

The data pointed to by the url inside the R program given in the problem contains also the information on the forecasts done by the author of the Best Buy article. To simplify the analysis, and because it is such a small data set, we provide the values of MIPS in the program that follows (`bb.data`) and enter the dates by making the data a ts object a ts object that uses the time information that is given in `http://jse.amstat.org/datasets/bestbuy.dat.txt`

Following is an R program adapted for the new location and containing the exponential smoothing code for this problem.

We notice in the time plot produced that the seasonality is not very regular. We check decomposition to see what would be the dimension of such seasonality, and we find that the seasonal effect goes from approximately −20 to approximately 45, so it is not negligible. We look at the seasonal boxplot which confirms that. The trend and seasonal exponential smoother is then computed. Notice that we do not know the data beyond July 2007 (we do not split into training and test set),so we do not calculate the RMSE. To calculate the latter, we need to have data for the future

```
bb.data =c(145.2, 150.5, 153.1, 136.1, 152.6, 143.5, 145.9, 148.3, 159.1,
165.1, 167.5, 179.1, 192.9, 197.9, 212.0, 191.1, 219.5, 196.1,
212.0, 226.6, 214.1, 206.6, 218.3, 235.0, 266.4, 218.9,254.8,
269.1, 297.3, 274.5, 279.4, 292.3, 305.7, 304.9, 347.8, 358.0,
393.5, 428.2, 432.6, 453.8, 531.8, 472.3, 436.5, 470.4, 425.0,
416.6, 385.1, 488.5)
head(bb.data)
length(bb.data)
# make a ts object (notice that the program posted for Problem 2.5 in the
# book is missing by mistake the frequency argument, which is necessary
# because the time series is monthly. We correct that)
bb=ts(bb.data,start=c(1996,8), end=c(2000,7), frequency=12)
bb

plot.ts(bb, main="Best Buy computer usage",
ylab="MIPS")
## Seasonality is not obvious in the time plot. Look closer.
decompose(bb)
plot(decompose(bb))
boxplot(bb~cycle(bb), xlab="Time",
ylab="MIPS", main="Monthly Best Buy computer usage\n
is higher in June and July")

### We now do HoltWinters with seasonal and trend fitting

HW1= HoltWinters(bb)
HW1
fitted=fitted(HW1)
```

```
fitted
plot(fitted(HW1))

forecasts=predict(HW1,n.ahead=17, prediction.interval = TRUE,
level = 0.95)
class(forecasts)
forecasts

# Make a time series that contains the given data and
# the forecasts

new=c(bb.data, as.numeric(forecasts[,1]))
new.ts=ts(new,start=c(1996,8), end=c(2001,12), frequency=12)
new.ts

plot.ts(new.ts, lty=1,col="black",
main="HW trend exponential smoother",
ylab="MIPS")
lines(fitted[,1], lty=2,col="red",lwd=3)
lines(forecasts[,1],lty=2,col="blue",lwd=3) # point forecasts
lines(forecasts[,3], lty=5,col="green",lwd=3) # lower prediction interval
lines(forecasts[,2], lty=5,col="green", lwd=3) # upper prediction interval


legend("topleft",
legend=c("data","fitted","forecast",
"lower and upper forecast interval"),
col=c("black","red","blue","green","green"),
lty=c(1,2,2,5,5),
lwd=c(1,3,3,3,3))
```

□

2.9 Quiz

Question 2.1

”that the seasonal effect is constant for each season over time”

The seasonal effect is a constant that differs across seasons but for a given season is constant each year. The way we calculate implies that. If additive decomposition applies, then we add the seasonal effect to the trend. If multiplicative decomposition applies, then we multiply the seasonal effect to the trend and we say that the seasonal effect is proportional to the trend. See sections 2.3 and 2.4 in Chapter 2.

□

Question 2.2

Closest answer: ”the seasonal effect is proportional to the long term trend.”

We multiply the seasonal effect (which is constant for a given season each year) to the long term trend.

□

Question 2.3

”by adding the seasonal effect”

See section 2.2.2. □

Question 2.4

The seasonal swing is how much more or how much less than the trend value is the seasonal. In a multiplicative decomposition model (see Section 2.4), we would calculate the seasonal swing at each time t as indicated in the following program. Notice that we are not asking for the seasonal effect, but the swing value. The seasonal effect for a given season is constant each year, but the seasonal swing is not.

```
?UKgas  # find out the series' metadata
plot(UKgas)
UKgas
boxplot(UKgas~cycle(UKgas)) #inspect seasonal swings

a=decompose(UKgas,type="mult")
trend=a$trend
random=a$random

# seasonal swing value is calculated as follows
swing=UKgas/(trend)
swing
```

When we print the `swing` object, we notice how the values of the seasonal swing are different for a given quarter each year in contrast with the seasonal effect that we estimate with `decompose`. To see this graphically, the reader can execute the following program.

```
par(
mfrow=c(1,1),
font.axis=2,
mar=c(5,5,5,5),
font.main=2,
font.lab=2
)


plot(swing,lty=1,lwd=1.5,cex=0.5 )
seasonal=a$seasonal
lines(seasonal, col="red", lty=2,lwd=1.5,cex=0.5)
```

```
legend("topleft",
legend=c("seasonal swing","seasonal effect"),
col=c("black","red"),
lty=c(1,2),
lwd=c(1.5,1.5))


#to see the seasonal contrasting with increasing swing add
abline(h=max(seasonal))
abline(h=min(seasonal))
# The red seasonal effect is between the horizontal bars.
legend("topleft", legend=c("swing", "seasonal effect"), col=c("black","red"), lty=c(1,2))

# dev.off()
```

□

Question 2.5

an estimate of the seasonal swing

□

Question 2.6

4.4963

You may find out as follows, using R as a calculator.

```
y=ts(c(3.3602, -3.1769, 0.3484, 7.469, 4.4963), frequency=4)
T1=mean(c(3.3602, -3.1769, 0.3484, 7.469))
T2=mean(c( -3.1769, 0.3484, 7.469,4.4963))
mean(c(t1,t2))
```

□

Question 2.7

If there is a constant in the model, it will have 11 dummy variables. If there is no constant, it will have 12 dummy variables. □

Question 2.8

subtracting the value of the moving average from the data.

□

Question 2.9

In additive decomposition the trend, the seasonal swing and random term are added and in multiplicative decomposition they are multiplied, in both cases to recover the actual value of the time series.

□

Question 2.10

There is only one solution: ”When using moving average smoothers”

□