!Rmd_02_forecasting.Rmd

Diego Uchendu Learning from Prof Dr Thomas Kirschstein

01/12/2020

Time series demand example

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(forecast)
```

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

```r
library(expsmooth)
library(TTR)
library(fpp)
```

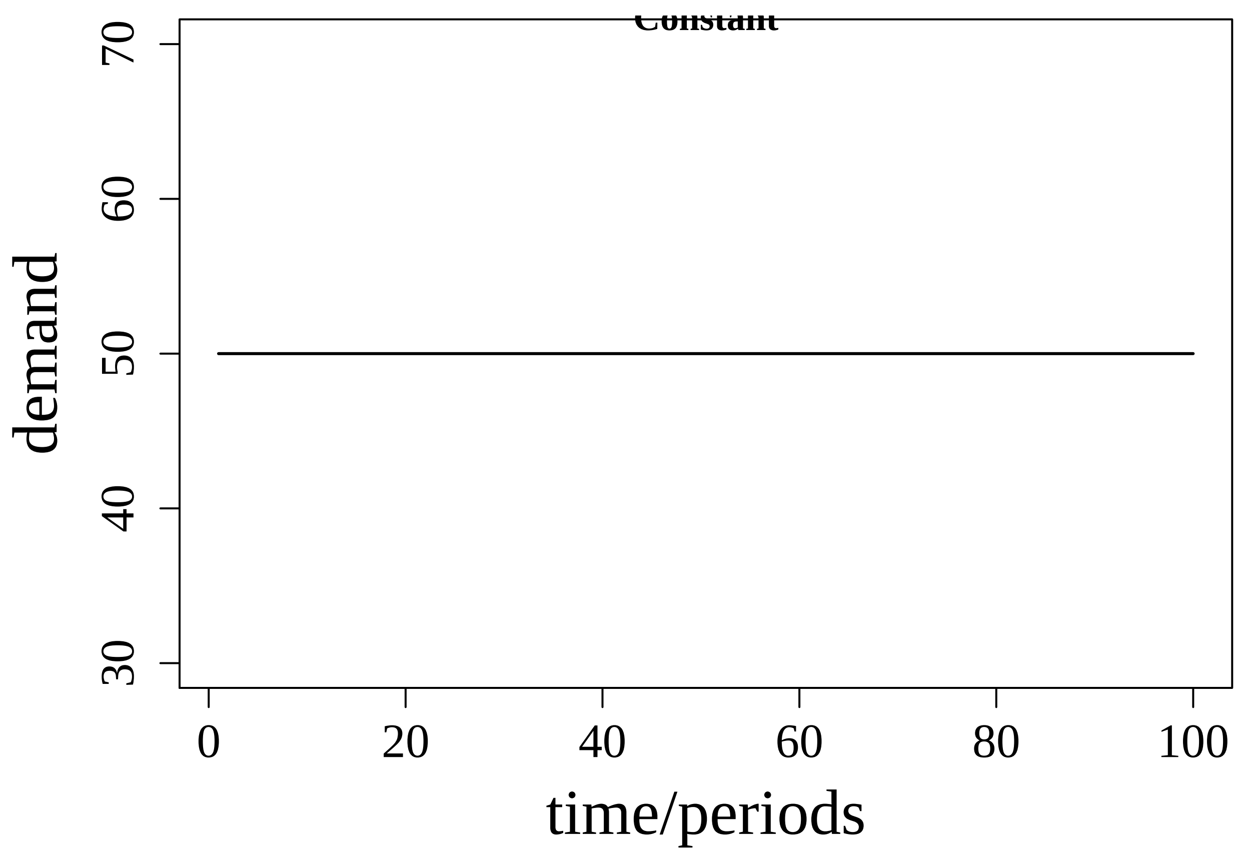
```
## Loading required package: fma

## Loading required package: lmtest

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
## Loading required package: tseries
```

```
library(fpp2)
```

```
##
## Attaching package: 'fpp2'

## The following objects are masked from 'package:fpp':
##
##     ausair, ausbeer, austa, austourists, debitcards, departures,
##     elecequip, euretail, guinearice, oil, sunspotarea, usmelec
```

Constant

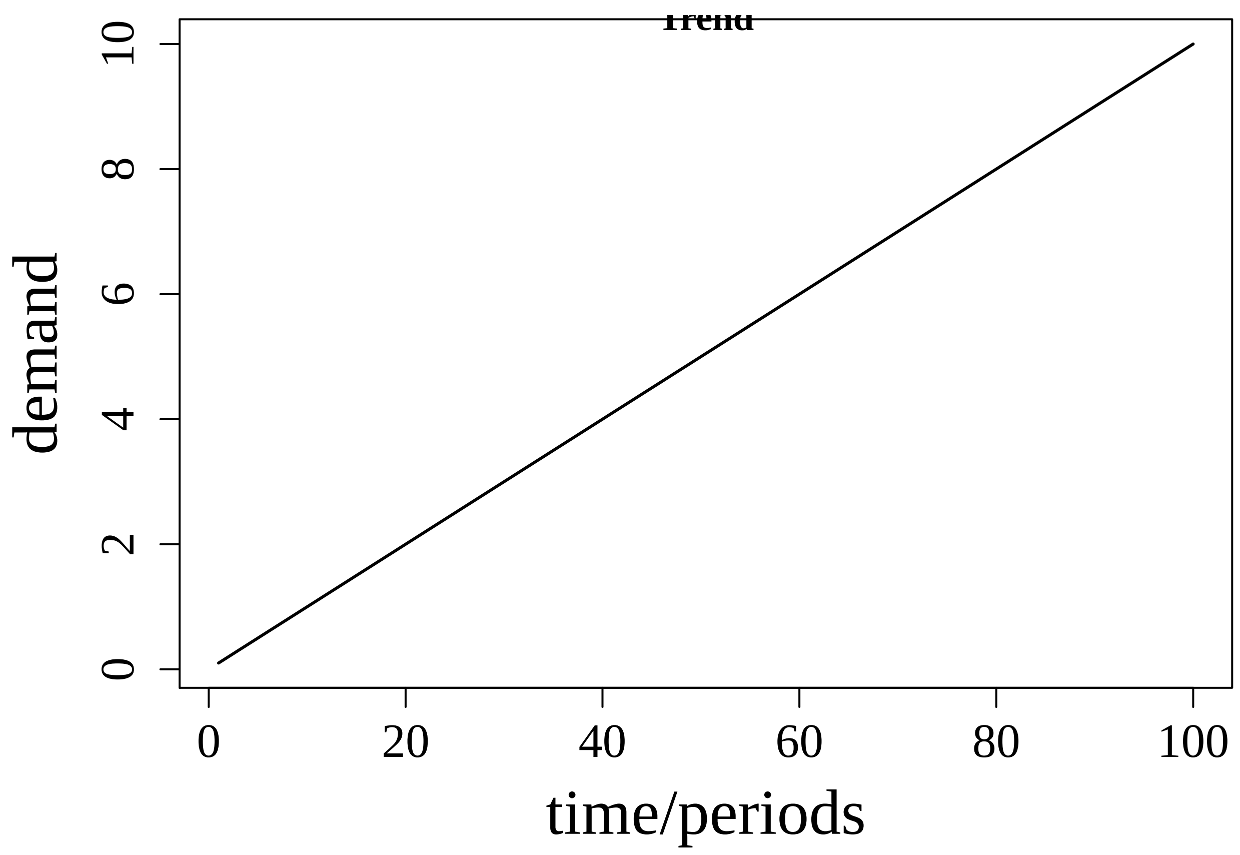
```
len <- 100

y.const <- rep(50, len) # replicates value 50 100 times



x <- 1:len #1 to 100

#png(file = "forecast_ts_const.png", bg = "transparent", width=600, height = 400)
#win.metafile("forecast_ts_const.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x , y.const , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis
```



```
#dev.off()
```

TREND

```
# TREND

y.trend <- 1:len*0.1 #between 0.1 to 10

#png(file = "forecast_ts_trend.png", bg = "transparent", width=600, height = 400)
#win.metafile("forecast_ts_trend.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x , y.trend , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis
```

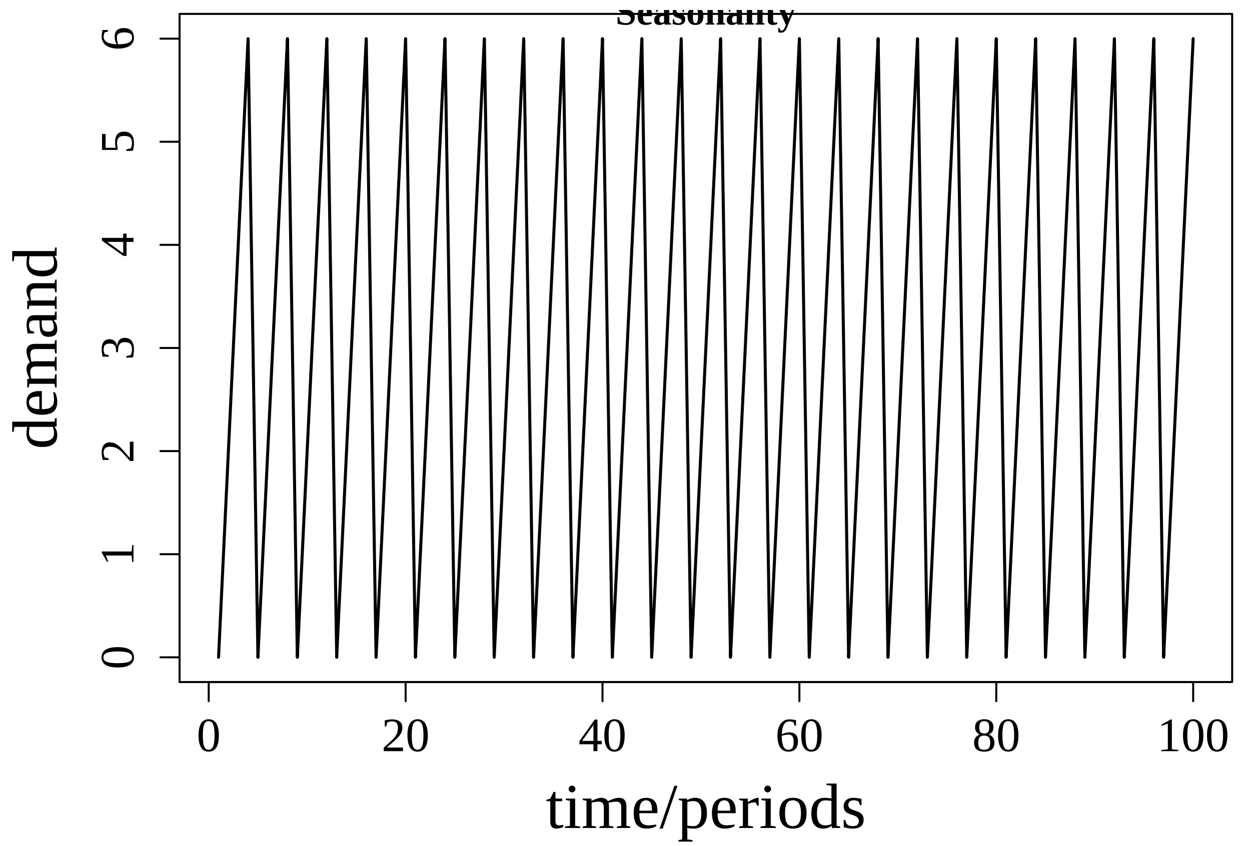


```
#dev.off()
```

Seasonality

```
#SEASONALITY

x.seas <- 0:(len-1)%%4 + 1 # creates seasonal values. 1,2,3,4 multiple times.

#drops as low as 0 and peaks as high as 6, {0,2,4,6} repeated.
y.seas <- (x.seas-1)*2 # quarterly fluctuations
```

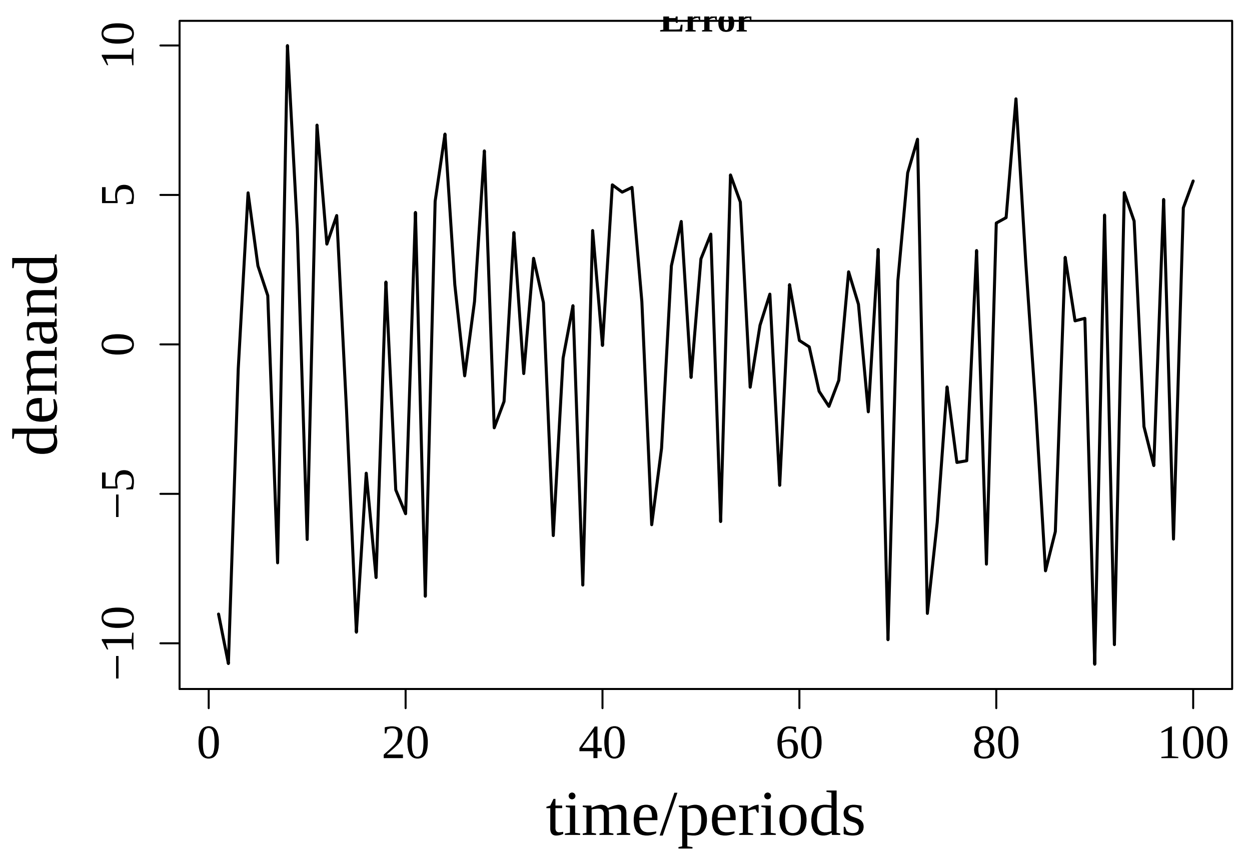
```
#png(file = "forecast_ts_seas.png", bg = "transparent", width=600, height = 400)
# win.metafile("forecast_ts_seas.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x , y.seas , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis =
```



```
#dev.off()
```

Error

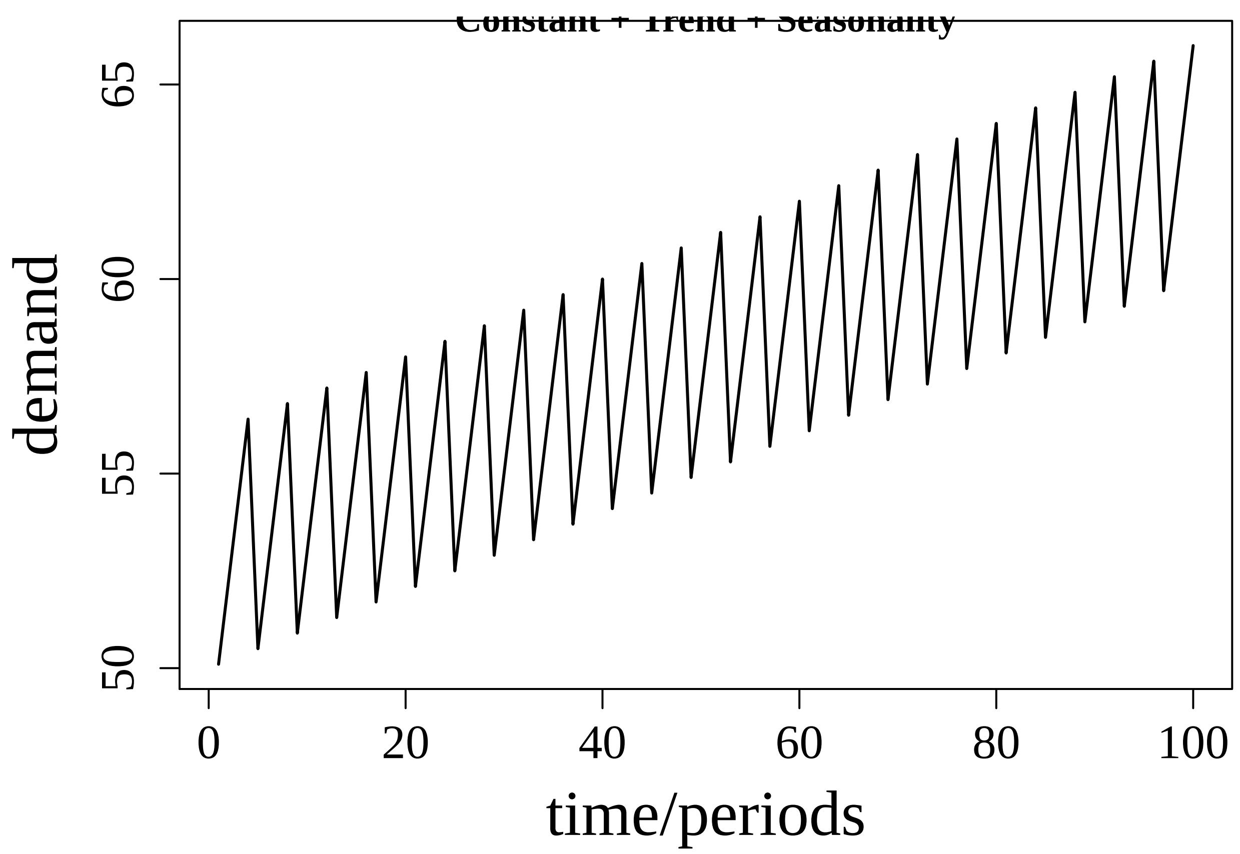
```
#ERROR

y.err <- rnorm(len, 0 , 5)

# win.metafile("forecast_ts_err.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x , y.err , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis =
```



```
#dev.off()
```

Constant + Trend + Seasonality

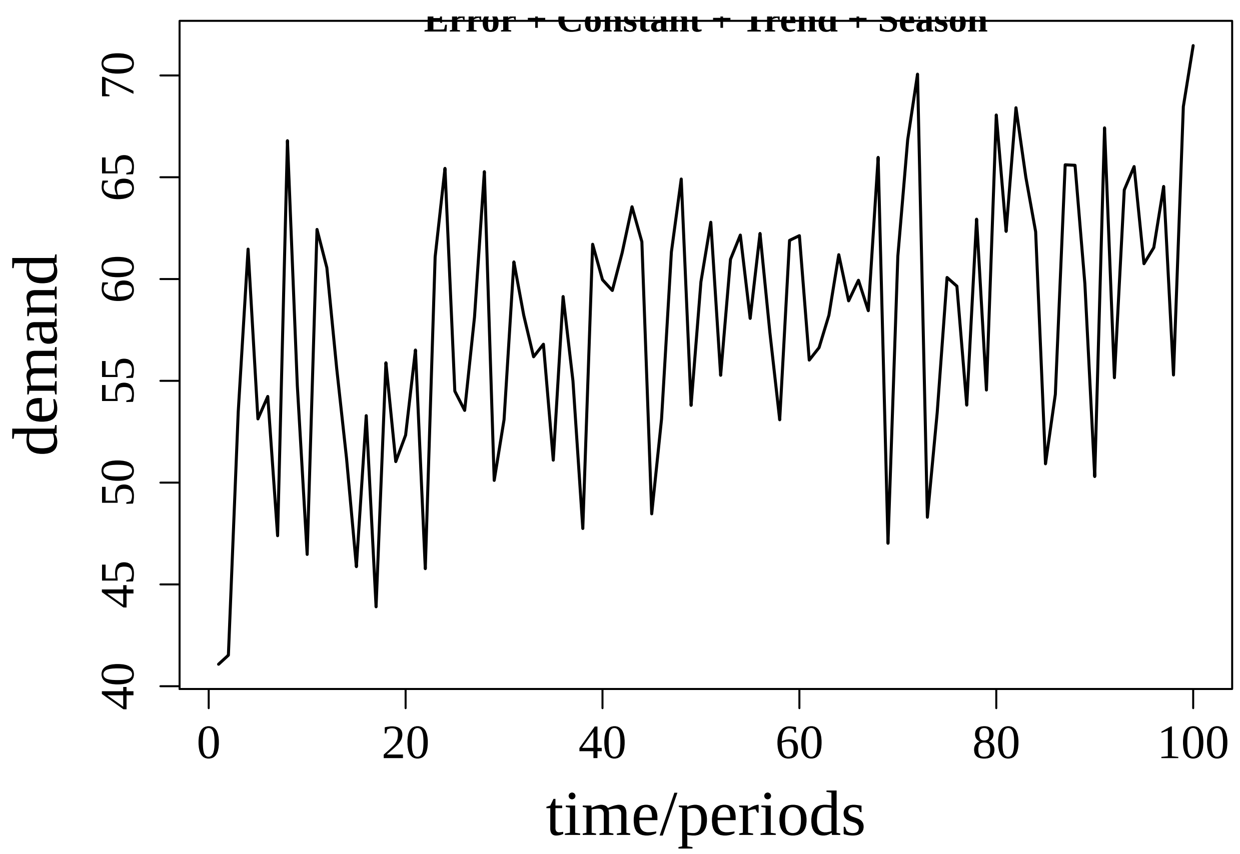
```
#png(file = "forecast_ts_all.png", bg = "transparent", width=600, height = 400)
# win.metafile("forecast_ts_all.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x , y.const + y.trend + y.seas , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex
```



```r
#dev.off()
```

Error + Constant + Trend + Season

```r
# win.metafile("forecast_ts_all_err.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x ,y.err +  y.const + y.trend + y.seas , type = "l", xlab= "time/periods", ylab ="demand", lwd = 1
```



```
#dev.off()
```

The better the forecast, the smaller is the forecasting error ϵ_{it} for material i in a particular time period t. The forecast error determines the safety stock level. ϵ_{it} follows a stochastic process.

Time series values

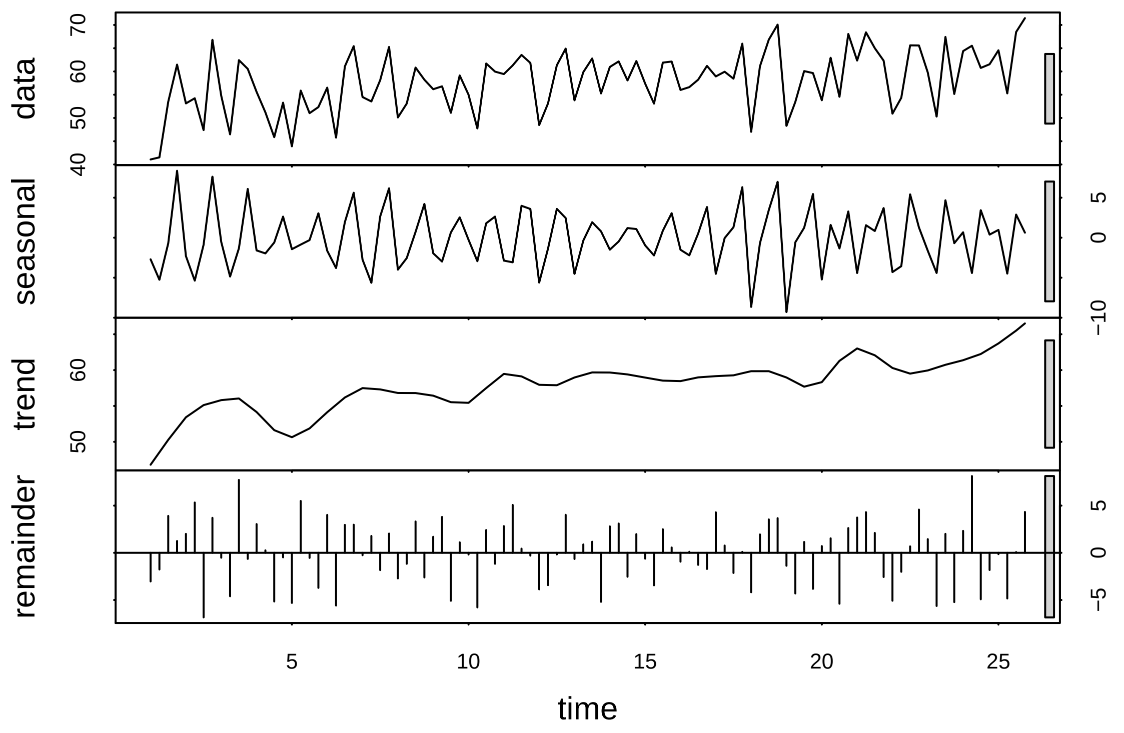
```
# ts values
y.com <- y.err +  y.const + y.trend + y.seas
#y.com # 100 values

# deltat is the fraction of the sampling period between successive observations; e.g., 1/12 for monthly
# 1/4 for quarterly data
y.com.ts <- ts(y.com, deltat = 1/4) # deltat divides the 100 values into 4 parts.

y.com.ts
```

```
##          Qtr1     Qtr2     Qtr3     Qtr4
## 1  41.07870 41.52250 53.48038 61.47107
## 2  53.12897 54.23298 47.39164 66.79688
## 3  54.76466 46.47228 62.43661 60.55508
## 4  55.61078 51.19392 45.87166 53.29079
## 5  43.89968 55.88601 51.03258 52.33453
## 6  56.51442 45.77370 61.08879 65.43711
## 7  54.49331 53.54765 58.13461 65.27377
## 8  50.10905 53.08995 60.84112 58.22309
## 9  56.17976 56.79900 51.10337 59.14135
## 10 54.99535 47.74771 61.70944 59.96436
## 11 59.43793 61.29333 63.55247 61.83070
## 12 48.46623 53.12446 61.32752 64.91346
## 13 53.79418 59.86021 62.78989 55.27280
## 14 60.96869 62.16132 58.06684 62.23980
## 15 57.38166 53.08558 61.89881 62.13047
## 16 56.01710 56.63138 58.22594 61.19625
## 17 58.92870 59.94121 58.44445 65.97665
## 18 47.01992 61.11997 66.83825 70.06445
## 19 48.29897 53.44721 60.07681 59.64961
## 20 53.80663 62.94098 54.54828 68.05963
## 21 62.34036 68.41522 65.00977 62.32254
## 22 50.92384 54.33806 65.61265 65.58734
## 23 59.77175 50.29896 67.42626 55.15428
## 24 64.37850 65.52567 60.75222 61.54490
## 25 64.54914 55.28668 68.46943 71.46731
```

TS decomosition by stl

seasonal, trend and irregular components using loess.

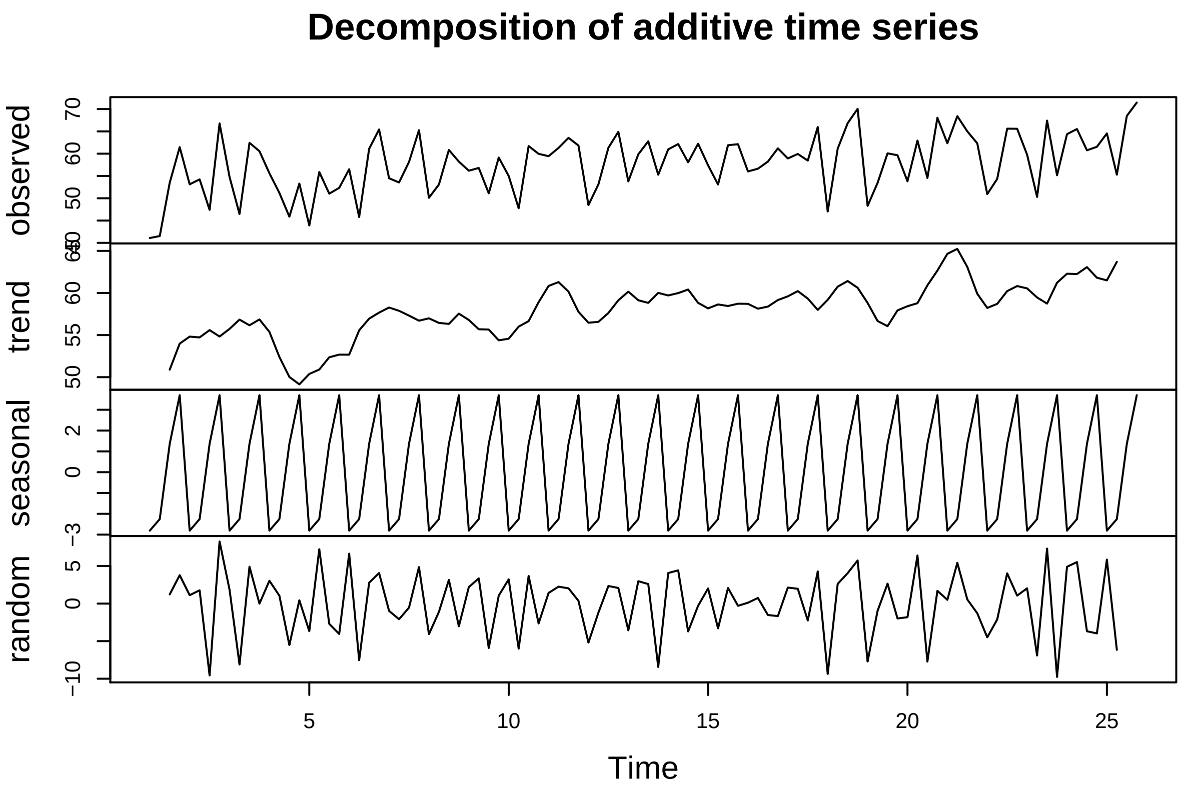
```r
stl.y <- stl(y.com.ts, s.window = 4)
plot(stl.y)
```



TS decomposition by decompose

decomposition of additive time series.

```r
decompose.y <- decompose(y.com.ts)
plot(decompose.y)
```



TS decomosition by lm

lm is linear models, It can be used to carry out regression, single stratum analysis of variance and analysis of covariance.

x.seas <- 0:(len-1)%%4 + 1 # creates seasonal values. 1,2,3,4 multiple times.

y.seas <- (x.seas-1)*2 # quarterly fluctuations

Regression Approach

- y_t : demand in period $t = \{1, ..., T\}$.
- $o \in \mathbb{N}$ the periodicity, the season of a period t is defined as $s_t \in \{1, ..., o\}$.
- $x^t = (1, ..., T)$ denotes the time vector.
- $x^s = (s_1, ..., s_T)$ the seasonality vector.

$$y_t = \beta_0 + \beta_{st} + \beta_1 \cdot t + \epsilon_t$$

- o represents seasonality.
- if $o = 12$ means seasons in months.
- $o = 4$ seasons Quarterly.

- $o = 30$ seasons for days in a month.
- $o = 7$ seasons for days in a week.

All β parameters are to be estimated.

- β_{st} additive seasonal effect.
- β_1 slope parameter representing linear trend increasing or decreasing over time.

We usually run test whether one of these $\beta_0, \beta_{st}, \beta_1$ is significantly different from 0.

- We can use T-test or do complete regression analysis, so the complete model is explaining something.
- f-test look at the individual model and try to reduce that.
- We can reduce that using stepwise regression technique, backward regression starting with the full model and reducing the parameter iteratively by backwards algorithm.

```
# TS decomosition by lm
#x.seas is seasonal ordering data
#x is the main data
#y.com is time series values y.err +  y.const + y.trend + y.seas

# as.factor meaning x.seas is ordered, as.factor(x.seas) indicates 4 levels
as.factor(x.seas)
```

```
##   [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1
##  [38] 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2
##  [75] 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
## Levels: 1 2 3 4
```

```
y.lm.decom <- lm(y.com ~ x + as.factor(x.seas) )
y.lm.decom
```

```
##
## Call:
## lm(formula = y.com ~ x + as.factor(x.seas))
##
## Coefficients:
##       (Intercept)                  x  as.factor(x.seas)2  as.factor(x.seas)3
##           49.7145             0.1012              0.5739              4.5684
## as.factor(x.seas)4
##            7.0179
```

```
summary(y.lm.decom)
```

```
##
## Call:
## lm(formula = y.com ~ x + as.factor(x.seas))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.8905  -3.8593   0.8466   3.9840   9.8267
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          49.71453    1.32182  37.611  < 2e-16 ***
## x                     0.10122    0.01746   5.796 8.82e-08 ***
## as.factor(x.seas)2  0.57389    1.42486   0.403  0.68803
## as.factor(x.seas)3  4.56842    1.42518   3.206  0.00184 **
## as.factor(x.seas)4  7.01793    1.42572   4.922 3.59e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.037 on 95 degrees of freedom
## Multiple R-squared:  0.4212, Adjusted R-squared:  0.3968
## F-statistic: 17.28 on 4 and 95 DF,  p-value: 1.103e-10
```

```
anova(y.lm.decom)
```

```
## Analysis of Variance Table
##
## Response: y.com
##                    Df  Sum Sq Mean Sq F value    Pr(>F)
## x                   1  918.27  918.27  36.190 3.328e-08 ***
## as.factor(x.seas)  3  835.91  278.64  10.981 2.954e-06 ***
## Residuals          95 2410.53   25.37
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Predicted and residual

```
y.lm.pred <- predict(y.lm.decom)
y.lm.res <- residuals(y.lm.decom)
```

Errors

MSE Mean Squared Error

$\epsilon_t^2 = (y_t - \hat{y}_t)^2$

$$MSE = \frac{1}{T}\sum_{t=1}^{T} \epsilon_t^2$$

```
# MSE
sum(y.lm.res^2)/100
```

```
## [1] 24.10531
```

MAD

Median absolute deviation (MAD).

$|\epsilon_t| = |y_t - \hat{y}_t|$

$$meadian(|\epsilon|)$$

```
median(abs(y.lm.res))
```

```
## [1] 4.035698
```

Absolute percentage error

$\frac{|\epsilon_t|}{y_t} = \frac{|y_t - \hat{y}_t|}{y_t}$

$$MAPE = \frac{1}{T}\sum_{t=1}^{T}\frac{|\epsilon_t|}{y_t}$$

```
# MAPE
mean(abs(y.lm.res)/y.lm.pred)
```

```
## [1] 0.07292474
```

Mean squared log accuracy ratio

$q_t = In(\frac{\hat{y}_t}{y_t})^2$

$$\frac{1}{T}\sum_{t=1}^{T}q_t$$

```
# log q
mean(log(y.lm.pred/y.com)^2)
```

```
## [1] 0.007785689
```

function for calculating accuracy measures

```
fc.stats <- function(yhat, y){
  y.res <- y -  yhat
  # MSE
  mse <- mean(y.res^2, na.rm =T)
  # MAD
  mad <- median(abs(y.res), na.rm =T)
  # MAPE
  mape <- mean(abs(y.res)/yhat, na.rm =T)
  # log q
  msla <- mean(log(yhat/y)^2, na.rm =T)
  #   return
  res <- c(mse,mad,mape,msla)
  names(res) <- c("MSE","MAD","MAPE","MSLA")
  return(res)
  }

# test functions
fc.stats(yhat = y.lm.pred, y = y.com)
```

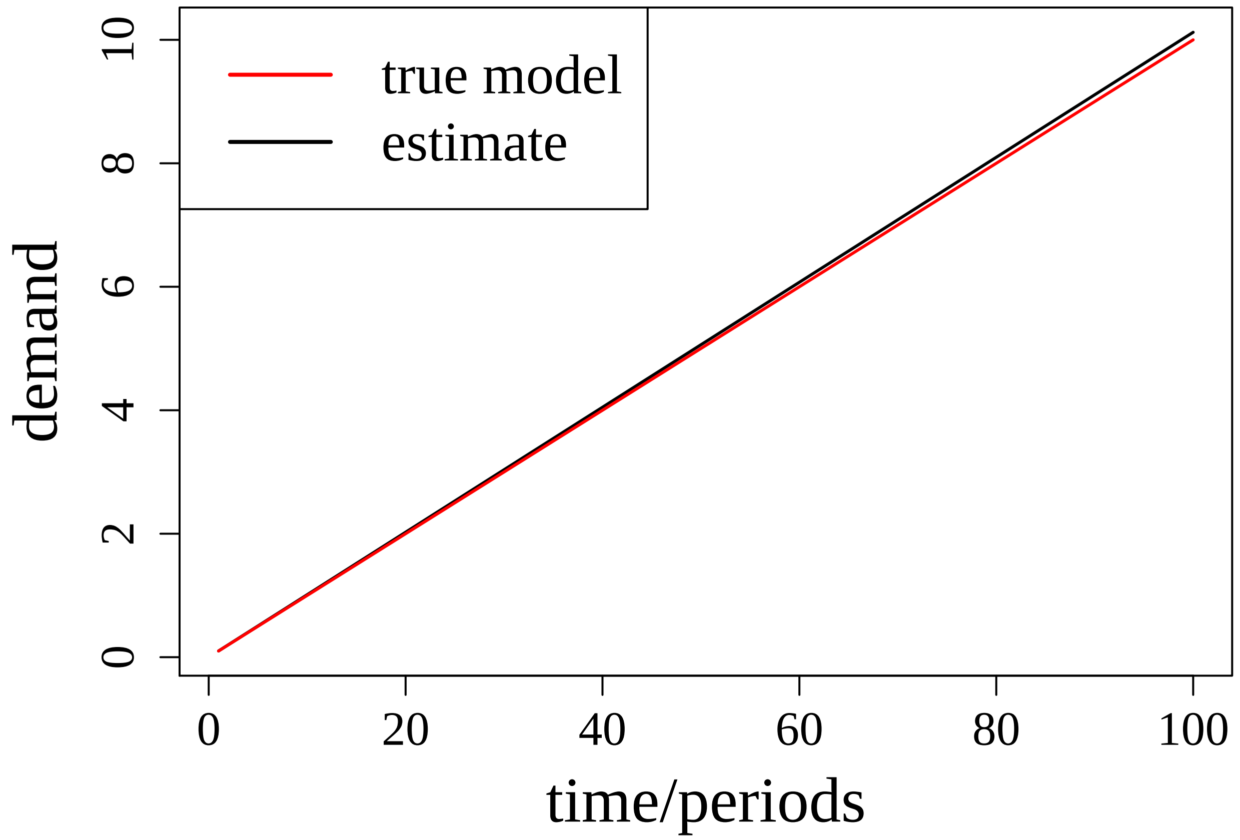
```
##           MSE          MAD         MAPE         MSLA
## 24.105306956  4.035698468  0.072924740  0.007785689
```

Regression analysis of examplary time series

- β_1 is trend estimate parameter
- $y_{t,trend}$ is the true demand value containing only the trend which is the $y.trend$
- $\hat{y}_t$ estimate demand value which is $y.lm.trend$
- x is the time/ period from 1 to 100.

$$\hat{y}_t = \beta_1 * \vec{x}$$

```
#
y.lm.trend <- coefficients(y.lm.decom)[2] * x


#win.metafile("forecast_ts_lm_trend.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.lm.trend, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis =
lines(x, y.trend, type="l", col="red", lwd = 1.5)
legend("topleft", col=c("red","black"), legend = c("true model","estimate"), lwd=c(2,2), cex=1.75, bg =
```



```
#dev.off()
```

β_{01} is 0.000000, β_{02} is 0.5738852 , β_{03} is 4.5684198 , β_{04} is 7.0179348.

$y_{t,seas} = \beta_{01}, \beta_{02}, \beta_{03}, \beta_{04}$ replicated lenght of $\frac{lenght|\vec{x}|}{4}$ from 1 to lenght $\vec{x}$

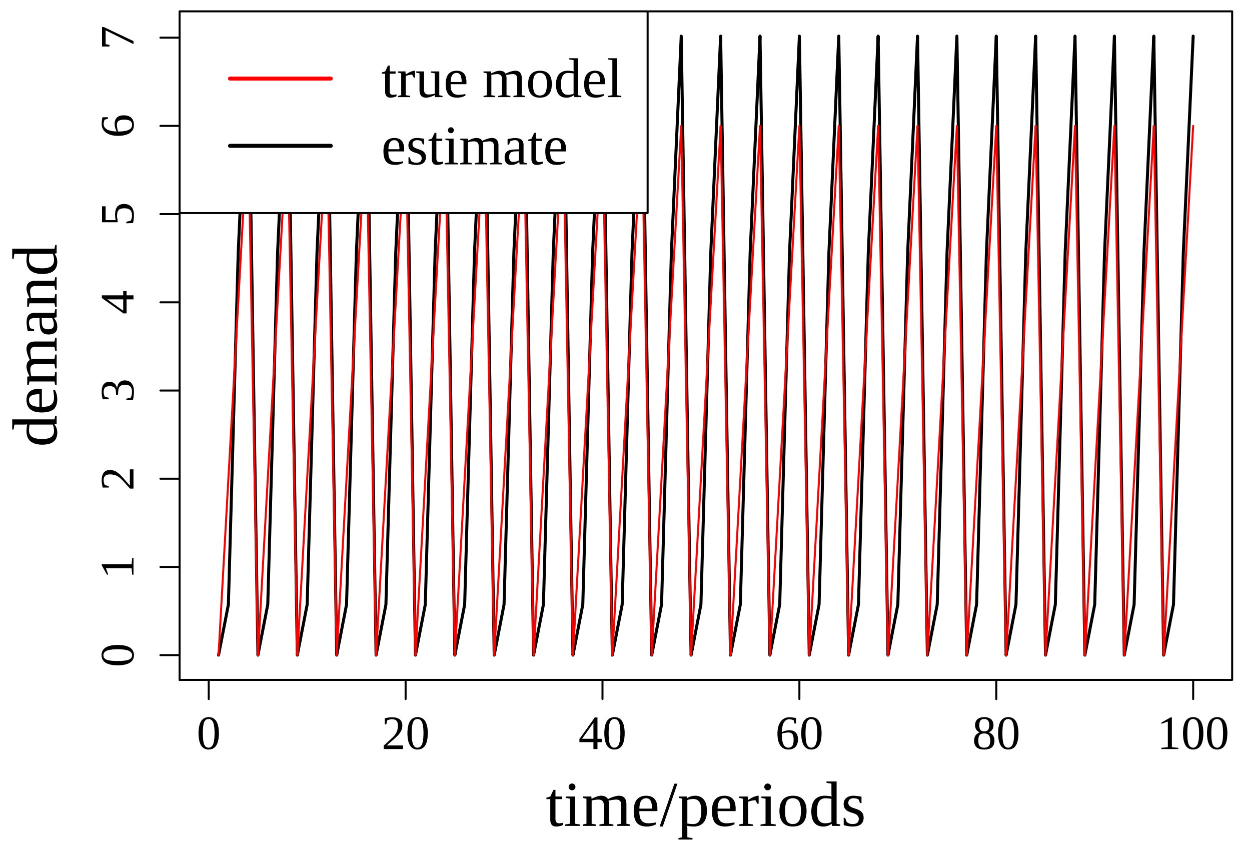
```
# tail(coefficients(y.lm.decom),3) means extract the last 3 values from the y.lm.decom vector

c(0,tail(coefficients(y.lm.decom),3))
```

```
##                    as.factor(x.seas)2 as.factor(x.seas)3 as.factor(x.seas)4
##          0.0000000          0.5738852          4.5684198          7.0179348
```

```
lm.seas.coeff <- c(0,tail(coefficients(y.lm.decom),3))
y.lm.seas <- rep(lm.seas.coeff, ceiling(len/4))[1:len]
#y.lm.seas
head(y.lm.seas)
```

```
##                    as.factor(x.seas)2 as.factor(x.seas)3 as.factor(x.seas)4
##          0.0000000          0.5738852          4.5684198          7.0179348
##                    as.factor(x.seas)2
##          0.0000000          0.5738852
```

```
#y.lm.seas <- rep(lm.seas.coeff, ceiling(len/4))[1:len]

#win.metafile("forecast_ts_lm_seas.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.lm.seas, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis = 
lines(x, y.seas, type="l", col="red")
legend("topleft", col=c("red","black"), legend = c("true model","estimate"), lwd=c(2,2), cex=1.75, bg = 
```

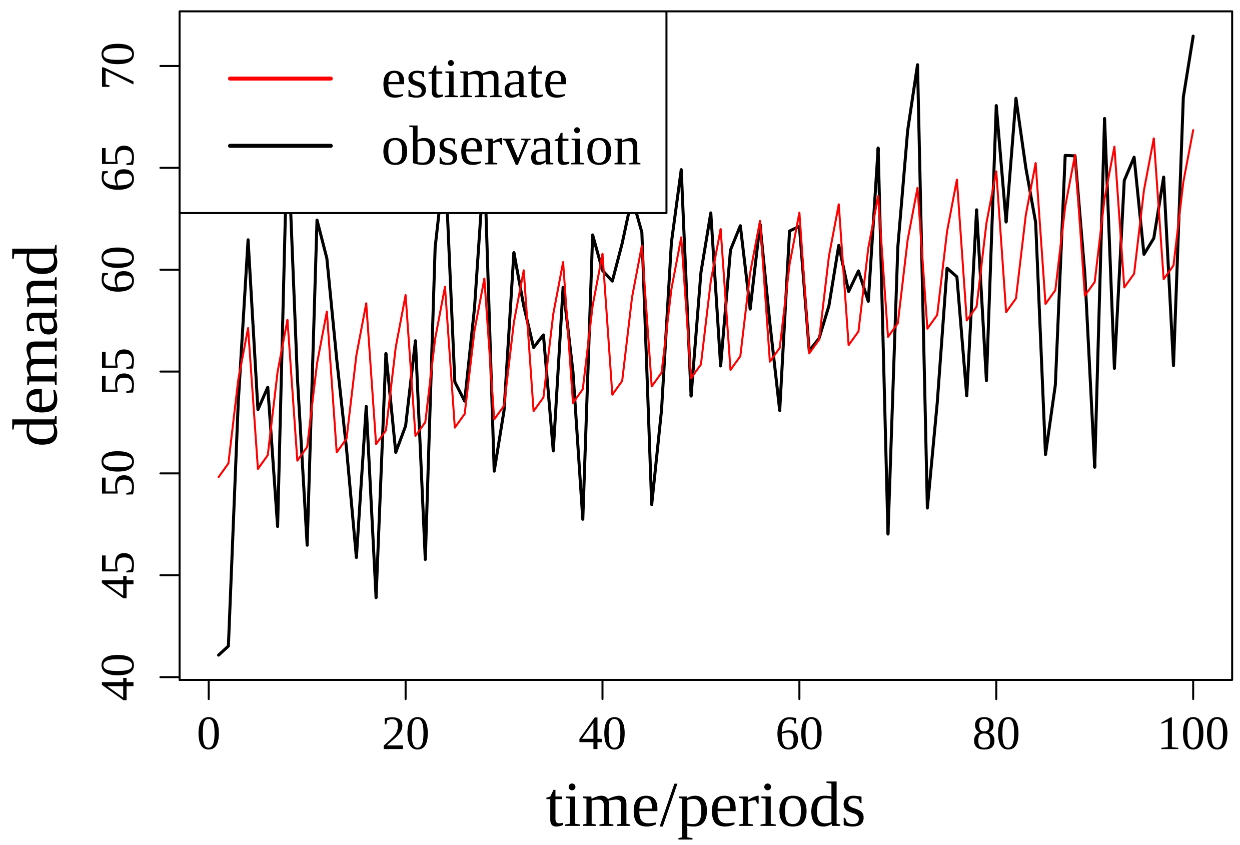


```
#dev.off()
```

estimated(predicted) vs actual

y.com= y.err + y.const + y.trend + y.seas

```
#win.metafile("forecast_ts_lm_pred.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.5 , cex.lab = 2, cex.axis = 1.5)
lines(x, y.lm.pred, type="l", col="red")
legend("topleft", col=c("red","black"), legend = c("estimate","observation"), lwd=c(2,2), cex=1.75, bg =
```

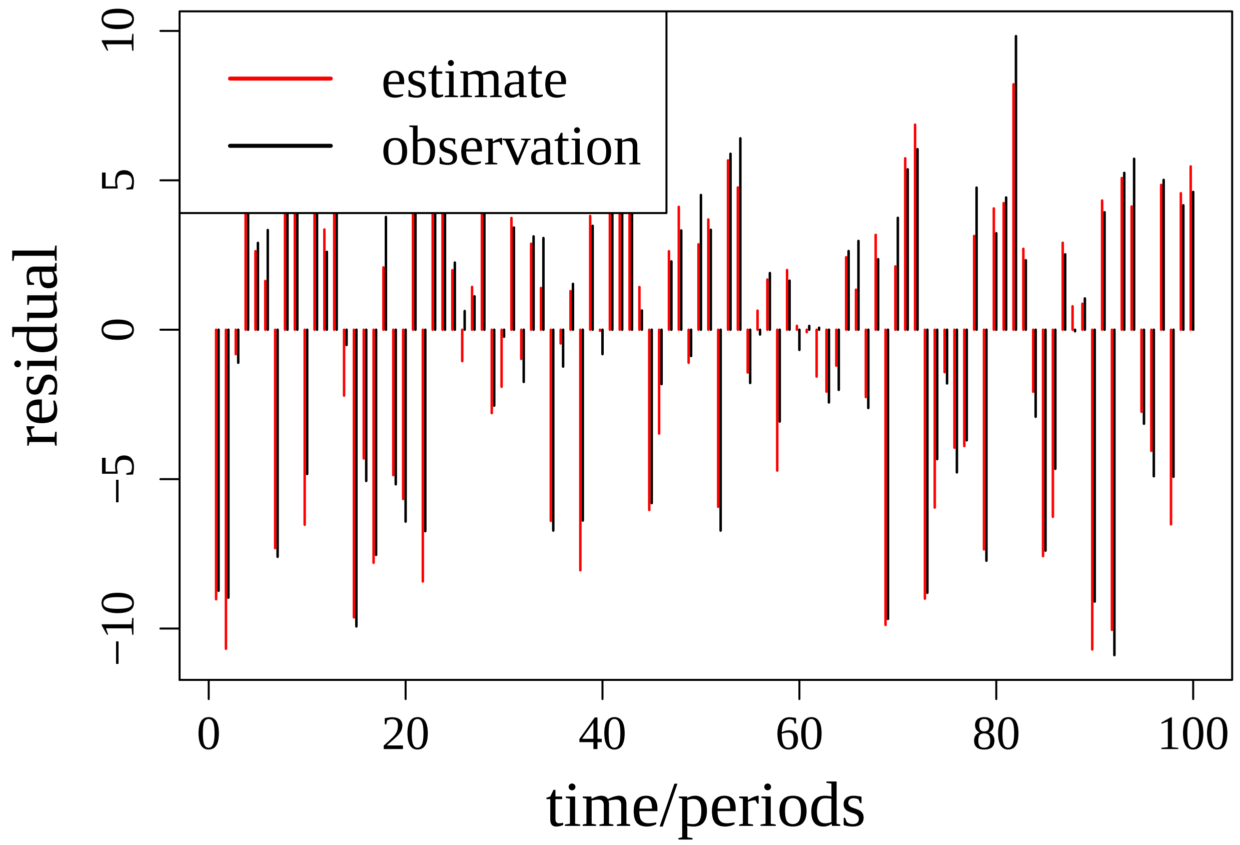


```
#dev.off()
```

Error

y.error = actual error, thus actual variance. residuals(y.lm.decom)= estimated error.

```
#win.metafile("forecast_ts_lm_err.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, residuals(y.lm.decom), type="h", xlab= "time/periods", ylab ="residual", lwd = 1.25 , cex.lab =
points(x-.25, y.err, type="h", col="red", lwd=1.25)
legend("topleft", col=c("red","black"), legend = c("estimate","observation"), lwd=c(2,2), cex=1.75, bg =
```



```
#dev.off()
```

The residuals flunctuate around zero, we also observe that the corresponding variants are equally distributed between + or - thus $E[\epsilon] \sim 0$.

moving average

$$\hat{y} = \sum_{\tau=t-n}^{t-1} \frac{1}{n} \cdot y_{t-\tau} = \frac{1}{n} \cdot \sum_{\tau=t-n}^{t-1} y_{t-\tau}$$

```
y.com.ts
```

```
##        Qtr1     Qtr2     Qtr3     Qtr4
## 1  41.07870 41.52250 53.48038 61.47107
## 2  53.12897 54.23298 47.39164 66.79688
## 3  54.76466 46.47228 62.43661 60.55508
## 4  55.61078 51.19392 45.87166 53.29079
## 5  43.89968 55.88601 51.03258 52.33453
## 6  56.51442 45.77370 61.08879 65.43711
## 7  54.49331 53.54765 58.13461 65.27377
## 8  50.10905 53.08995 60.84112 58.22309
## 9  56.17976 56.79900 51.10337 59.14135
## 10 54.99535 47.74771 61.70944 59.96436
## 11 59.43793 61.29333 63.55247 61.83070
## 12 48.46623 53.12446 61.32752 64.91346
## 13 53.79418 59.86021 62.78989 55.27280
## 14 60.96869 62.16132 58.06684 62.23980
## 15 57.38166 53.08558 61.89881 62.13047
## 16 56.01710 56.63138 58.22594 61.19625
## 17 58.92870 59.94121 58.44445 65.97665
## 18 47.01992 61.11997 66.83825 70.06445
## 19 48.29897 53.44721 60.07681 59.64961
## 20 53.80663 62.94098 54.54828 68.05963
## 21 62.34036 68.41522 65.00977 62.32254
## 22 50.92384 54.33806 65.61265 65.58734
## 23 59.77175 50.29896 67.42626 55.15428
## 24 64.37850 65.52567 60.75222 61.54490
## 25 64.54914 55.28668 68.46943 71.46731
```

SMA(x, n = 10, ...)

n= Number of periods to average over.

```
# SMA Calculate various moving averages (MA) of a series. SMA(x, n = 10, ...)
y.sma.4 <- SMA(y.com.ts, n =4)
y.sma.4
```

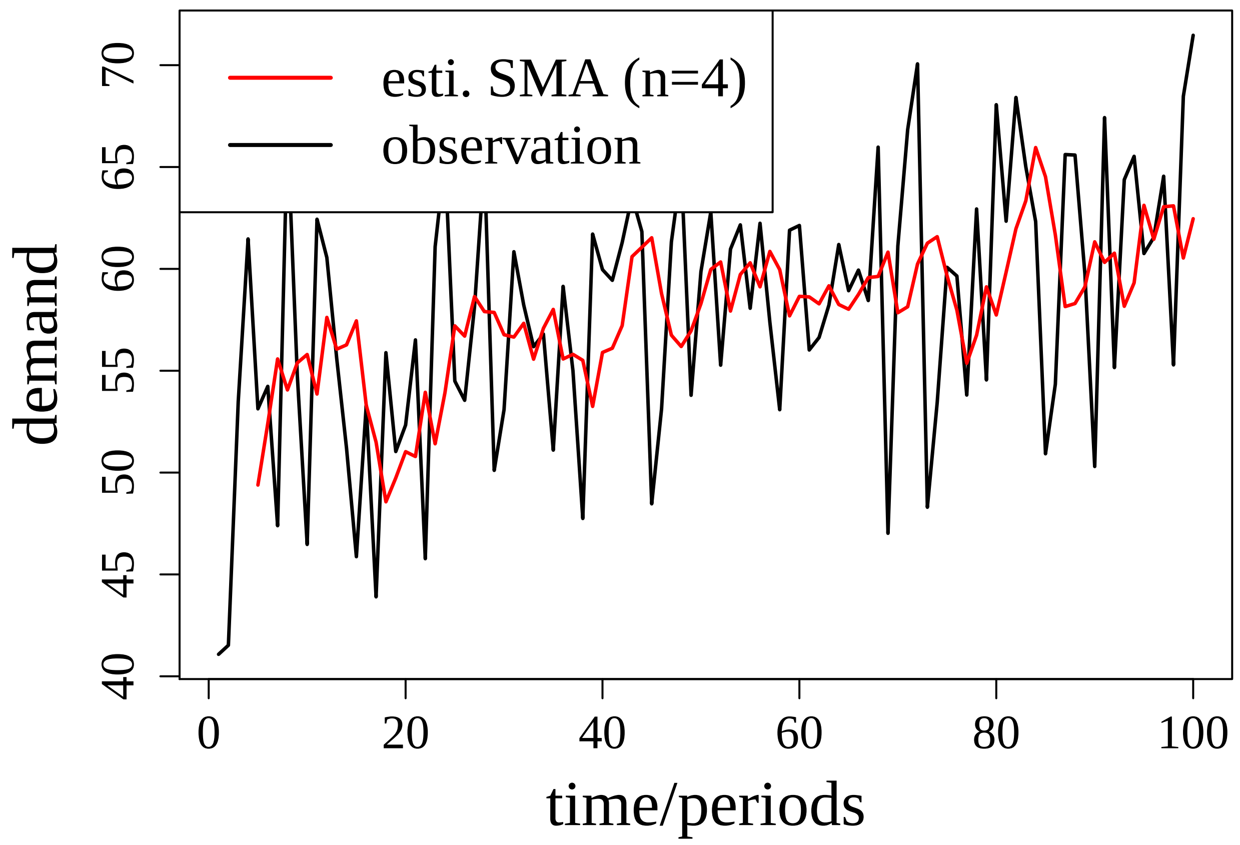
```
##        Qtr1     Qtr2     Qtr3     Qtr4
## 1        NA       NA       NA 49.38816
## 2  52.40073 55.57835 54.05617 55.38762
## 3  55.79654 53.85636 57.61760 56.05716
## 4  56.26869 57.44910 53.30786 51.49179
## 5  48.56401 49.73703 51.02726 50.78820
## 6  53.94189 51.41381 53.92786 57.20351
## 7  56.69823 58.64172 57.90317 57.86234
## 8  56.76627 56.65185 57.32848 55.56580
## 9  57.08348 58.01074 55.57630 55.80587
## 10 55.50977 53.24694 55.89846 56.10421
## 11 57.21486 60.60126 61.06202 61.52861
## 12 58.78568 56.74347 56.18723 56.95792
## 13 58.28991 59.97384 60.33943 57.92927
## 14 59.72290 60.29817 59.11741 60.85916
## 15 59.96241 57.69347 58.65146 58.62413
## 16 58.28299 59.16944 58.25122 58.01767
## 17 58.74557 59.57302 59.62765 60.82275
## 18 57.84556 58.14025 60.23870 61.26065
## 19 61.58041 59.66222 57.97186 55.36815
## 20 56.74506 59.11851 57.73638 59.83888
## 21 61.97231 63.34087 65.95624 64.52197
## 22 61.66784 58.14855 58.29927 59.11547
## 23 61.32745 60.31768 60.77108 58.16281
## 24 59.31450 63.12118 61.45267 63.05032
## 25 63.09298 60.53323 62.46254 64.94314
```

calculate for the $\widehat{y}_4 = \frac{y_1+y_2+y_3+y_4}{n}$, where n= 4.

```
sum(43.53065,46.03681,55.24236,56.08508)/4
```

```
## [1] 50.22373
```

```
#win.metafile("forecast_ts_sma_4.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis =
lines(x, c(NA, head(y.sma.4,99)), col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c("esti. SMA (n=4)","observation"), lwd=c(2,2), cex=1.
```



```
#dev.off()
```

```
fc.stats(yhat = y.sma.4, y = y.com)
```

```
##          MSE          MAD         MAPE         MSLA
## 25.965515909  3.506343108  0.071077434  0.008030984
```

find best moving average parameter

```
sim.sma <- function(x.n , y=y.com.ts){
  # calculates SMA statistics for a vector of time windows x.n
  sapply(x.n, function(x){
    tmp <- SMA(y, n =x)
    fc.stats(yhat = c(NA, head(tmp,99)), y = as.numeric(y))
  })
}
```

```
# Results
x.n <- 2:20
res.sma <- sim.sma(x.n = x.n)
res.sma
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
## MSE  55.38031704 46.16091717 37.91183997 37.08802290 37.81763301 37.79566097
## MAD   5.70802159  4.67512414  4.07218364  4.52236954  4.31634344  3.98274055
## MAPE  0.10835153  0.09514925  0.08648536  0.08636743  0.08629916  0.08644296
## MSLA  0.01724091  0.01421831  0.01167344  0.01135988  0.01163414  0.01160050
##             [,7]        [,8]        [,9]       [,10]        [,11]        [,12]
## MSE  33.49573632 34.40930089 35.04600272 33.95328069 31.736711183 31.794656410
## MAD   4.08477469  4.32830992  4.13502146  4.19837858  4.131470803  4.201531581
## MAPE  0.08230411  0.08337619  0.08399596  0.08306705  0.080338853  0.079769331
## MSLA  0.01029633  0.01056720  0.01069716  0.01038647  0.009702937  0.009703775
##            [,13]        [,14]        [,15]       [,16]        [,17]
## MSE  33.29509104 32.415133790 31.323806659 30.45720818 31.378276836
## MAD   4.03807965  4.071978919  3.996352124  4.01343684  4.128924922
## MAPE  0.08144688  0.080574244  0.079804135  0.07848071  0.079733148
## MSLA  0.01012752  0.009751564  0.009409303  0.00903189  0.009302805
##             [,18]        [,19]
## MSE  31.463581464 31.186289317
## MAD   4.133041125  4.266683708
## MAPE  0.080103264  0.080367060
## MSLA  0.009318247  0.009218315
```

best n w.r.t. MSE

```
x.n[which.min(res.sma["MSE",])]
```

```
## [1] 17
```

optimal statistics

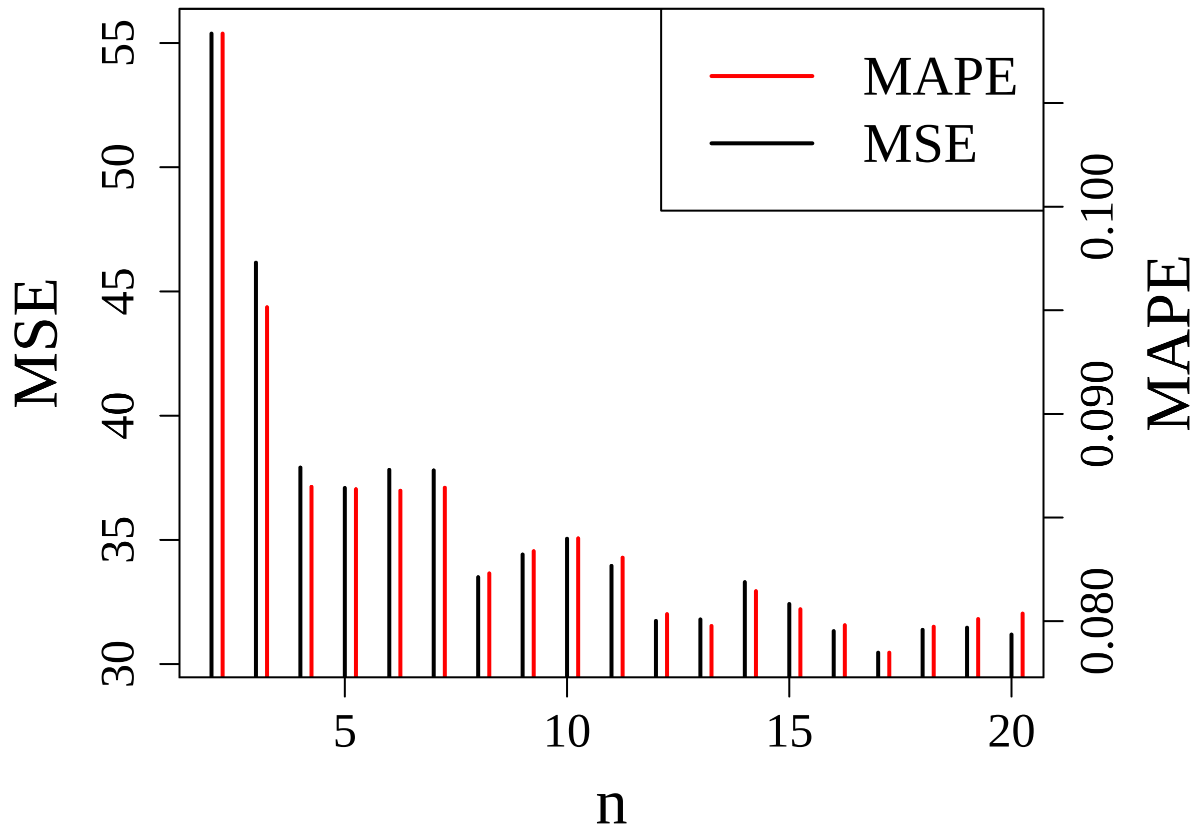
```
#gets the error values where MSE is Minimum

res.sma[,which.min(res.sma["MSE",])]
```

```
##         MSE         MAD        MAPE        MSLA
## 30.45720818  4.01343684  0.07848071  0.00903189
```

```
# Minimum MAPE
res.sma[,which.min(res.sma["MAPE",])]
```

```
##         MSE         MAD        MAPE        MSLA
## 30.45720818  4.01343684  0.07848071  0.00903189
```

```
#win.metafile("forecast_ts_sma_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,5))
plot(x.n, res.sma["MSE",], type="h", xlab= expression(n), ylab ="MSE", lwd = 2 , cex.lab = 2, cex.axis =
par(new = TRUE)
plot(x.n+.25, res.sma["MAPE",], type="h", col="red", lwd=2, xaxt = "n", yaxt = "n", xlab="", ylab="", xl
axis(side = 4, cex.axis = 1.5)
mtext("MAPE", side = 4, line = 3, cex = 2)
legend("topright", col=c("red","black"), legend = c("MAPE","MSE"), lwd=c(2,2), cex=1.75, bg ="white")
```



```
#dev.off()
```

Finds the best time window by minimizing MSE and MAPE with the same result, that is minimum for both measures is achieved for 17 periods for MA. We are estimating contant term in MA.

1st order exp. smoothing

Exponentially decaying weights

$$\hat{y}_t = (1-\alpha)\cdot\hat{y}_{t-1} + \alpha\cdot y_{t-1}$$

$$\hat{y}_t = \sum_{\tau=1}^{t-1} \alpha\cdot(1-\alpha)^{\tau-1}\cdot y_{t-\tau}$$

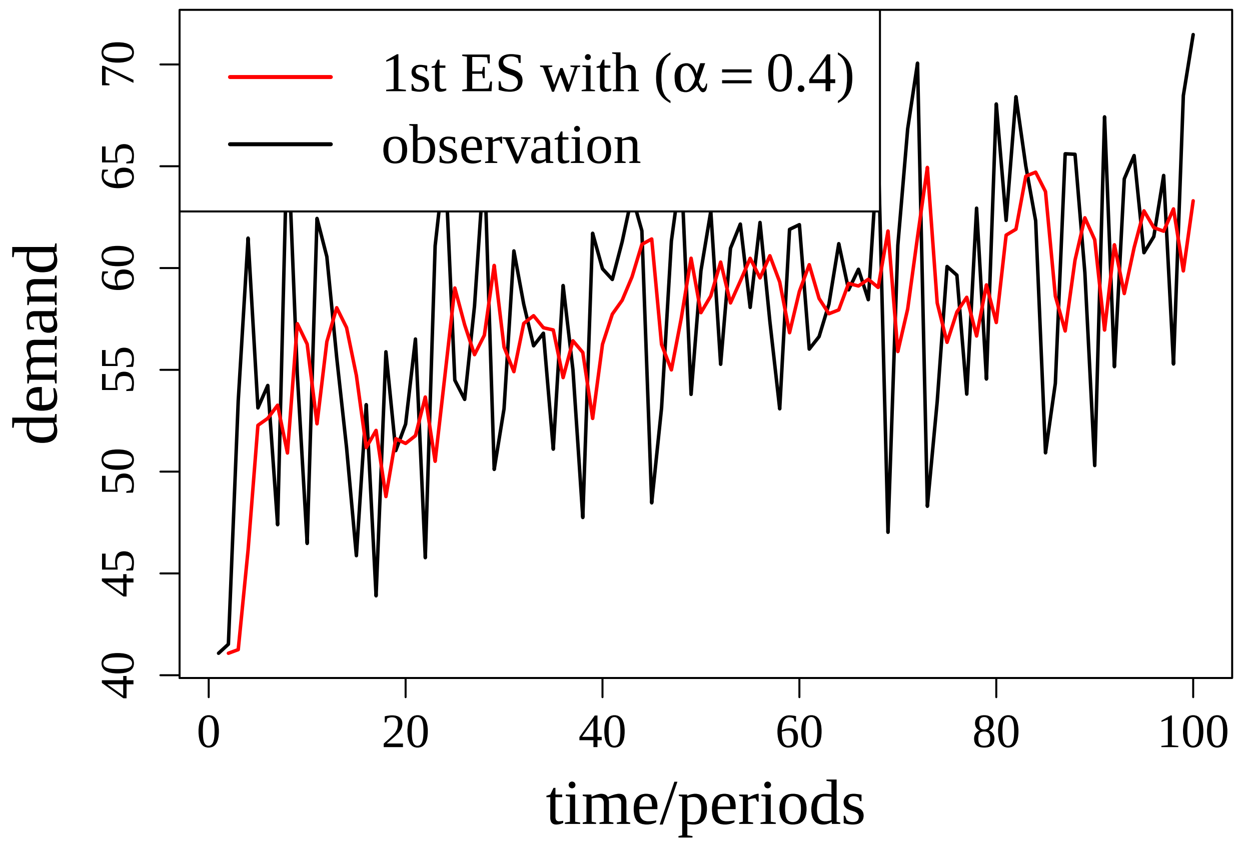
These vales $\alpha = 0.4$ is an arbitrary value, so no statistical method was applied in choosing it.

```
y.1st.exp.smoo <- HoltWinters(y.com.ts, alpha = .4, beta = F, gamma = F)

round(head(cbind(y.com, c(NA, y.1st.exp.smoo$fitted[,1])),10),2)
```

```
##        y.com
##  [1,] 41.08    NA
##  [2,] 41.52 41.08
##  [3,] 53.48 41.26
##  [4,] 61.47 46.15
##  [5,] 53.13 52.28
##  [6,] 54.23 52.62
##  [7,] 47.39 53.26
##  [8,] 66.80 50.91
##  [9,] 54.76 57.27
## [10,] 46.47 56.27
```

Only the first 10 values was displayed.

```
#win.metafile("forecast_ts_1stem_04.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis = 
lines(x, c(NA, y.1st.exp.smoo$fitted[,1]), col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("1st ES with (", alpha==0.4,")")),"
```



```r
#dev.off()
```

```r
fc.stats(yhat = c(NA, y.1st.exp.smoo$fitted[,1]), y=y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 43.45742282  4.52586324  0.09430960  0.01367058
```

find best alpha

```r
sim.1st.es <- function(x.alpha , y=y.com.ts){
  # calculates 1stES statistics for a vector of alphas x.alpha
  sapply(x.alpha, function(x){
    tmp <- HoltWinters(y, alpha = x, beta = F, gamma = F)
    fc.stats(yhat = c(NA, tmp$fitted[,1]), y = as.numeric(y))
  })
}
# results
x.alp <- seq(0.05,.95, length.out = 100)
res.sim.1stes <- sim.1st.es(x.alpha = x.alp)
res.sim.1stes
```

```
##             [,1]        [,2]        [,3]        [,4]        [,5]        [,6]
## MSE  55.13910241 51.11528287 48.38263329 46.44896424 45.03764738 43.98339049
## MAD   5.15182909  4.99948852  4.86303411  4.90351509  4.80832748  4.61129552
## MAPE  0.11613153  0.10961928  0.10497930  0.10165755  0.09918400  0.09724226
## MSLA  0.01812824  0.01672389  0.01576507  0.01508253  0.01458108  0.01420370
##             [,7]        [,8]        [,9]       [,10]       [,11]       [,12]
## MSE  43.18230797 42.56631465 42.08920055 41.71866566 41.43156107 41.21093636
## MAD   4.55157643  4.55850114  4.57320361  4.58657716  4.49517339  4.58709068
## MAPE  0.09592232  0.09493229  0.09408360  0.09334809  0.09291567  0.09260246
## MSLA  0.01391444  0.01368970  0.01351339  0.01337428  0.01326429  0.01317750
##            [,13]       [,14]       [,15]       [,16]       [,17]       [,18]
## MSE  41.04414678 40.92160447 40.83593373 40.78138672 40.75343133 40.74845532
## MAD   4.60050745  4.55872229  4.52202513  4.48961440  4.46078466  4.43497081
## MAPE  0.09233487  0.09210572  0.09190929  0.09177760  0.09167649  0.09159541
## MSLA  0.01310949  0.01305690  0.01301715  0.01298823  0.01296854  0.01295682
##            [,19]       [,20]       [,21]       [,22]       [,23]       [,24]
## MSE  40.76355067 40.79635407 40.84492760 40.90766844 40.98323975 41.07051752
## MAD   4.41171293  4.53763140  4.63110510  4.64878912  4.52735180  4.32242782
## MAPE  0.09154405  0.09156704  0.09161437  0.09166743  0.09172803  0.09181046
## MSLA  0.01295205  0.01295339  0.01296017  0.01297182  0.01298788  0.01300795
##            [,25]       [,26]       [,27]       [,28]       [,29]       [,30]
## MSE  41.16854911 41.27652091 41.39373279 41.51957786 41.65352636 41.79511277
## MAD   4.30834607  4.30744278  4.32156152  4.38642340  4.55107141  4.56971416
## MAPE  0.09189817  0.09203159  0.09220470  0.09237717  0.09254897  0.09272008
## MSLA  0.01303169  0.01305883  0.01308911  0.01312233  0.01315829  0.01319685
##            [,31]       [,32]       [,33]       [,34]       [,35]       [,36]
## MSE  41.94392536 42.09959785 42.26180257 42.43024487 42.60465861 42.78480238
## MAD   4.52603517  4.58914517  4.61739198  4.69228348  4.66381295  4.63450881
## MAPE  0.09289050  0.09306021  0.09322921  0.09339749  0.09356505  0.09373187
## MSLA  0.01323785  0.01328118  0.01332672  0.01337437  0.01342404  0.01347565
##            [,37]       [,38]       [,39]       [,40]       [,41]       [,42]
## MSE  42.97045640 43.16141997 43.35750928 43.55855564 43.76440397 43.97491147
## MAD   4.54775712  4.57353498  4.54193085  4.59536735  4.67501292  4.66109305
## MAPE  0.09389793  0.09406321  0.09422768  0.09439131  0.09455406  0.09471589
## MSLA  0.01352913  0.01358441  0.01364143  0.01370014  0.01376048  0.01382241
##            [,43]       [,44]       [,45]       [,46]       [,47]       [,48]
## MSE  44.18994654 44.40938785 44.63312351 44.86105035 45.09307336 45.32910508
## MAD   4.71602725  4.70882953  4.69782547  4.68632642  4.67441727  4.66217822
## MAPE  0.09487673  0.09505903  0.09529220  0.09554476  0.09579811  0.09605033
## MSLA  0.01388589  0.01395089  0.01401735  0.01408526  0.01415458  0.01422529
##            [,49]       [,50]       [,51]       [,52]       [,53]       [,54]
## MSE  45.56906519 45.81288004 46.06048231 46.31181067 46.56680948 46.82542854
## MAD   4.64968469  4.67753629  4.76598104  4.78953107  4.73975071  4.68749315
## MAPE  0.09632556  0.09659758  0.09687227  0.09715415  0.09743220  0.09770639
## MSLA  0.01429736  0.01437078  0.01444552  0.01452156  0.01459889  0.01467749
##            [,55]       [,56]       [,57]       [,58]       [,59]       [,60]
## MSE  47.08762283 47.35335234 47.62258183 47.89528070 48.17142282 48.45098642
## MAD   4.72298963  4.76241228  4.80443403  4.93078265  5.05112245  5.07503978
## MAPE  0.09797668  0.09824303  0.09850540  0.09877492  0.09904669  0.09931289
## MSLA  0.01475736  0.01483848  0.01492085  0.01500445  0.01508928  0.01517534
##            [,61]       [,62]       [,63]       [,64]       [,65]       [,66]
## MSE  48.73395395 49.02031200 49.31005118 49.60316609 49.89965522 50.19952094
## MAD   5.09946726  5.12434692  5.14961993  5.17522672  5.20110713  5.22720054
## MAPE  0.09957349  0.09982847  0.10007783  0.10032155  0.10055960  0.10079699
## MSLA  0.01526262  0.01535113  0.01544086  0.01553181  0.01562398  0.01571739
##            [,67]       [,68]       [,69]       [,70]       [,71]      [,72]
## MSE  50.50276941 50.80941060 51.11945825 51.43292989 51.74984679 52.0702340
## MAD   5.25344603  5.27978250  5.30614882  5.33248396  5.35872716  5.3848180
## MAPE  0.10104146  0.10128724  0.10156758  0.10184288  0.10211309  0.1023782
## MSLA  0.01581203  0.01590791  0.01600504  0.01610342  0.01620307  0.0163040
##            [,73]       [,74]      [,75]       [,76]       [,77]       [,78]
## MSE  52.39412054 52.72153901 53.0525261 53.38712234 53.72537233 54.06732471
## MAD   5.41069671  5.42912214  5.4615816  5.44531885  5.46135234  5.53486720
## MAPE  0.10264749  0.10291569  0.1031853  0.10349295  0.10379507  0.10410533
## MSLA  0.01640622  0.01650975  0.0166146  0.01672078  0.01682831  0.01693722
##            [,79]       [,80]      [,81]       [,82]       [,83]       [,84]
## MSE  54.41303226 54.76255199 55.1159453 55.47327786 55.83462010 56.20004696
## MAD   5.52819998  5.49450630  5.4598556  5.41828629  5.34227851  5.35044499
## MAPE  0.10446060  0.10481130  0.1051577  0.10553908  0.10591656  0.10629006
## MSLA  0.01704753  0.01715924  0.0172724  0.01738702  0.01750314  0.01762076
##            [,85]      [,86]       [,87]       [,88]       [,89]       [,90]
## MSE  56.56963823 56.9434786 57.32165789 57.70427106 58.09141852 58.48320621
## MAD   5.31225830  5.2732692  5.23351379  5.19303014  5.15185808  5.11003936
## MAPE  0.10665957  0.1070331  0.10740574  0.10778485  0.10816689  0.10854596
## MSLA  0.01773994  0.0178607  0.01798306  0.01810708  0.01823277  0.01836019
##            [,91]       [,92]       [,93]       [,94]       [,95]       [,96]
## MSE  58.87974582 59.28115496 59.68755735 60.09908304 60.51586863 60.93805746
## MAD   5.06761762  5.02463841  4.98114920  4.97719952  5.04472374  5.13794503
## MAPE  0.10893481  0.10936617  0.10981424  0.11025999  0.11071773  0.11117786
## MSLA  0.01848936  0.01862034  0.01875316  0.01888787  0.01902452  0.01916317
##            [,97]       [,98]       [,99]     [,100]
## MSE  61.36579989 61.79925352 62.23858343 62.6839625
## MAD   5.23213034  5.32723994  5.26304472  5.3280475
## MAPE  0.11163576  0.11209141  0.11255218  0.1130247
## MSLA  0.01930385  0.01944663  0.01959156  0.0197387
```

optimal alphas

For MSE

```
x.alp[which.min(res.sim.1stes["MSE",])]
```

```
## [1] 0.2045455
```

For MAPE

```
x.alp[which.min(res.sim.1stes["MAPE",])]
```

```
## [1] 0.2136364
```

optimal accuracy measures

For MSE

```
res.sim.1stes[,which.min(res.sim.1stes["MSE",])]
```

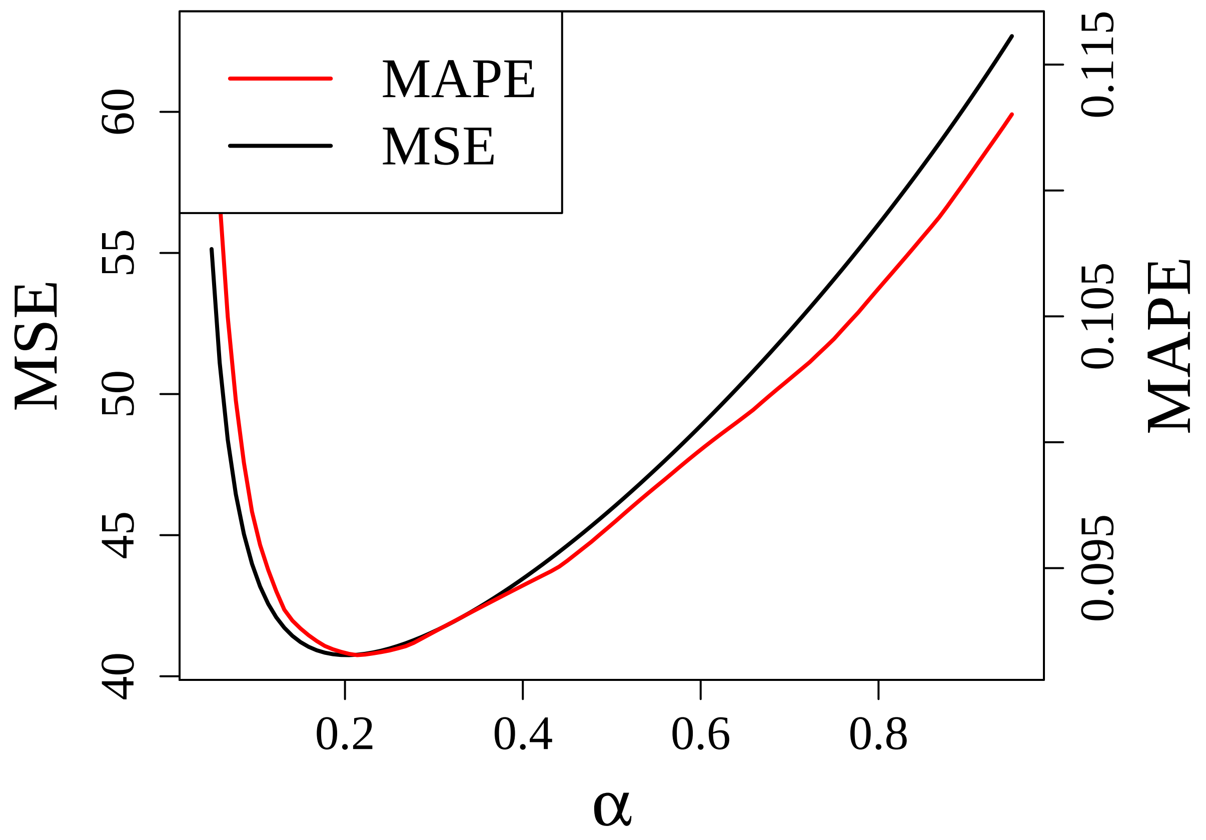
```
##         MSE         MAD        MAPE        MSLA
## 40.74845532  4.43497081  0.09159541  0.01295682
```

For MPE

```
res.sim.1stes[,which.min(res.sim.1stes["MAPE",])]
```

```
##         MSE         MAD        MAPE        MSLA
## 40.76355067  4.41171293  0.09154405  0.01295205
```

```
#win.metafile("forecast_ts_1stem_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,5))
plot(x.alp, res.sim.1stes["MSE",], type="l", xlab= expression(alpha), ylab ="MSE", lwd = 2 , cex.lab = 
par(new = TRUE)
plot(x.alp, res.sim.1stes["MAPE",], type="l", col="red", lwd=2, xaxt = "n", yaxt = "n", xlab="", ylab="
axis(side = 4, cex.axis = 1.5)
mtext("MAPE", side = 4, line = 3, cex = 2)
legend("topleft", col=c("red","black"), legend = c("MAPE","MSE"), lwd=c(2,2), cex=1.75, bg ="white")
```



```
#dev.off()
```

As seen in the diagram above, the best estimate for minimizing MSE $\alpha = 0.240909\ 1$, while that of MAPE $\alpha = 0.3863636$.

The forecasting accuracy is low, it does not account for linear trend and periodic patterns.

2nd order exponential smoothing

Used for time series with linear trends.

- a_t is constant estimated in period t.
- b_t the slope estimated in period t.

Forecast for period $t + k$ with $k \geq 0$ is:

$$\hat{y}_{t+k} = a_{t-1} + (k+1) \cdot b_{t-1}$$

Parameter update:

Exponential smoothing is applied with smoothing parameters α and β .

$$a_t = \alpha \cdot y_t(1-\alpha) \cdot (a_{t-1} + b_{t-1})$$

$$b_t = \beta \cdot (a_t - a_{t-1}) + (1 - \beta) \cdot b_{t-1}$$

$\alpha = 0.4$ and $\beta = 0.2$

function for calculating 2ndES forecasts

```r
sec.es <- function(y, alpha = .4, beta = .6, initial = c(mean(y), 0 )){

  n <- length(y)
  res <- matrix(NA, ncol=4, nrow=n+2)
  rownames(res) <- 0:(n+1)
  colnames(res) <- c("y","a","b","y.hat")

  res["0", c("a","b")] <- initial
  res[2:(n+1),"y"] <- y

  for(i in 2:(nrow(res)-1) ){
    res[i, "y.hat"] <- res[i-1, "a"] + res[i-1, "b"]
    res[i, "a"] <- alpha * res[i, "y"] + (1 - alpha) * res[i, "y.hat"]
    res[i, "b"] <- beta * (res[i, "a"]-res[i-1, "a"]) + (1 - beta) * res[i-1, "b"]
  }
  res[n+2, "y.hat"] <- res[n+1, "a"] + res[n+1, "b"]
  return(res)
}

y.2nd.exp.smoo <- sec.es(y.com, alpha = .4, beta = 0.2, initial = c(50,0))
y.2nd.exp.smoo
```

```
##            y        a            b    y.hat
## 0         NA 50.00000  0.000000000       NA
## 1   41.07870 46.43148 -0.713703971 50.00000
## 2   41.52250 44.03967 -1.049325980 45.71778
## 3   53.48038 47.18636 -0.210122719 42.99034
## 4   61.47107 52.77417  0.949463948 46.97623
## 5   53.12897 53.48577  0.901891238 53.72363
## 6   54.23298 54.32579  0.889517125 54.38766
## 7   47.39164 52.08584  0.263623923 55.21531
## 8   66.79688 58.12843  1.419416965 52.34946
## 9   54.76466 57.63457  1.036761987 59.54785
## 10  46.47228 53.79171  0.060837395 58.67133
## 11  62.43661 57.28617  0.747562396 53.85255
## 12  60.55508 59.04227  0.949270226 58.03373
## 13  55.61078 58.23924  0.598809196 59.99154
## 14  51.19392 55.78040 -0.012720622 58.83805
## 15  45.87166 51.80927 -0.804402155 55.76768
## 16  53.29079 51.91924 -0.621528626 51.00487
## 17  43.89968 48.33850 -1.213370811 51.29771
## 18  55.88601 50.62948 -0.512500042 47.12513
## 19  51.03258 50.48322 -0.439252088 50.11698
## 20  52.33453 50.96019 -0.256007013 50.04397
## 21  56.51442 53.02828  0.208812139 50.70419
## 22  45.77370 50.25173 -0.388259521 53.23709
## 23  61.08879 54.35360  0.509766039 49.86348
## 24  65.43711 59.09287  1.355665396 54.86337
## 25  54.49331 58.06644  0.879247948 60.44853
## 26  53.54765 56.78647  0.447404483 58.94569
## 27  58.13461 57.59417  0.519463167 57.23388
## 28  65.27377 60.97769  1.092274276 58.11364
## 29  50.10905 57.28560  0.135401092 62.06997
## 30  53.08995 55.68858 -0.211082728 57.42100
## 31  60.84112 57.62295  0.218007284 55.47750
## 32  58.22309 57.99381  0.248577742 57.84096
## 33  56.17976 57.41734  0.083567875 58.24239
## 34  56.79900 57.22014  0.027415802 57.50090
## 35  51.10337 54.78988 -0.464119811 57.24756
## 36  59.14135 56.25200 -0.078873121 54.32576
## 37  54.99535 55.70201 -0.173095118 56.17312
## 38  47.74771 52.41644 -0.795591488 55.52892
## 39  61.70944 55.65628  0.011495847 51.62084
## 40  59.96436 57.38641  0.355222570 55.66778
## 41  59.43793 58.42015  0.490926314 57.74163
## 42  61.29333 59.86398  0.681506233 58.91108
## 43  63.55247 61.74828  0.922065431 60.54548
## 44  61.83070 62.33449  0.854893688 62.67035
## 45  48.46623 57.30012 -0.322958026 63.18938
## 46  53.12446 55.43608 -0.631173984 56.97716
## 47  61.32752 57.41396 -0.109364797 54.80491
## 48  64.91346 60.34814  0.499344444 57.30459
## 49  53.79418 58.02616 -0.064919373 60.84748
## 50  59.86021 58.72083  0.086997579 57.96124
## 51  62.78989 60.40065  0.405562628 58.80783
## 52  55.27280 58.59285 -0.037110485 60.80621
## 53  60.96869 59.52092  0.155925502 58.55574
## 54  62.16132 60.67063  0.354683534 59.67684
## 55  58.06684 59.84193  0.118005803 61.02532
## 56  62.23980 60.87188  0.300395118 59.95993
## 57  57.38166 59.65603 -0.002854195 61.17228
## 58  53.08558 57.02614 -0.528261821 59.65317
## 59  61.89881 58.65825 -0.096187016 56.49787
## 60  62.13047 59.98943  0.189285928 58.56206
## 61  56.01710 58.51407 -0.143642973 60.17871
## 62  56.63138 57.67481 -0.282766340 58.37042
## 63  58.22594 57.72560 -0.216054572 57.39204
## 64  61.19625 58.98423  0.078881479 57.50955
## 65  58.92870 59.00935  0.068129069 59.06311
## 66  59.94121 59.42297  0.137227746 59.07747
## 67  58.44445 59.11390  0.047967776 59.56020
## 68  65.97665 61.88778  0.593150668 59.16186
## 69  47.01992 56.29653 -0.643730053 62.48093
## 70  61.11997 57.83966 -0.206356163 55.65280
## 71  66.83825 61.31529  0.530039139 57.63331
## 72  70.06445 65.13297  1.187568891 61.84532
## 73  48.29897 59.11191 -0.254156504 66.32054
## 74  53.44721 56.69354 -0.687000354 58.85776
## 75  60.07681 57.63465 -0.361378864 56.00654
## 76  59.64961 58.22380 -0.171271451 57.27327
## 77  53.80663 56.35417 -0.510943922 58.05253
## 78  62.94098 58.68233  0.056876290 55.84323
## 79  54.54828 57.06284 -0.278397246 58.73920
## 80  68.05963 61.29451  0.623617904 56.78444
## 81  62.34036 62.08702  0.657395705 61.91813
## 82  68.41522 65.01274  1.111059653 62.74442
## 83  65.00977 65.67819  1.021937438 66.12380
## 84  62.32254 64.94909  0.671730428 66.70012
## 85  50.92384 59.74203 -0.504028130 65.62082
## 86  54.33806 57.27802 -0.896023349 59.23800
## 87  65.61265 60.07426 -0.157570828 56.38200
## 88  65.58734 62.18495  0.296081240 59.91669
## 89  59.77175 61.39732  0.079338501 62.48103
## 90  50.29896 57.00558 -0.814877217 61.47666
## 91  67.42626 60.68493  0.083967767 56.19070
## 92  55.15428 58.52305 -0.365201515 60.76889
## 93  64.37850 60.64611  0.132451186 58.15785
## 94  65.52567 62.67740  0.512219971 60.77856
## 95  60.75222 62.21466  0.317227767 63.18962
## 96  61.54490 62.13709  0.238268386 62.53189
## 97  64.54914 63.24487  0.412170524 62.37536
## 98  55.28668 60.30890 -0.257458899 63.65704
## 99  68.46943 63.41863  0.415980595 60.05144
## 100 71.46731 66.88769  1.026596124 63.83462
## 101       NA       NA           NA 67.91429
```

```r
fc.stats(yhat = y.2nd.exp.smoo[2:101,"y.hat"] , y = y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 47.81697466  5.13178269  0.09948266  0.01508159
```

alternative from stats package (initial values cannot be controlled) y.2nd.exp.smoo <- HoltWinters(y.com.ts, alpha = T, beta = T, gamma = F, l.start = 50, b.start = 0, start.periods = 0)

alternative from forecast package

```r
y.2nd.exp.smoo <- holt(y.com.ts , h = 4 , initial = "simple")
y.2nd.exp.smoo
```

```
##       Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 26 Q1       64.55917 56.43266 72.68568 52.13074 76.98759
## 26 Q2       64.73595 56.42450 73.04740 52.02468 77.44722
## 26 Q3       64.91273 56.36853 73.45693 51.84551 77.97996
## 26 Q4       65.08952 56.26219 73.91685 51.58928 78.58975
```

```r
y.2nd.exp.smoo$model$fitted
```

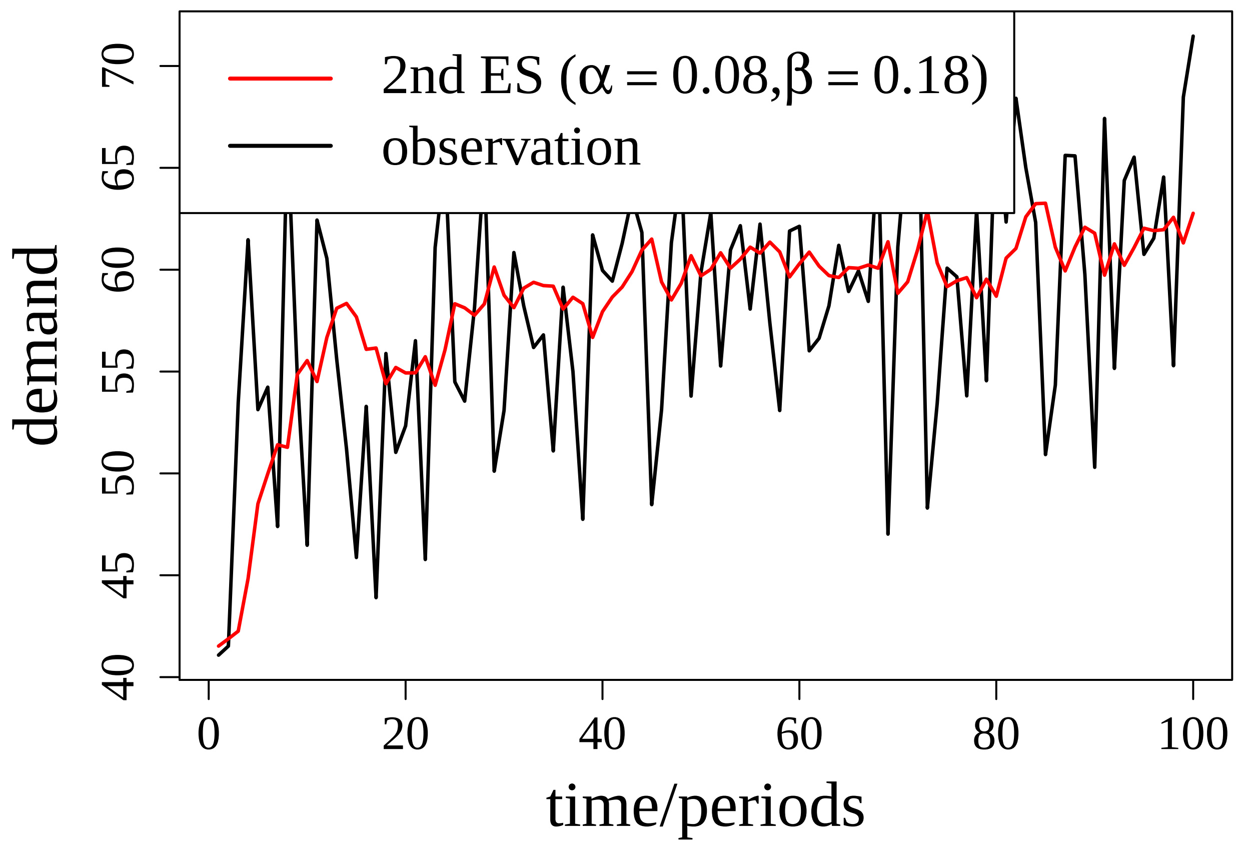
```
##        Qtr1     Qtr2     Qtr3     Qtr4
## 1  41.52250 41.88163 42.25451 44.83576
## 2  48.50980 49.98123 51.40755 51.27946
## 3  54.85644 55.53925 54.50917 56.67244
## 4  58.10687 58.34530 57.68204 56.09121
## 5  56.15552 54.40068 55.20125 54.93112
## 6  54.93838 55.72770 54.32572 56.05936
## 7  58.32844 58.12734 57.76345 58.31936
## 8  60.13330 58.74549 58.13696 59.09281
## 9  59.38142 59.22041 59.19095 58.06734
## 10 58.64794 58.33255 56.67480 57.93986
## 11 58.65784 59.14935 59.90529 60.95963
## 12 61.50403 59.39946 58.51466 59.32981
## 13 60.68880 59.69726 60.01494 60.83186
## 14 60.07373 60.51690 61.10790 60.81385
## 15 61.35562 60.87484 59.64465 60.28853
## 16 60.86594 60.17678 59.71009 59.61748
## 17 60.10122 60.06859 60.22901 60.07257
## 18 61.37340 58.84121 59.40466 60.96394
## 19 62.88145 60.32968 59.16814 59.45585
## 20 59.61208 58.62476 59.53717 58.69759
## 21 60.56905 61.04291 62.59496 63.24104
## 22 63.26422 61.10323 59.93921 61.11177
## 23 62.08648 61.78986 59.72999 61.26871
## 24 60.21411 61.08753 62.03571 61.91626
## 25 61.96389 62.57361 61.31379 62.77006
```

Errors

```
fc.stats(yhat = y.2nd.exp.smoo$model$fitted , y = y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 40.21017833  4.16653455  0.08686160  0.01284011
```

```
#png(file = "forecast_ts_2ndem_best.png", bg = "transparent", width=600, height = 400)
#win.metafile("forecast_ts_2ndem_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis = 
lines(x, y.2nd.exp.smoo$model$fitted, col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("2nd ES (", alpha==0.08,",",beta==0
```



```
#dev.off()
```

This result is worse than 1st order exponential smoothing, because it is not capturing all the elements in time series.

3rd order exponential smoothing

Here we add the seasonal effect.

Starting values are needed for these parameters a_t, b_t, c_t.

- a_t is constant estimated in period t.
- b_t is the slope estimated in period t.
- c_t the seasonal offset estimated in period t.

remember o = season which can be monthly weekly, quarterly e.t.c. The forecast for period $t + k$ with $0 \leq k < o$ is:

$$\hat{y}_{t+k} = a_{a-1} + (k+1) \cdot b_{t-1} + c_{t-o+k}$$

Parameter update

For Updating a_t, b_t and c_t, exponential smoothing is applied with smoothing parameters α, β and γ.

$$a_t = \alpha \cdot (y_t - c_{t-o}) + (1-\alpha) \cdot (a_{t-1} + b_{t-1})$$

$$b_t = \beta \cdot (a_t - a_{t-1}) + (1+\beta) \cdot b_{t-1}$$
$$c_t = \gamma \cdot (y_t - a_{t-1} - b_{t-1}) + (1-\gamma) \cdot c_{t-o}$$

## function for calculating 3rdES forecasts

```r
thi.es <- function(y, alpha = .4, beta = .2, gamma=.3, periods = 4, initial = list( a = mean(y), b= 0,
  n <- length(y)
  res <- matrix(NA, ncol=5, nrow=n+1+periods)
  rownames(res) <- (-periods+1):(n+1)
  colnames(res) <- c("y","a","b","c","y.hat")

  res["0", c("a","b")] <- c(initial$a, initial$b)
  res[1:periods, "c"] <- initial$c
  res[(periods+1):(n+periods),"y"] <- y

  for(i in (periods+1):(nrow(res)-1) ){
    res[i, "y.hat"] <- res[i-1, "a"] + res[i-1, "b"]+ res[i-periods, "c"]
    res[i, "a"] <- alpha * (res[i, "y"] - res[i-periods, "c"]) + (1 - alpha) * (res[i-1, "a"] + res[i-1
    res[i, "b"] <- beta * (res[i, "a"] - res[i-1, "a"]) + (1 - beta) * res[i-1, "b"]
    res[i, "c"] <- gamma * (res[i, "y"] - res[i-1, "a"] - res[i-1, "b"]) + (1 - gamma) *res[i-periods,
  }
  res[nrow(res), "y.hat"] <- res[nrow(res)-1, "a"] + res[nrow(res)-1, "b"] + res[nrow(res)-periods, "c"
  return(res)
}

thi.es(y.com.ts)
```

```
##            y        a            b           c    y.hat
## -3        NA       NA           NA  0.00000000       NA
## -2        NA       NA           NA  0.00000000       NA
## -1        NA       NA           NA  0.00000000       NA
## 0         NA 57.86622  0.000000000  0.00000000       NA
## 1   41.07870 51.15121 -1.343001756 -5.03625658 57.86622
## 2   41.52250 46.49393 -2.005858614 -2.48571322 49.80821
## 3   53.48038 48.08499 -1.286473651  2.69769361 44.48807
## 4   61.47107 52.66754 -0.112669888  4.40176411 46.79852
## 5   53.12897 54.79901  0.336158893 -3.35314865 47.51861
## 6   54.23298 55.76858  0.462840793 -2.01065609 52.64946
## 7   47.39164 51.61643 -0.460157241 -0.76354902 58.92912
## 8   66.79688 55.65181  0.438949755  7.77341535 55.55804
## 9   54.76466 56.90158  0.601113547 -2.74503443 52.73761
## 10  46.47228 53.89479 -0.120467313 -4.71658432 55.49204
## 11  62.43661 57.54466  0.633599739  2.06420243 53.01077
## 12  60.55508 56.01962  0.201872662  6.15443881 65.95167
## 13  55.61078 57.07522  0.372618475 -2.10473763 53.47646
## 14  51.19392 56.83291  0.249632013 -5.17778355 52.73126
## 15  45.87166 51.77251 -0.812374658 -1.91832259 59.14674
## 16  53.29079 49.43062 -1.118277322  5.00730382 57.11457
## 17  43.89968 47.38917 -1.302911211 -2.79711472 46.20760
## 18  55.88601 52.07727 -0.104708559 -0.68452361 40.90848
## 19  51.03258 52.36390 -0.026441672 -1.62482176 50.05424
## 20  52.33453 50.33337 -0.427260183  3.50423440 57.34476
## 21  56.51442 53.66828  0.325174554  0.02451554 47.10899
## 22  45.77370 50.97936 -0.277644037 -2.94509332 53.30893
## 23  61.08879 55.50648  0.683307979  1.97874830 49.07689
## 24  65.43711 58.48702  1.142755360  5.22716208 59.69402
## 25  54.49331 57.56539  0.729877039 -1.52377816 59.65429
## 26  53.54765 57.57425  0.585675431 -3.48584935 55.35017
## 27  58.13461 57.35830  0.425350225  1.37752877 60.13868
## 28  65.27377 58.68884  0.606386932  5.90604973 63.01082
## 29  50.10905 56.23027 -0.006604687 -3.82249673 57.77145
## 30  53.08995 56.36452  0.021566591 -3.38020706 52.73781
## 31  60.84112 57.61709  0.267767465  2.30078205 57.76361
## 32  58.22309 55.65773 -0.177658050  4.23570405 63.79091
## 33  56.17976 57.28895  0.184117079 -2.46584000 51.65757
## 34  56.79900 58.55552  0.400608894 -2.56836276 54.09286
## 35  51.10337 54.89471 -0.411674963 -0.74528241 61.25691
## 36  59.14135 54.65208 -0.377866567  4.36248553 58.71874
## 37  54.99535 55.54900 -0.122908541 -1.50974740 51.80837
## 38  47.74771 53.38209 -0.531710003 -4.10136824 52.85773
## 39  61.70944 56.69211  0.236637331  2.13602009 52.10509
## 40  59.96436 56.39800  0.130487299  3.96442292 61.29124
## 41  59.43793 58.29616  0.484022447 -0.18399060 55.01874
## 42  61.29333 61.42599  1.013183184 -2.11701547 54.67882
## 43  63.55247 62.03009  0.931365650  1.82920434 64.57519
## 44  61.83070 60.92338  0.523751625  2.43587032 66.92587
## 45  48.46623 56.32837 -0.500001013 -4.02306299 61.26314
## 46  53.12446 55.59361 -0.546952096 -2.29308203 53.71135
## 47  61.32752 56.82732 -0.190819319  3.16470225 56.87587
## 48  64.91346 58.97294  0.276467158  4.18819461 59.07238
## 49  53.79418 58.67654  0.161894531 -4.45271034 55.22634
## 50  59.86021 60.16438  0.427082607 -1.29862674 56.54535
## 51  62.78989 60.20495  0.349780787  2.87482043 63.75616
## 52  55.27280 56.76668 -0.407829290  1.34715682 64.74293
## 53  60.96869 59.98387  0.317174420 -1.73394643 51.90614
## 54  62.16132 61.56460  0.569886484 -0.35095650 59.00242
## 55  58.06684 59.35750  0.014489162  0.79208047 65.00931
## 56  62.23980 59.98025  0.136141143  1.80335175 60.71915
## 57  57.38166 59.71608  0.056078007 -2.03418319 58.38245
## 58  53.08558 57.23791 -0.450771682 -2.25164284 59.42120
## 59  61.89881 58.51497 -0.105204284  2.08795821 57.57922
## 60  62.13047 59.17671  0.048183894  2.37855742 60.21312
## 61  56.01710 58.75545 -0.045704908 -2.38626620 57.19071
## 62  56.63138 58.77906 -0.031842489 -2.19965876 56.45810
## 63  58.22594 57.70352 -0.240581241  1.30518789 60.83517
## 64  61.19625 58.00484 -0.132201306  2.78498218 59.84150
## 65  58.92870 59.24957  0.143185128 -1.35356708 55.48637
## 66  59.94121 60.49200  0.363034030 -1.37522538 57.19310
## 67  58.44445 59.36872  0.065771961  0.19045513 62.16022
## 68  65.97665 60.93736  0.366345716  3.91213376 62.21948
## 69  47.01992 56.13162 -0.668072152 -5.23263408 59.95014
## 70  61.11997 58.27621 -0.105540517  0.73426825 54.08832
## 71  66.83825 61.56152  0.572629740  2.73359360 58.36112
## 72  70.06445 63.74141  0.894082926  5.11758320 66.04628
## 73  48.29897 60.19394  0.005771895 -8.56380044 59.40286
## 74  53.44721 57.20500 -0.593169794 -1.51176309 60.93398
## 75  60.07681 56.90439 -0.534659480  2.95300727 59.34543
## 76  59.64961 55.63465 -0.681675492  4.56627316 61.48731
## 77  53.80663 57.91995 -0.088279003 -6.33856361 46.38917
## 78  62.94098 60.48010  0.441406403  0.47455719 56.31991
## 79  54.54828 57.19102 -0.304692054  0.15513806 63.87452
## 80  68.05963 59.52914  0.223870457  6.54838258 61.45260
## 81  62.34036 63.32337  0.937943412 -3.66079003 53.41444
## 82  68.41522 65.73305  1.232290971  1.57836054 64.73587
## 83  65.00977 66.12106  1.063433948 -0.47807578 67.12048
## 84  62.32254 62.62036  0.150606772  3.12528067 73.73288
## 85  50.92384 59.49643 -0.504300185 -6.11669112 59.11017
## 86  54.33806 56.49916 -1.002894725 -0.29136899 60.57049
## 87  65.61265 59.73405 -0.155337181  2.70026501 55.01819
## 88  65.58734 60.73205  0.075330694  3.99028519 62.70399
## 89  59.77175 62.83980  0.481815225 -4.59237412 54.69069
## 90  50.29896 58.22910 -0.536688039 -4.11075623 63.03025
## 91  67.42626 60.50585  0.025998563  4.81033977 60.39268
## 92  55.15428 56.78471 -0.723429826  1.17992873 64.52213
## 93  64.37850 61.22512  0.309338420 -0.71949320 51.46890
## 94  65.52567 64.77524  0.957496122 -1.68016484 57.42370
## 95  60.75222 61.81640  0.174227496  1.87308242 70.54308
## 96  61.54490 61.34036  0.044175167  0.69223250 63.17055
## 97  64.54914 62.93818  0.354902748  0.44573523 60.66504
## 98  55.28668 60.76258 -0.151196255 -3.57803611 61.61291
## 99  68.46943 63.00537  0.327600697  3.66857099 62.48447
## 100 71.46731 66.30981  0.922969088  2.92486396 64.02520
## 101       NA       NA           NA          NA 67.67852
```

note: c= 0.00 for the first 4 periods.

alternative from forecast package

```
y.3rd.exp.smoo <- hw(y.com.ts , h = 1 , initial = "simple", seasonal ="additive", alpha = 0.4, beta = 0
y.3rd.exp.smoo
```

```
##       Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 26 Q1       67.67846 58.91409 76.44283 54.27451 81.08241
```

$\hat{y}_t$ are as follows:

```
y.3rd.exp.smoo$model$fitted
```

```
##        Qtr1     Qtr2     Qtr3     Qtr4
## 1  42.57856 43.80229 56.04575 64.00255
## 2  42.93737 48.82891 64.90014 66.53431
## 3  50.27166 51.85959 55.28927 72.63834
## 4  52.24007 49.85879 59.71944 61.20327
## 5  45.74515 38.88507 49.92879 59.84158
## 6  47.06591 51.98105 48.71934 61.20343
## 7  59.79827 54.52252 59.75271 63.90791
## 8  57.97146 52.24460 57.42837 64.31202
## 9  51.84971 53.81210 60.99397 59.01277
## 10 51.96755 52.70631 51.91130 61.45102
## 11 55.13993 54.60292 64.43852 67.00839
## 12 61.35026 53.67757 56.78276 59.11185
## 13 55.28628 56.53373 63.69458 64.75944
## 14 51.94590 59.00151 64.96964 60.72413
## 15 58.40798 59.42476 57.55431 60.21283
## 16 57.20660 56.46293 60.81992 59.83920
## 17 55.49595 57.19771 62.15113 62.21677
## 18 59.95572 54.09215 58.35587 66.04384
## 19 59.40598 60.93692 59.34249 61.48536
## 20 46.39082 56.32205 63.87295 61.45115
## 21 53.41526 64.73736 67.11970 73.73186
## 22 59.11052 60.57150 55.01783 62.70330
## 23 54.69080 63.03091 60.39255 64.52168
## 24 51.46890 57.42412 70.54305 63.17027
## 25 60.66500 61.61318 62.48449 64.02503
```

optimal 3rd ES

```r
y.3rd.exp.smoo <- hw(y.com.ts , h = 1 , initial = "simple", seasonal ="additive")
y.3rd.exp.smoo$model$par
```

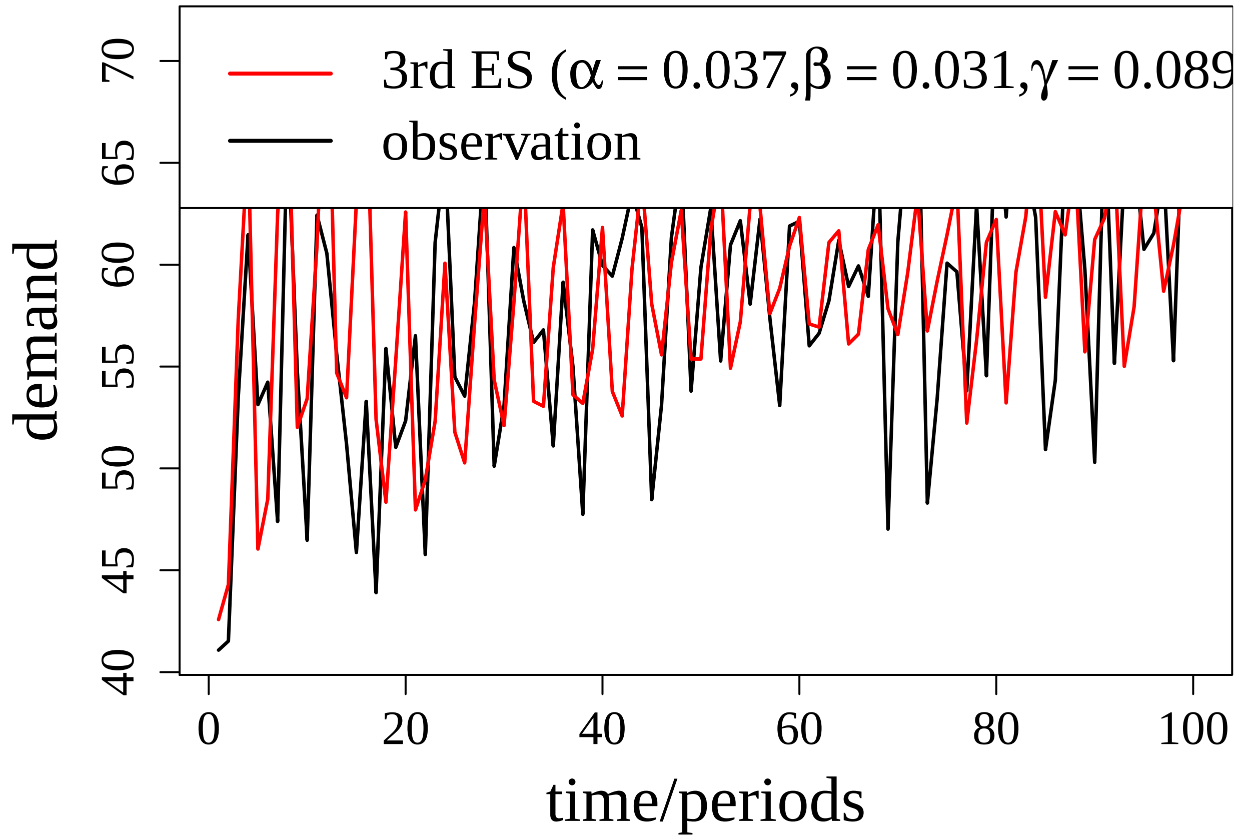
```
##       alpha        beta       gamma           l           b          s1          s2
##   0.1271242   0.1778950   0.2082107  49.3881624   1.4998638  12.0829047   4.0922186
##          s3          s4
## -7.8656613 -8.3094620
```

in the above values, * $l == a_0$ that is initial constant. * $s_t== c_t $ e.g $s1 = c_1, s2 = c_2, s3 = c_3, s4 = c_4$

```r
fc.stats(yhat = y.3rd.exp.smoo$model$fitted , y = y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 37.41774343  4.39198096  0.08625502  0.01153101
```

```r
#png(file = "forecast_ts_3rdem_best.png", bg = "transparent", width=600, height = 400)
#win.metafile("forecast_ts_3rdem_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis = 
lines(x, y.3rd.exp.smoo$model$fitted, col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("3rd ES (", alpha==0.037,",",beta==
```



```
#dev.off()
```

best smoothing model - AICc minimization

We assumed earlier that seasonal effects are additive. Additional smoothing methods: damped trends and multliplicative seasonality. see slide 3b of inventory management from Prof Dr T Kirschstein “Categorization and Optimal smoothing”, for more details.

In practice, we optimize the parameter of smoothing method in software suits. values for α, β, γ, also initial values are important.

determine optimal ES model

Smoothing functions can be optimized for forecasting criteria like MSE, MAPE etc. Typically, however, likelihood-based measures are used like Akaike information criterion (AIC). Parameters to be optimized over are the smoothing parameter (,...,) as well as the starting values (_0, _0,...). (T Kirschstein, 2020) slide 3b.14

```
best.smooth <- ets(y.com.ts)
best.smooth
```

```
## ETS(A,A,A)
##
```

```
## Call:
##  ets(y = y.com.ts)
##
##   Smoothing parameters:
##     alpha = 0.0047
##     beta  = 0.0047
##     gamma = 1e-04
##
##   Initial states:
##     l = 51.95
##     b = 0.2345
##     s = 3.7871 1.5938 -2.11 -3.2709
##
##   sigma:  5.2059
##
##      AIC     AICc      BIC
## 800.1357 802.1357 823.5823
```

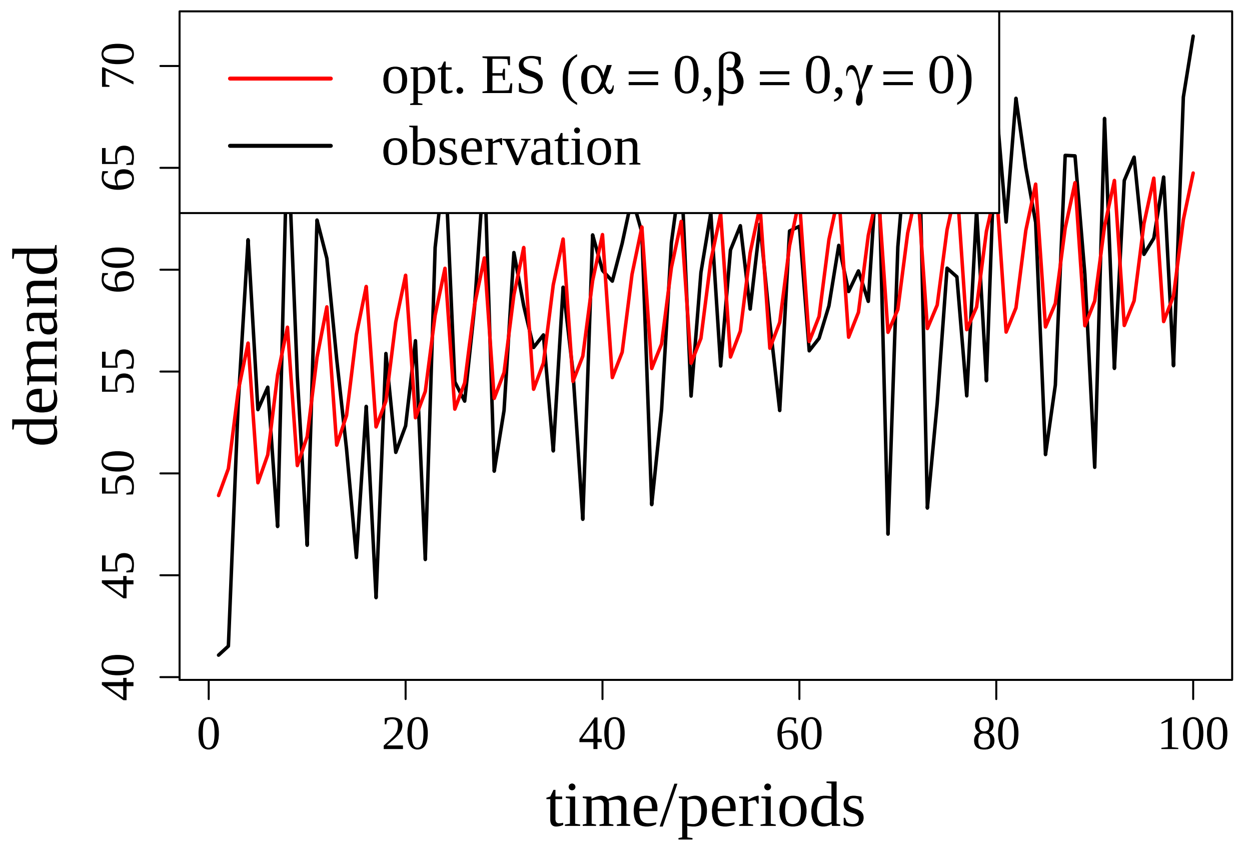
Error

```
fc.stats(yhat = best.smooth$fitted , y = y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 24.93287005  3.58279568  0.07368963  0.00804360
```

Optimal smoothing

```
#png(file = "forecast_ts_ets_best.png", bg = "transparent", width=600, height = 400)
#win.metafile("forecast_ts_ets_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis =
lines(x, best.smooth$fitted, col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("opt. ES (", alpha==0,",",beta==0,"
```



```r
#dev.off()
```

Arima

Optimal arima model

```r
best.arima <- auto.arima(y.com.ts)
best.arima$fitted
```

```
##        Qtr1     Qtr2     Qtr3     Qtr4
## 1  41.03774 41.30799 44.84595 49.60727
## 2  48.13018 48.31860 52.86375 55.03678
## 3  52.49216 49.78273 54.08183 59.09285
## 4  50.63245 49.51523 56.43202 60.06950
## 5  55.05135 52.35371 52.68206 59.01224
## 6  53.33877 51.53052 55.81648 56.72764
## 7  54.78267 52.57173 53.42378 56.95412
## 8  52.65546 54.88329 56.68486 58.54561
## 9  56.00047 52.77264 60.79542 61.36651
## 10 56.18215 55.44118 57.42335 61.16061
## 11 54.38602 55.06796 59.69653 59.78439
## 12 58.22931 57.46358 57.87369 60.08815
## 13 56.43334 54.68552 62.46293 61.20108
## 14 57.73964 60.32967 62.74430 60.63160
## 15 56.23159 58.49202 61.05533 62.25471
## 16 57.96057 59.59766 61.61917 59.33299
## 17 59.88582 59.71077 60.04358 61.62894
## 18 59.45372 56.58916 61.80490 61.94262
## 19 58.10269 58.63371 62.22980 62.39501
## 20 56.99669 58.59083 61.98788 62.29062
## 21 54.63094 60.23582 64.07113 65.40892
## 22 57.65532 59.73276 61.97268 61.79900
## 23 58.24294 62.07030 60.75626 65.72124
## 24 59.65821 62.66406 64.44410 62.21961
## 25 58.16908 59.99750 63.56365 64.23059
```

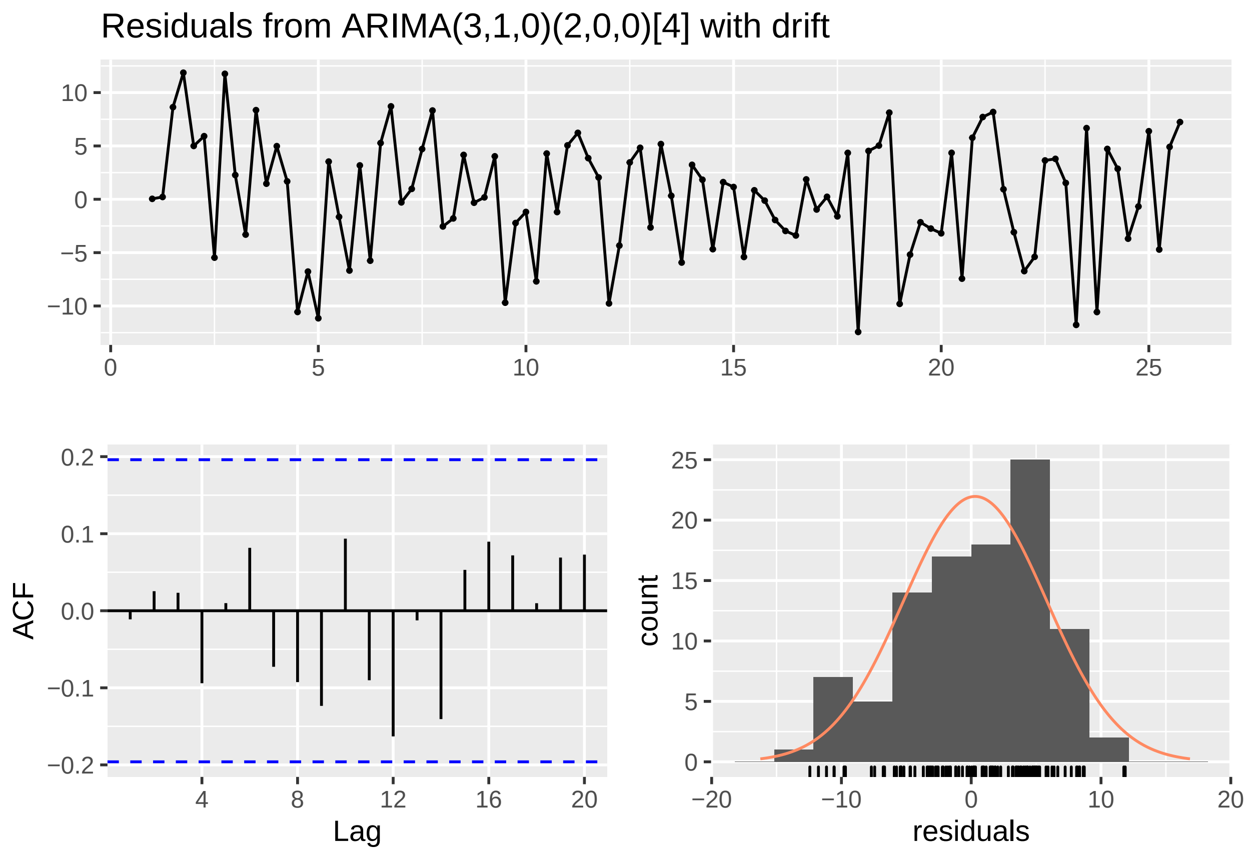
```
best.arima$coef
```

```
##        ar1        ar2        ar3       sar1       sar2      drift
## -0.9202750 -0.9247682 -0.8364942 -0.6552400 -0.4268379  0.1182317
```

```
fc.stats(yhat = best.arima$fitted , y = y.com)
```

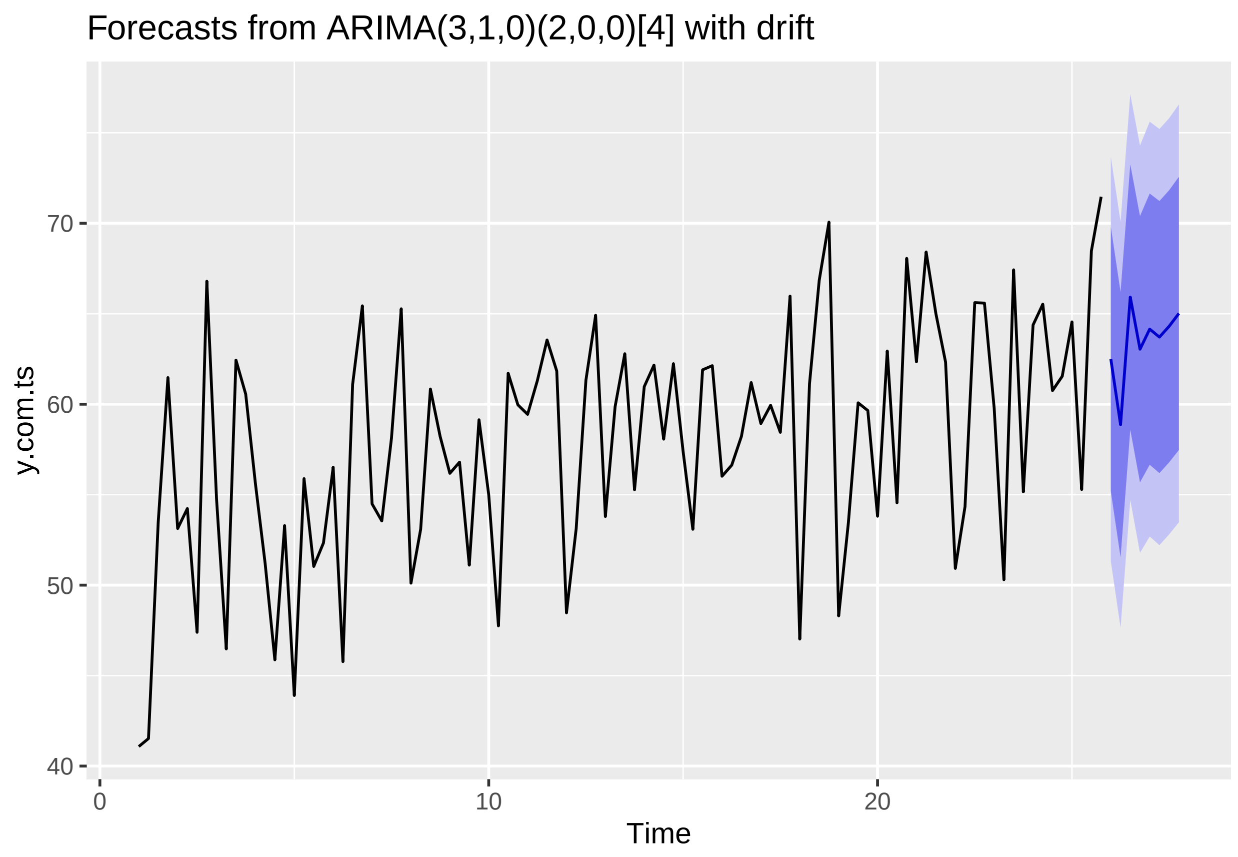
```
##         MSE         MAD        MAPE        MSLA
## 30.23025380  4.31259722  0.07921469  0.00950506
```

```
checkresiduals(best.arima)
```

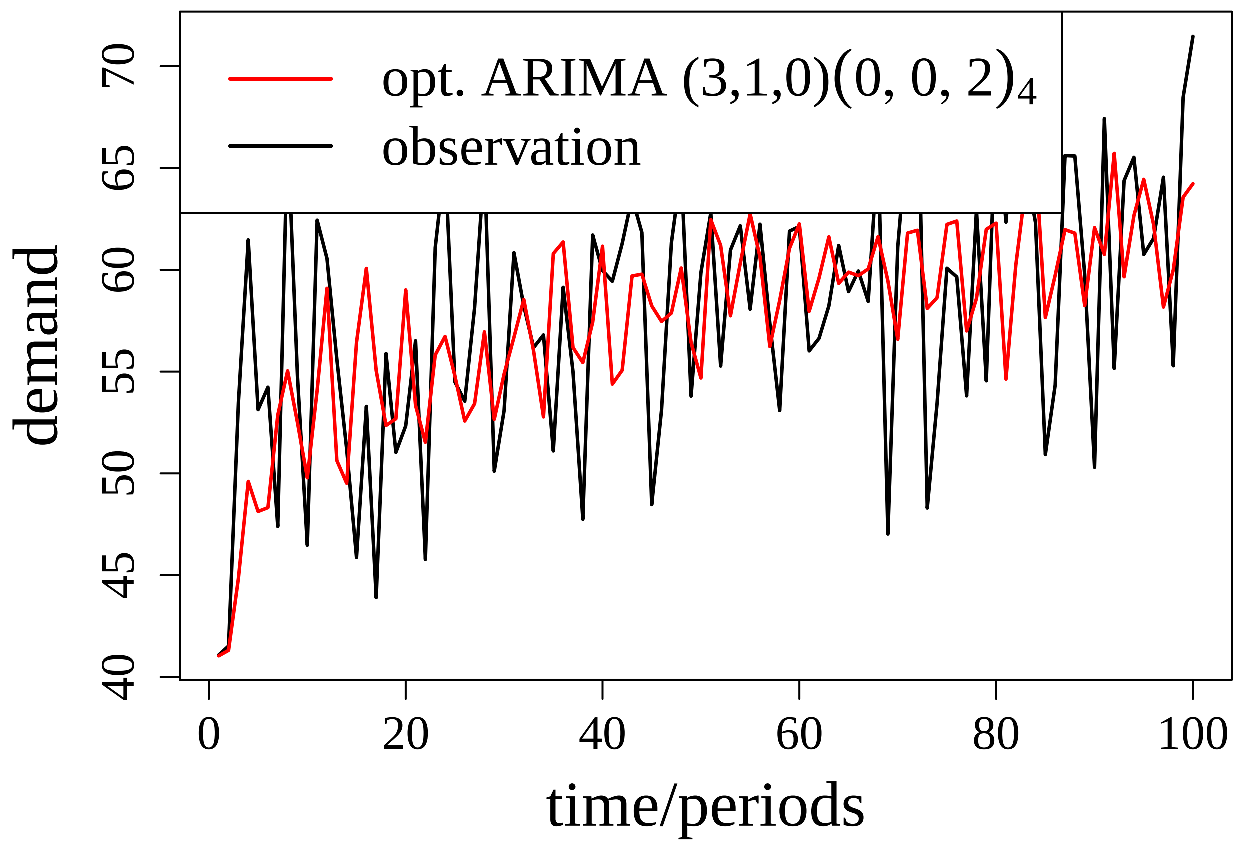


```
##
##  Ljung-Box test
##
## data:  Residuals from ARIMA(3,1,0)(2,0,0)[4] with drift
## Q* = 5.0462, df = 3, p-value = 0.1684
##
## Model df: 6.   Total lags used: 9
```

```
autoplot(forecast(best.arima))
```



```
#win.metafile("forecast_ts_arima_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis = 
lines(x, best.arima$fitted, col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("opt. ARIMA (3,1,0)",group("(",list
```



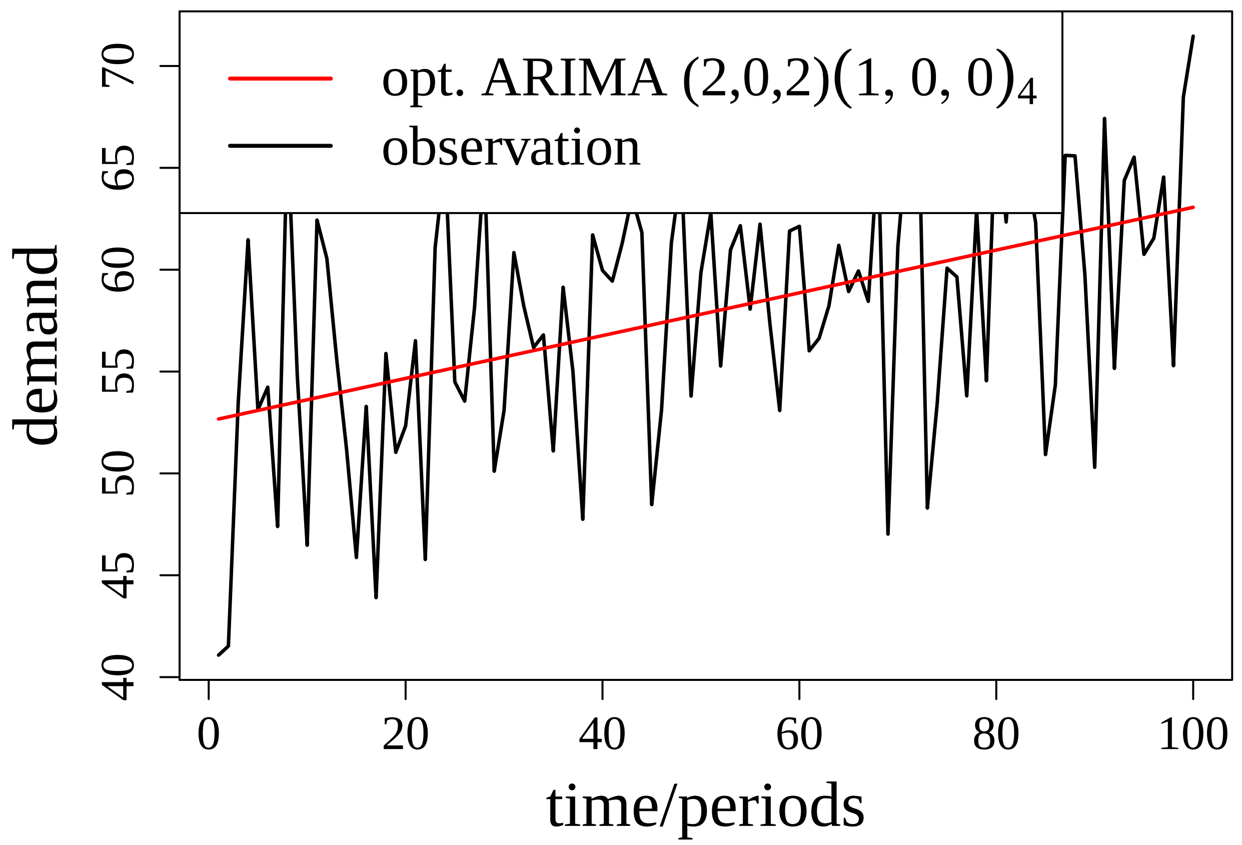
```
#dev.off()
```

optimal arima model with external regressors

```
best.arimax <- auto.arima(y.com.ts, xreg = 1:100)
best.arimax
```

```
## Series: y.com.ts
## Regression with ARIMA(0,0,0) errors
##
## Coefficients:
##       intercept    xreg
##         52.5648  0.1050
## s.e.     1.1482  0.0197
##
## sigma^2 estimated as 33.13:  log likelihood=-315.9
## AIC=637.8   AICc=638.05   BIC=645.62
```

```
#win.metafile("forecast_ts_arimax_best.wmf", width=9, height = 6)
par(family="serif", mar = c(5,5,.1,.1))
plot(x, y.com.ts, type="l", xlab= "time/periods", ylab ="demand", lwd = 1.75 , cex.lab = 2, cex.axis = 
lines(x, best.arimax$fitted, col="red", lwd=1.75)
legend("topleft", col=c("red","black"), legend = c(expression(paste("opt. ARIMA (2,0,2)",group("(",list
```



```
#dev.off()
```

```
fc.stats(yhat = best.arimax$fitted , y = y.com)
```

```
##         MSE         MAD        MAPE        MSLA
## 32.46438031  3.79032003  0.07963301  0.01043078
```

mn