**LECTURE 9 – TIME SERIES (continuation)**

**EXEMPLES**

## the age of death of successive kings of England, starting with William the Conqueror

# #Source: McNeill, "Interactive Data Analysis"

kings<-read.csv("E:\\29\_03\_2018\_Toshiba\\Alina\_27.03.2018\\0\_Docum univ\\2019-2020\\Cursuri\\Cursuri inregistrate\\Info 3\\Kings.csv", sep=",", header=F)

kings

plot(kings)

kingstimeseries <- ts(kings) ## stocarea datelor ca serie de timp

Sometimes the time series data set that you have may have been collected at regular intervals that were less than one year, for example, monthly or quarterly. In this case, you can specify the number of times that data was collected per year by using the ‘frequency’ parameter in the ts() function. For monthly time series data, you set frequency=12, while for quarterly time series data, you set frequency=4.

You can also specify the first year that the data was collected, and the first interval in that year by using the ‘start’ parameter in the ts() function. For example, if the first data point corresponds to the second quarter of 1986, you would set start=c(1986,2).

An example is a data set of the number of births per month in New York city, from January 1946 to December 1959 (<http://robjhyndman.com/tsdldata/data/nybirths.dat>):

births <- scan("http://robjhyndman.com/tsdldata/data/nybirths.dat")

births

birthstimeseries <- ts(births, frequency=12, start=c(1946,1))

Similarly, the third example (http://robjhyndman.com/tsdldata/data/fancy.dat) contains monthly sales for a souvenir shop at a beach resort town in Queensland, Australia, for January 1987-December 1993

souvenir <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")

souvenirtimeseries <- ts(souvenir, frequency=12, start=c(1987,1))

souvenirtimeseries

1. ***Plotting the time series***

Once you have read a time series into R, the next step is usually to make a plot of the time series data, which you can do with the plot.ts() function in R.

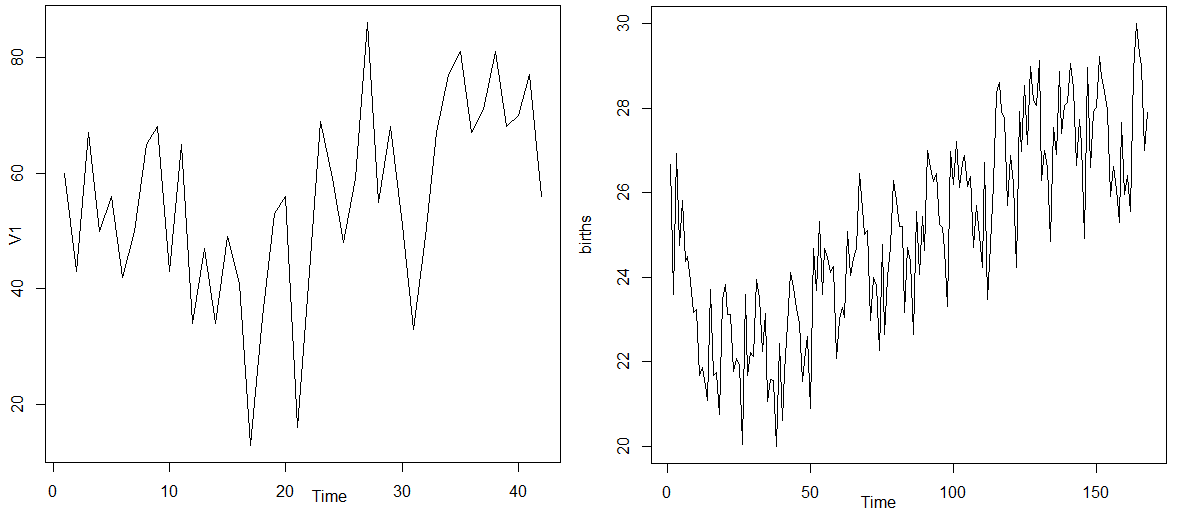
plot.ts(kingstimeseries)

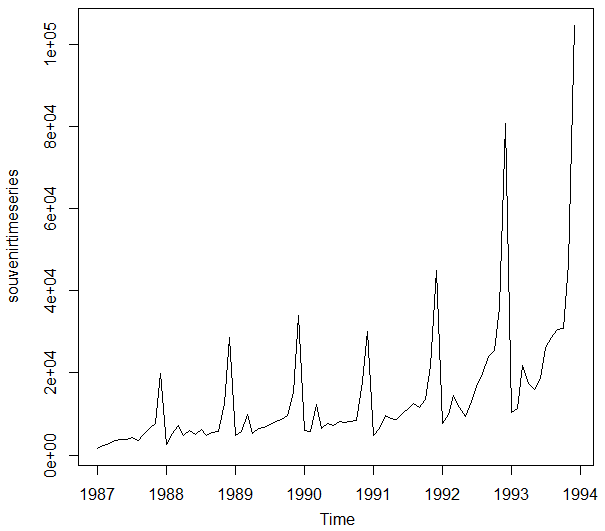
plot(births)

plot.ts(souvenirtimeseries)

We can see from the time plot of Kings data series that this time series could probably be described using an additive model, since the random fluctuations in the data are roughly constant in size over time.

For Births series, one can see that there seems to be seasonal variation in the number of births per month: there is a peak every summer, and a trough every winter. Again, it seems that this time series could probably be described using an additive model, as the seasonal fluctuations are roughly constant in size over time and do not seem to depend on the level of the time series, and the random fluctuations also seem to be roughly constant in size over time.

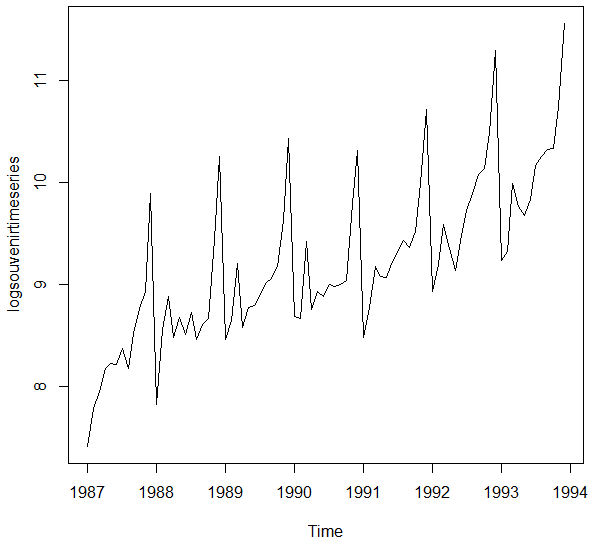




For Souvenirs series it appears that an additive model is not appropriate for describing this time series, since the size of the seasonal fluctuations and random fluctuations seem to increase with the level of the time series. Thus, we may need to transform the time series in order to get a transformed time series that can be described using an additive model. For example, we can transform the time series by calculating the natural log of the original data:

logsouvenirtimeseries <- log(souvenirtimeseries)

plot.ts(logsouvenirtimeseries)



Here we can see that the size of the seasonal fluctuations and random fluctuations in the log-transformed time series seem to be roughly constant over time, and do not depend on the level of the time series. Thus, the log-transformed time series can probably be described using an additive model.

***2. Decomposing time series***

***2.1. Decomposing non-seasonal data***

- involves trying to separate the time series into these components, that is, estimating the the trend component and the irregular component.

To estimate the trend component of a non-seasonal time series that can be described using an additive model, it is common to use a smoothing method, such as calculating the simple moving average of the time series.

The SMA() function in the “**TTR**” R package can be used to smooth time series data using a simple moving average.

library("TTR")

par(mfrow=c(2,2))

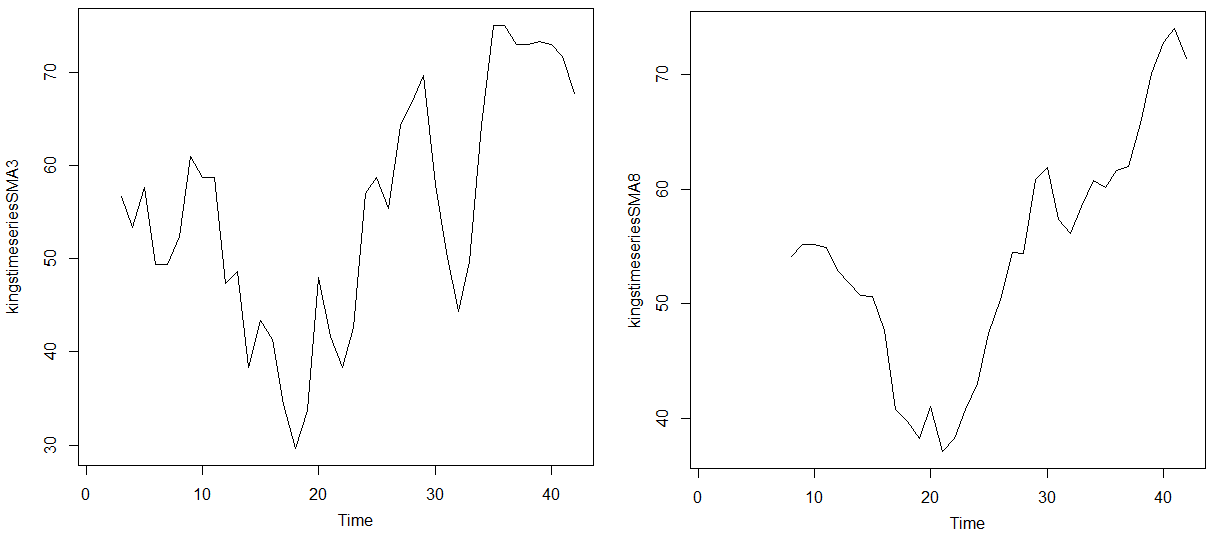
kingstimeseriesSMA3 <- SMA(kingstimeseries,n=3)

plot.ts(kingstimeseriesSMA3)

kingstimeseriesSMA8 <- SMA(kingstimeseries,n=8)

plot.ts(kingstimeseriesSMA8)

The data smoothed with a simple moving average of order 8 gives a clearer picture of the trend component, and we can see that the age of death of the English kings seems to have decreased from about 55 years old to about 38 years old during the reign of the first 20 kings, and then increased after that to about 73 years old by the end of the reign of the 40th king in the time series.



*Discover the random variable from Kings series and analyse it.*

***2.2. Decomposing Seasonal Data***

A seasonal time series consists of a trend component, a seasonal component and an irregular component. Decomposing the time series means separating the time series into these three components: that is, estimating these three components.

To estimate the trend component and seasonal component of a seasonal time series that can be described using an additive model, we can use the “decompose()” function in R. This function estimates the trend, seasonal, and irregular components of a time series that can be described using an additive model.

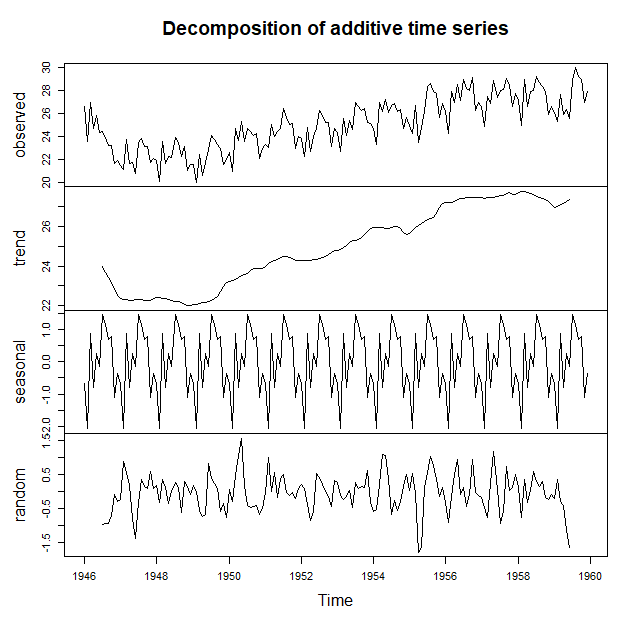
The function “decompose()” returns a list object as its result, where the estimates of the seasonal component, trend component and irregular component are stored in named elements of that list objects, called “seasonal”, “trend”, and “random” respectively.

birthstimeseriescomponents <- decompose(birthstimeseries)

birthstimeseriescomponents$seasonal # give the seasonal compound

birthstimeseriescomponents$seasonal

plot(birthstimeseriescomponents)



***2.3 Seasonally Adjusting***

If you have a seasonal time series that can be described using an additive model, you can seasonally adjust the time series by estimating the seasonal component, and subtracting the estimated seasonal component from the original time series. We can do this using the estimate of the seasonal component calculated by the “decompose()” function.

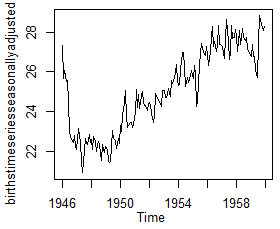
For example, to seasonally adjust the time series of the number of births per month in New York city, we can estimate the seasonal component using “decompose()”, and then subtract the seasonal component from the original time series:

birthstimeseriescomponents <- decompose(birthstimeseries)

birthstimeseriesseasonallyadjusted <- birthstimeseries - birthstimeseriescomponents$seasonal

plot(birthstimeseriesseasonallyadjusted)

You can see that the seasonal variation has been removed from the seasonally adjusted time series. The seasonally adjusted time series now just contains the trend component and an irregular component.



souvenir <- scan("http://robjhyndman.com/tsdldata/data/fancy.dat")

Read 84 items

souvenirtimeseries <- ts(souvenir, frequency=12, start=c(1987,1))

souvenirtimeseries

Jan Feb Mar Apr May Jun Jul

1987 1664.81 2397.53 2840.71 3547.29 3752.96 3714.74 4349.61

1988 2499.81 5198.24 7225.14 4806.03 5900.88 4951.34 6179.12

1989 4717.02 5702.63 9957.58 5304.78 6492.43 6630.80 7349.62

1990 5921.10 5814.58 12421.25 6369.77 7609.12 7224.75 8121.22

1991 4826.64 6470.23 9638.77 8821.17 8722.37 10209.48 11276.55

1992 7615.03 9849.69 14558.40 11587.33 9332.56 13082.09 16732.78

1993 10243.24 11266.88 21826.84 17357.33 15997.79 18601.53 26155.15

Aug Sep Oct Nov Dec

1987 3566.34 5021.82 6423.48 7600.60 19756.21

1988 4752.15 5496.43 5835.10 12600.08 28541.72

1989 8176.62 8573.17 9690.50 15151.84 34061.01

1990 7979.25 8093.06 8476.70 17914.66 30114.41

1991 12552.22 11637.39 13606.89 21822.11 45060.69

1992 19888.61 23933.38 25391.35 36024.80 80721.71

1993 28586.52 30505.41 30821.33 46634.38 104660.67

ST<-ts(souvenir)

ST

Time Series:

Start = 1

End = 84

Frequency = 1

[1] 1664.81 2397.53 2840.71 3547.29 3752.96 3714.74 4349.61

[8] 3566.34 5021.82 6423.48 7600.60 19756.21 2499.81 5198.24

[15] 7225.14 4806.03 5900.88 4951.34 6179.12 4752.15 5496.43

[22] 5835.10 12600.08 28541.72 4717.02 5702.63 9957.58 5304.78

[29] 6492.43 6630.80 7349.62 8176.62 8573.17 9690.50 15151.84

[36] 34061.01 5921.10 5814.58 12421.25 6369.77 7609.12 7224.75

[43] 8121.22 7979.25 8093.06 8476.70 17914.66 30114.41 4826.64

[50] 6470.23 9638.77 8821.17 8722.37 10209.48 11276.55 12552.22

[57] 11637.39 13606.89 21822.11 45060.69 7615.03 9849.69 14558.40

[64] 11587.33 9332.56 13082.09 16732.78 19888.61 23933.38 25391.35

[71] 36024.80 80721.71 10243.24 11266.88 21826.84 17357.33 15997.79

[78] 18601.53 26155.15 28586.52 30505.41 30821.33 46634.38 104660.67

ST<-ts(souvenir,start=c(1987,1))

ST

Time Series:

Start = 1987

End = 2070

Frequency = 1

[1] 1664.81 2397.53 2840.71 3547.29 3752.96 3714.74 4349.61

[8] 3566.34 5021.82 6423.48 7600.60 19756.21 2499.81 5198.24

[15] 7225.14 4806.03 5900.88 4951.34 6179.12 4752.15 5496.43

[22] 5835.10 12600.08 28541.72 4717.02 5702.63 9957.58 5304.78

[29] 6492.43 6630.80 7349.62 8176.62 8573.17 9690.50 15151.84

[36] 34061.01 5921.10 5814.58 12421.25 6369.77 7609.12 7224.75

[43] 8121.22 7979.25 8093.06 8476.70 17914.66 30114.41 4826.64

[50] 6470.23 9638.77 8821.17 8722.37 10209.48 11276.55 12552.22

[57] 11637.39 13606.89 21822.11 45060.69 7615.03 9849.69 14558.40

[64] 11587.33 9332.56 13082.09 16732.78 19888.61 23933.38 25391.35

[71] 36024.80 80721.71 10243.24 11266.88 21826.84 17357.33 15997.79

[78] 18601.53 26155.15 28586.52 30505.41 30821.33 46634.38 104660.67

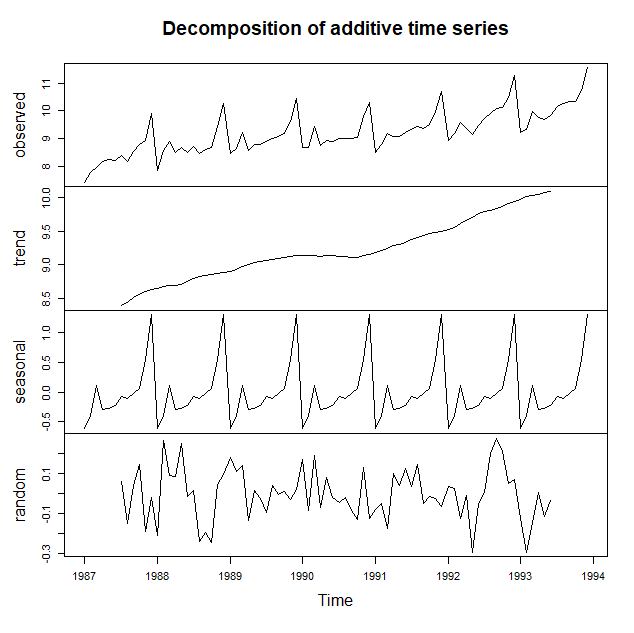
plot.ts(souvenirtimeseries)

logsouvenirtimeseries <- log(souvenirtimeseries)

plot.ts(logsouvenirtimeseries)

DLSTS<- decompose(logsouvenirtimeseries)

plot(DLSTS)



3. TIME SERIES PredicTiON

***3.1. Extrapolation***

yt = y1 + (t-1)⋅  , **N ,** t > n, (1)

y1 and yn are given.

 **N**\*. (2)

***3.2. Moving average method***

 (3)

εt = yt – ft , t ,

 (4)

 (5)

***3.3. Single exponential smoothing***

 (7)

If: F1 = y1 then:

F4 = α⋅y3 + α⋅(1-α)⋅y2 + α⋅(1-α)2⋅y1

Ft+1 = Ft + α⋅(yt-Ft). (8)

εt = yt-Ft,(9)

***Exemple:***

The file http://robjhyndman.com/tsdldata/hurst/precip1.dat contains total annual rainfall in inches for London, from 1813-1912 (original data from Hipel and McLeod, 1994). We can read the data into R and plot it by typing:

rain <- scan("http://robjhyndman.com/tsdldata/hurst/precip1.dat",skip=1)

rainseries <- ts(rain,start=c(1813))

library(trend)

mk.test(rainseries)

Mann-Kendall trend test

data: rainseries

z = -0.28888, n = 100, p-value = 0.7727

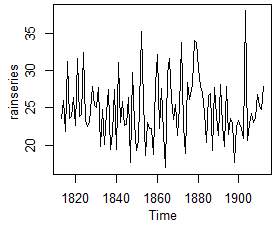
alternative hypothesis: true S is not equal to 0

sample estimates:

S varS tau

-9.800000e+01 1.127480e+05 -1.980198e-02

plot.ts(rainseries)



You can see from the plot that there is roughly constant level (the mean stays constant at about 25 inches). The random fluctuations in the time series seem to be roughly constant in size over time, so it is probably appropriate to describe the data using an additive model. Thus, we can make forecasts using simple exponential smoothing.

To make forecasts using simple exponential smoothing in R, we can fit a simple exponential smoothing predictive model using the “HoltWinters()” function in R.

To use HoltWinters() for simple exponential smoothing, we need to set the parameters beta=FALSE and gamma=FALSE in the HoltWinters() function (the beta and gamma parameters are used for Holt’s exponential smoothing, or Holt-Winters exponential smoothing, as described below).

The HoltWinters() function returns a list variable, that contains several named elements. For example, to use simple exponential smoothing to make forecasts for the time series of annual rainfall in London, we type:

rainseriesforecasts <- HoltWinters(rainseries, beta=FALSE, gamma=FALSE)

rainseriesforecasts

Smoothing parameters:

alpha: 0.02412151

beta : FALSE

gamma: FALSE

Coefficients:

[,1]

a 24.67819

The output of HoltWinters() tells us that the estimated value of the alpha parameter is about 0.024. This is very close to zero, telling us that the forecasts are based on both recent and less recent observations (although somewhat more weight is placed on recent observations).

By default, HoltWinters() just makes forecasts for the same time period covered by our original time series. In thiscase, our original time series included rainfall for London from 1813-1912, so the forecasts are also for 1813-1912.

In the example above, we have stored the output of the HoltWinters() function in the list variable “rainseriesforecasts”.

The forecasts made by HoltWinters() are stored in a named element of this list variable called “fitted”, so we can get their values by typing:

fitted<-rainseriesforecasts$fitted

Time Series:

Start = 1814

End = 1912

Frequency = 1

Xhat level

1814 23.56000 23.56000

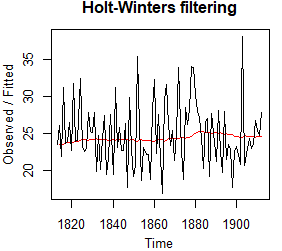
1815 23.62054 23.62054

1816 23.57808 23.57808

1817 23.76290 23.76290

We can plot the original time series against the forecasts by typing:

plot(rainseriesforecasts)



The plot shows the original time series in black, and the forecasts as a red line. The time series of forecasts is much smoother than the time series of the original data here. As a measure of the accuracy of the forecasts, we can calculate the sum of squared errors for the in-sample forecast errors, that is, the forecast errors for the time period covered by our original time series. The sum-of-squared-errors is stored in a named element of the list variable “rainseriesforecasts” called “SSE”, so we can get its value by typing:

rainseriesforecasts$SSE

[1] 1828.855

That is, here the sum-of-squared-errors is 1828.855.

It is common in simple exponential smoothing to use the first value in the time series as the initial value for the level. For example, in the time series for rainfall in London, the first value is 23.56 (inches) for rainfall in 1813. You can specify the initial value for the level in the HoltWinters() function by using the “l.start” parameter. For example, to make forecasts with the initial value of the level set to 23.56, we type:

HoltWinters(rainseries, beta=FALSE, gamma=FALSE, l.start=23.56)

As explained above, by default HoltWinters() just makes forecasts for the time period covered by the original data, which is 1813-1912 for the rainfall time series.

We can make forecasts for further time points by using the “forecast.HoltWinters()” function in the R “forecast” package. To use the forecast.HoltWinters() function, we first need to install the “forecast” R package. When using the forecast.HoltWinters() function, as its first argument (input), you pass it the predictive model that you have already fitted using the HoltWinters() function.

For example, in the case of the rainfall time series, we stored the predictive model made using HoltWinters() in the variable “rainseriesforecasts”. You specify how many further time points you want to make forecasts for by using the “h” parameter in forecast.HoltWinters(). For example, to make a forecast of rainfall for the years 1814-1820 (8 more years) using forecast.HoltWinters(), we type:

library("forecast")

y <- forecast:::forecast.HoltWinters(rainseriesforecasts, h=8) # prediction for the nex 8 years

y

Point Forecast Lo 80 Hi 80 Lo 95 Hi 95

1913 24.67819 19.17493 30.18145 16.26169 33.09470

1914 24.67819 19.17333 30.18305 16.25924 33.09715

1915 24.67819 19.17173 30.18465 16.25679 33.09960

1916 24.67819 19.17013 30.18625 16.25434 33.10204

1917 24.67819 19.16853 30.18785 16.25190 33.10449

1918 24.67819 19.16694 30.18945 16.24945 33.10694

1919 24.67819 19.16534 30.19105 16.24701 33.10938

1920 24.67819 19.16374 30.19265 16.24456 33.11182

The forecast.HoltWinters() function gives you the forecast for a year, a 80% prediction interval for the forecast, and a 95% prediction interval for the forecast. For example, the forecasted rainfall for 1920 is about 24.68 inches, with a 95% prediction interval of (16.24, 33.11).

To plot the predictions made by forecast.HoltWinters(), we can use the “plot()” function:

plot(y)

The ‘forecast errors’ are calculated as the observed values minus predicted values, for each time point. We can only calculate the forecast errors for the time period covered by our original time series, which is 1813-1912 for the rainfall data. As mentioned above, one measure of the accuracy of the predictive model is the sum-of-squared-errors (SSE) for the in-sample forecast errors.

The in-sample forecast errors are stored in the named element “residuals” of the list variable returned by forecast.HoltWinters(). If the predictive model cannot be improved upon, there should be no correlations between forecast errors for successive predictions. In other words, if there are correlations between forecast errors for successive predictions, it is likely that the simple exponential smoothing forecasts could be improved upon by another forecasting technique.

y$residual

Time Series:

Start = 1814

End = 1912

Frequency = 1

[1] NA 2.5100000 -1.7605450 7.6619220 -0.1128951 0.1198281

[7] 2.6469377 -1.1569105 7.8909960 -0.1293468 0.1237733 8.4407877

[13] -0.9328169 -1.6003159 -1.1317139 3.7755848 1.1245120 0.8573870

[19] 3.5167056 -4.5081227 0.5606200 -4.1129030 0.2063065 3.2813300

[25] -4.7778206 -2.4725723 3.4470698 -4.6960787 7.1171978 -1.0944797

[31] 1.6919208 -1.5488909 -1.4115293 2.2325189 -6.4813328 5.7850067

[37] -1.2345364 -4.9147575 -3.3862062 11.4054742 1.6803570 -5.6001757

[43] -1.0550911 -1.8796407 -1.8643009 -5.2293312 4.3368082 8.2621979

[49] -1.9070988 3.4389033 -2.6240483 -7.2207523 5.5034232 7.4906723

[55] 1.9599860 -0.9372918 1.1053171 -3.0213449 0.7515345 9.5734064

[61] -1.8475186 -5.6529537 4.1034041 1.7244238 3.6928281 9.5137515

[67] 9.0242655 5.2665866 2.7795486 1.9325016 -0.8541132 -4.8835107

[73] 1.5242869 1.8575188 -5.9872873 2.6871351 -1.2676827 -3.8571042

[79] 3.1559349 -2.4601910 -5.2108475 3.0548460 -3.4888415 -1.3546853

[85] -1.9820083 -7.1041993 -2.0828352 -1.2925941 -2.3714148 -3.6442127

[91] 13.7036912 -4.0768625 -1.6585224 -0.3285164 -1.5705920 -0.8727070

[97] 2.2283440 0.7845930 0.1956674 3.2809476

df <- na.omit(y$residual)

df

Time Series:

Start = 1814

End = 1912

Frequency = 1

[1] 2.5100000 -1.7605450 7.6619220 -0.1128951 0.1198281 2.6469377

[7] -1.1569105 7.8909960 -0.1293468 0.1237733 8.4407877 -0.9328169

[13] -1.6003159 -1.1317139 3.7755848 1.1245120 0.8573870 3.5167056

[19] -4.5081227 0.5606200 -4.1129030 0.2063065 3.2813300 -4.7778206

[25] -2.4725723 3.4470698 -4.6960787 7.1171978 -1.0944797 1.6919208

[31] -1.5488909 -1.4115293 2.2325189 -6.4813328 5.7850067 -1.2345364

[37] -4.9147575 -3.3862062 11.4054742 1.6803570 -5.6001757 -1.0550911

[43] -1.8796407 -1.8643009 -5.2293312 4.3368082 8.2621979 -1.9070988

[49] 3.4389033 -2.6240483 -7.2207523 5.5034232 7.4906723 1.9599860

[55] -0.9372918 1.1053171 -3.0213449 0.7515345 9.5734064 -1.8475186

[61] -5.6529537 4.1034041 1.7244238 3.6928281 9.5137515 9.0242655

[67] 5.2665866 2.7795486 1.9325016 -0.8541132 -4.8835107 1.5242869

[73] 1.8575188 -5.9872873 2.6871351 -1.2676827 -3.8571042 3.1559349

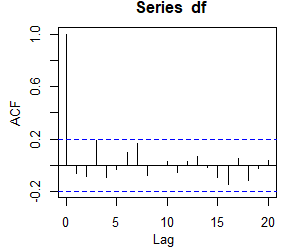
[79] -2.4601910 -5.2108475 3.0548460 -3.4888415 -1.3546853 -1.9820083

[85] -7.1041993 -2.0828352 -1.2925941 -2.3714148 -3.6442127 13.7036912

[91] -4.0768625 -1.6585224 -0.3285164 -1.5705920 -0.8727070 2.2283440

[97] 0.7845930 0.1956674 3.2809476

acf(df,lag.max=20)



Pentru a testa existenta unor corelatii nenule la intarzierile 1-20, se face un test Ljung-Box test folosind functia “Box.test()”. Intarzierea maxima pe care vrem sa o analizam este specificata de parametrul “lag” din functia Box.test().

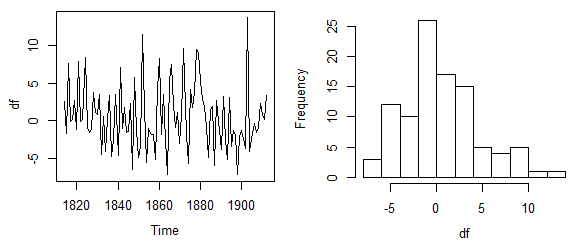
Box.test(df, lag=20, type="Ljung-Box")

Box-Ljung test

data: df

X-squared = 17.401, df = 20, p-value = 0.6268

plot(df); hist(y$residual)



The Ljung-Box test showed that there is little evidence of non-zero autocorrelations in the in-sample forecast errors, and the distribution of forecast errors seems to be normally distributed with mean zero. This suggests that the simple exponential smoothing method provides an adequate predictive model for London rainfall, which probably cannot be improved upon.

Furthermore, the assumptions that the 80% and 95% predictions intervals were based upon (that there are no autocorrelations in the forecast errors, and the forecast errors are normally distributed with mean zero and

constant variance) are probably valid.