Week 4 - Exponential Smoothing & Autoregressions

Jonathan Thong

2023-03-17

Exponential Smoothing

As always, let's make sure that our environment is clear:

```
rm(list = ls())
```

We will be using the **fpp3**, **forecast**, **fpp tidyverse** and **tidyquant** packages in this section. Please make sure to install them using the **install.packages()** function. Alternatively you can use the following lines of code:

```
if("fpp3" %in% rownames(installed.packages()) == FALSE) install.packages("fpp3")
if("forecast" %in% rownames(installed.packages()) == FALSE) install.packages("forecast")
if("fpp" %in% rownames(installed.packages()) == FALSE) install.packages("fpp")
if("tidyverse" %in% rownames(installed.packages()) == FALSE) install.packages("tidyverse")
if("tidyquant" %in% rownames(installed.packages()) == FALSE) install.packages("tidyquant")

library(fpp3)
```

```
## -- Attaching packages ---------------------------------------------- fpp3 0.5 --

## v tibble      3.1.8      v tsibble     1.1.3
## v dplyr       1.0.10     v tsibbledata 0.4.1
## v tidyr       1.3.0      v feasts      0.3.0
## v lubridate   1.8.0      v fable       0.3.2
## v ggplot2     3.4.1      v fabletools  0.3.2

## -- Conflicts ------------------------------------------------- fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()         masks stats::lag()
## x tsibble::setdiff()   masks base::setdiff()
## x tsibble::union()     masks base::union()
```

```
library(forecast)
```

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

##
## Attaching package: 'forecast'

## The following objects are masked from 'package:fabletools':
##
##     accuracy, forecast
```

```r
library(fpp)
```

```
## Loading required package: fma

## Loading required package: expsmooth

## Loading required package: lmtest

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following object is masked from 'package:tsibble':
##
##     index

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

## Loading required package: tseries

##
## Attaching package: 'fpp'

## The following object is masked from 'package:fpp3':
##
##     insurance
```

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --

## v readr   2.1.2     v stringr 1.5.0
## v purrr   1.0.1     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x tsibble::intersect()     masks lubridate::intersect(), base::intersect()
## x tsibble::interval()      masks lubridate::interval()
## x dplyr::lag()             masks stats::lag()
## x tsibble::setdiff()       masks lubridate::setdiff(), base::setdiff()
## x tsibble::union()         masks lubridate::union(), base::union()
```

```r
library(tidyquant)
```

```
## Loading required package: PerformanceAnalytics
## Loading required package: xts
##
## Attaching package: 'xts'
##
## The following objects are masked from 'package:dplyr':
##
##     first, last
##
##
## Attaching package: 'PerformanceAnalytics'
##
```

```
## The following object is masked from 'package:fpp3':
##
##     prices
##
## The following object is masked from 'package:graphics':
##
##     legend
##
## Loading required package: quantmod
## Loading required package: TTR
##
## Attaching package: 'tidyquant'
##
## The following object is masked from 'package:fable':
##
##     VAR
```

Now let's use the **tidyquant** package to download some stock price data directly into our environment. Using the **tq_get()** function, let's grab the daily stock price for Netflix Inc for the year of 2022 from Yahoo Finance and store it in an object called **nflx.p**.

```
nflx.p <- tq_get("NFLX", get = "stock.prices", from = "2022-01-01", to = "2022-12-31" )
head(nflx.p)
```

```
## # A tibble: 6 x 8
##   symbol date        open  high   low close  volume adjusted
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>
## 1 NFLX   2022-01-03  606.  610.  591.  597. 3067500     597.
## 2 NFLX   2022-01-04  600.  600.  582.  591. 4393100     591.
## 3 NFLX   2022-01-05  592   593.  567.  568. 4148700     568.
## 4 NFLX   2022-01-06  554.  563.  542.  553. 5711800     553.
## 5 NFLX   2022-01-07  549.  553.  538.  541. 3382900     541.
## 6 NFLX   2022-01-10  538.  544.  526.  540. 4486100     540.
```

Now, we want to take the adjusted prices and compute from them the daily return using the **tq_transmute** function. This creates a new data frame with the daily returns which we shall call **nflx.r**:

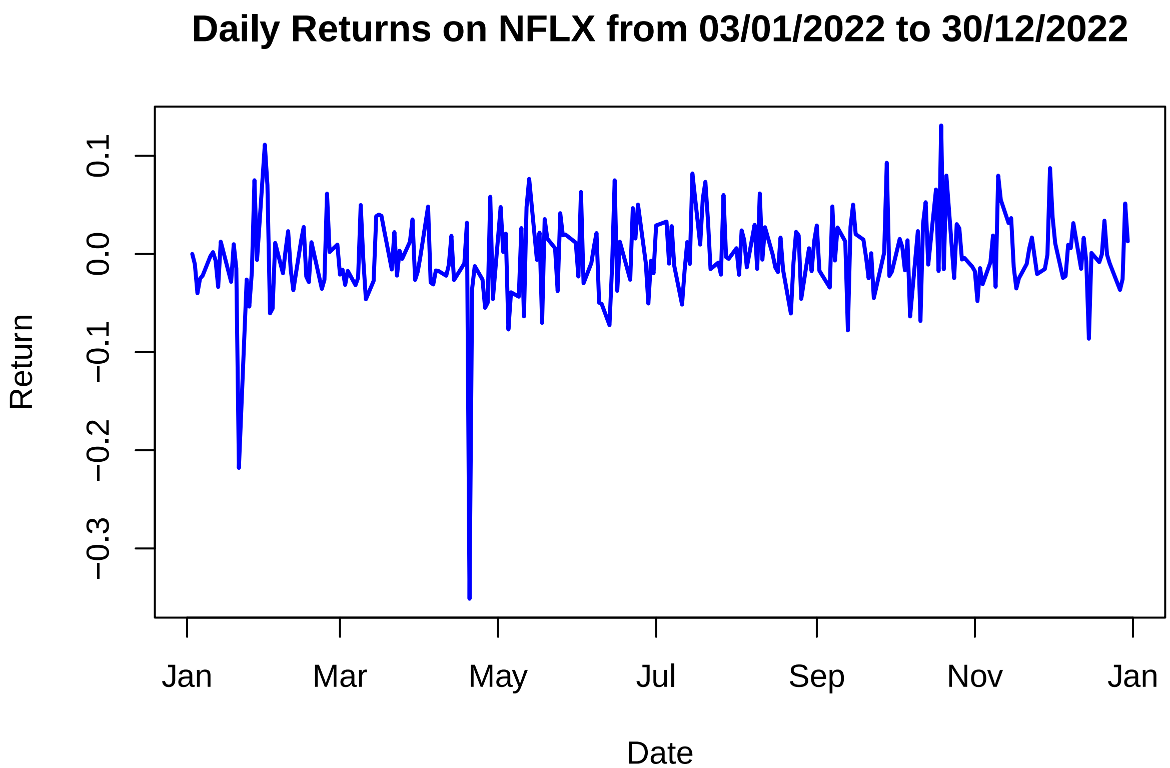
```
nflx.r <- tq_transmute(nflx.p, select = adjusted, mutate_fun = periodReturn, period = "daily", col_name
head(nflx.r)
```

```
## # A tibble: 6 x 2
##   date       daily.returns
##   <date>             <dbl>
## 1 2022-01-03       0
## 2 2022-01-04      -0.0104
## 3 2022-01-05      -0.0400
## 4 2022-01-06      -0.0251
## 5 2022-01-07      -0.0221
## 6 2022-01-10      -0.00224
```

Let's generate a plot of the daily returns. Notice that there is no clear trend of seasonal pattern present in the data:

```
plot(nflx.r$date, nflx.r$daily.returns,
     main = "Daily Returns on NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Return",
```

```
    type = 'l',
    lwd = 2.0,
    col = "blue")
```



Now let us proceed to generate a smoothed series using simple exponential smoothing in which we set the smoothing parameter $\alpha = 0.5$. We can do this using the **ses** function:

```
nflx.r.ses1 <- ses(nflx.r$daily.returns, alpha = 0.5)
summary(nflx.r.ses1)
```

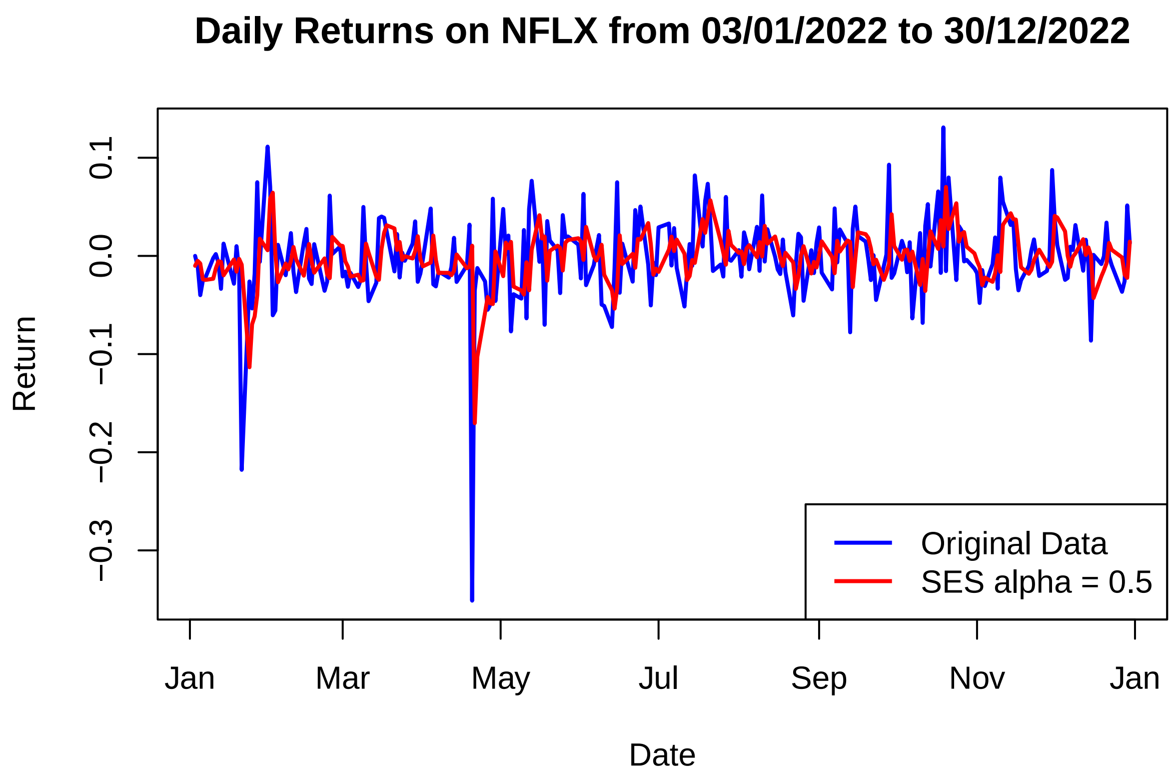
```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = nflx.r$daily.returns, alpha = 0.5)
##
##   Smoothing parameters:
##     alpha = 0.5
##
##   Initial states:
##     l = -0.0101
##
##   sigma:  0.0504
##
##       AIC      AICc       BIC
## -110.6918 -110.6434 -103.6409
##
```

```
## Error measures:
##                        ME       RMSE        MAE MPE MAPE     MASE       ACF1
## Training set 0.0001894896 0.05022732 0.03523807 Inf  Inf 0.812274 -0.2581886
##
## Forecasts:
##     Point Forecast       Lo 80      Hi 80       Lo 95     Hi 95
## 252     0.01372477 -0.05090212 0.07835166 -0.08511353 0.1125631
## 253     0.01372477 -0.05853029 0.08597983 -0.09677980 0.1242293
## 254     0.01372477 -0.06542668 0.09287622 -0.10732693 0.1347765
## 255     0.01372477 -0.07176857 0.09921811 -0.11702601 0.1444755
## 256     0.01372477 -0.07767145 0.10512099 -0.12605369 0.1535032
## 257     0.01372477 -0.08321556 0.11066510 -0.13453267 0.1619822
## 258     0.01372477 -0.08845931 0.11590885 -0.14255230 0.1700018
## 259     0.01372477 -0.09344680 0.12089634 -0.15018000 0.1776295
## 260     0.01372477 -0.09821228 0.12566182 -0.15746818 0.1849177
## 261     0.01372477 -0.10278301 0.13023255 -0.16445850 0.1919080
```

Note that the **ses** function also automatically generates $h = 10$ step ahead forecasts.

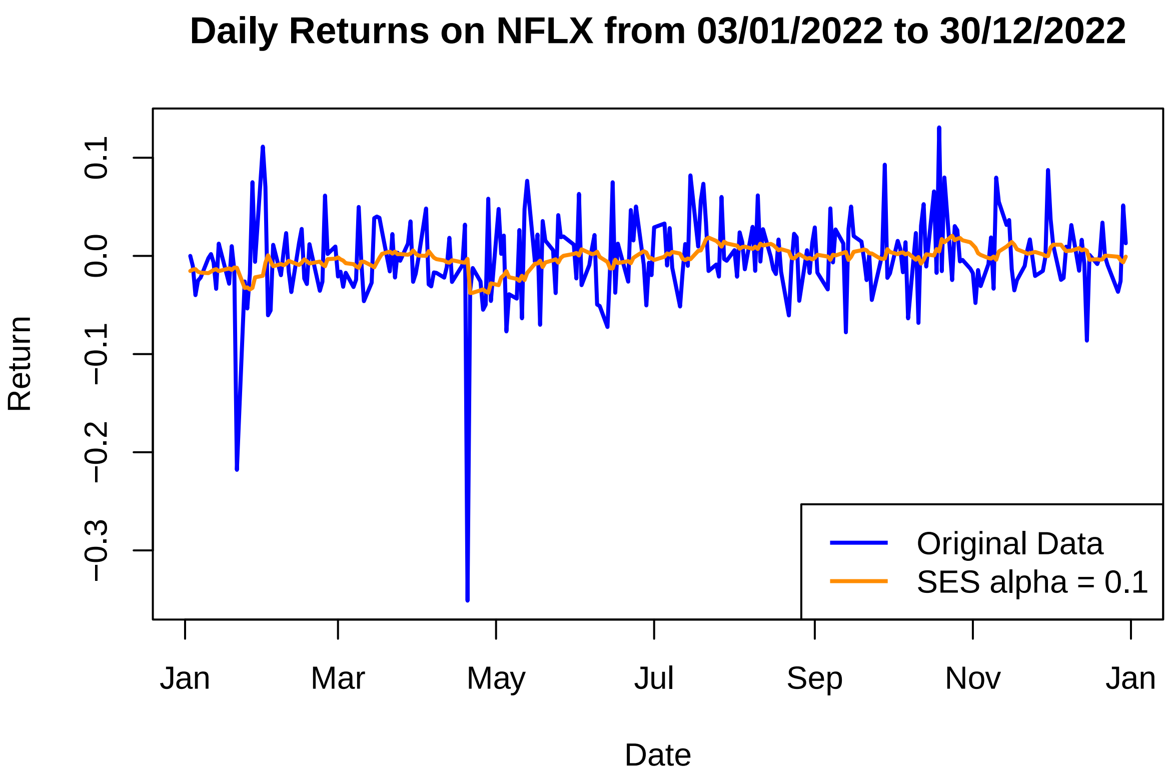
Having generated the smoothed series we can then plot it alongside our original data:

```
plot(nflx.r$date, nflx.r$daily.returns,
     main = "Daily Returns on NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Return",
     type = 'l',
     lwd = 2.0,
     col = "blue")
lines(nflx.r$date,nflx.r.ses1$fitted, lwd = 2.0, col = "red")

legend(x = "bottomright",
       legend = c("Original Data", "SES alpha = 0.5"),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("blue", "red"))
```



Now let's set $\alpha = 0.1$ and have a look at the smoothed series. We can see clearly that the closer α is to zero, the greater the degree of smoothing:

```
nflx.r.ses2 <- ses(nflx.r$daily.returns, alpha = 0.1)

plot(nflx.r$date, nflx.r$daily.returns,
     main = "Daily Returns on NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Return",
     type = 'l',
     lwd = 2.0,
     col = "blue")
lines(nflx.r$date,nflx.r.ses2$fitted, lwd = 2.0, col = "darkorange")

legend(x = "bottomright",
       legend = c("Original Data", "SES alpha = 0.1"),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("blue", "darkorange"))
```



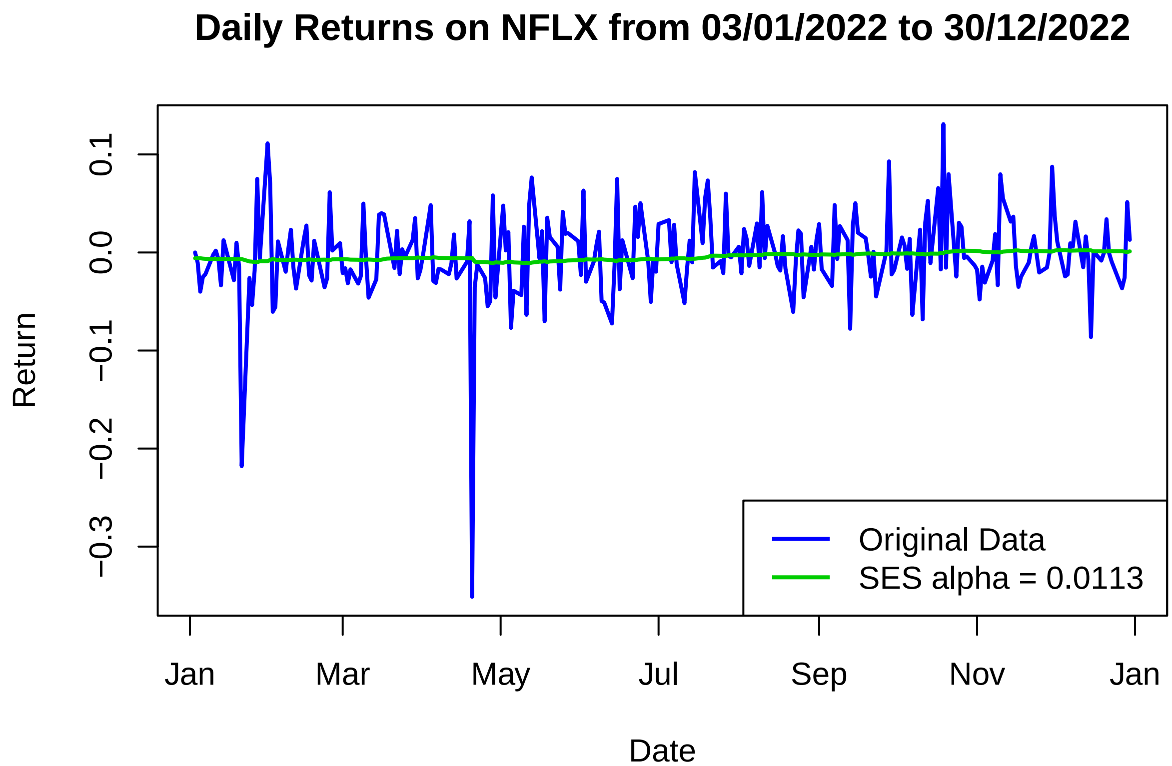
Now if we don't want to choose the parameter values ourselves, we can let the **ses** function estimate them for us:

```
nflx.r.ses3 <- ses(nflx.r$daily.returns, initial = "optimal")

summary(nflx.r.ses3)
```

```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
## Simple exponential smoothing
##
## Call:
##  ses(y = nflx.r$daily.returns, initial = "optimal")
##
##   Smoothing parameters:
##     alpha = 0.0113
##
##   Initial states:
##     l = -0.0059
##
##   sigma:  0.0443
##
##       AIC      AICc       BIC
## -174.1165 -174.0193 -163.5401
##
## Error measures:
##                       ME       RMSE        MAE MPE MAPE      MASE        ACF1
## Training set 0.002519487 0.04408992 0.02978897 Inf  Inf 0.6866667 0.007010676
```

```
##
## Forecasts:
##     Point Forecast       Lo 80      Hi 80       Lo 95      Hi 95
## 252    0.001245511 -0.05548446 0.05797548 -0.08551549 0.08800651
## 253    0.001245511 -0.05548807 0.05797910 -0.08552102 0.08801204
## 254    0.001245511 -0.05549169 0.05798271 -0.08552655 0.08801758
## 255    0.001245511 -0.05549531 0.05798633 -0.08553209 0.08802311
## 256    0.001245511 -0.05549893 0.05798995 -0.08553762 0.08802864
## 257    0.001245511 -0.05550254 0.05799357 -0.08554315 0.08803417
## 258    0.001245511 -0.05550616 0.05799718 -0.08554868 0.08803970
## 259    0.001245511 -0.05550978 0.05800080 -0.08555421 0.08804523
## 260    0.001245511 -0.05551339 0.05800441 -0.08555974 0.08805076
## 261    0.001245511 -0.05551701 0.05800803 -0.08556527 0.08805629
```

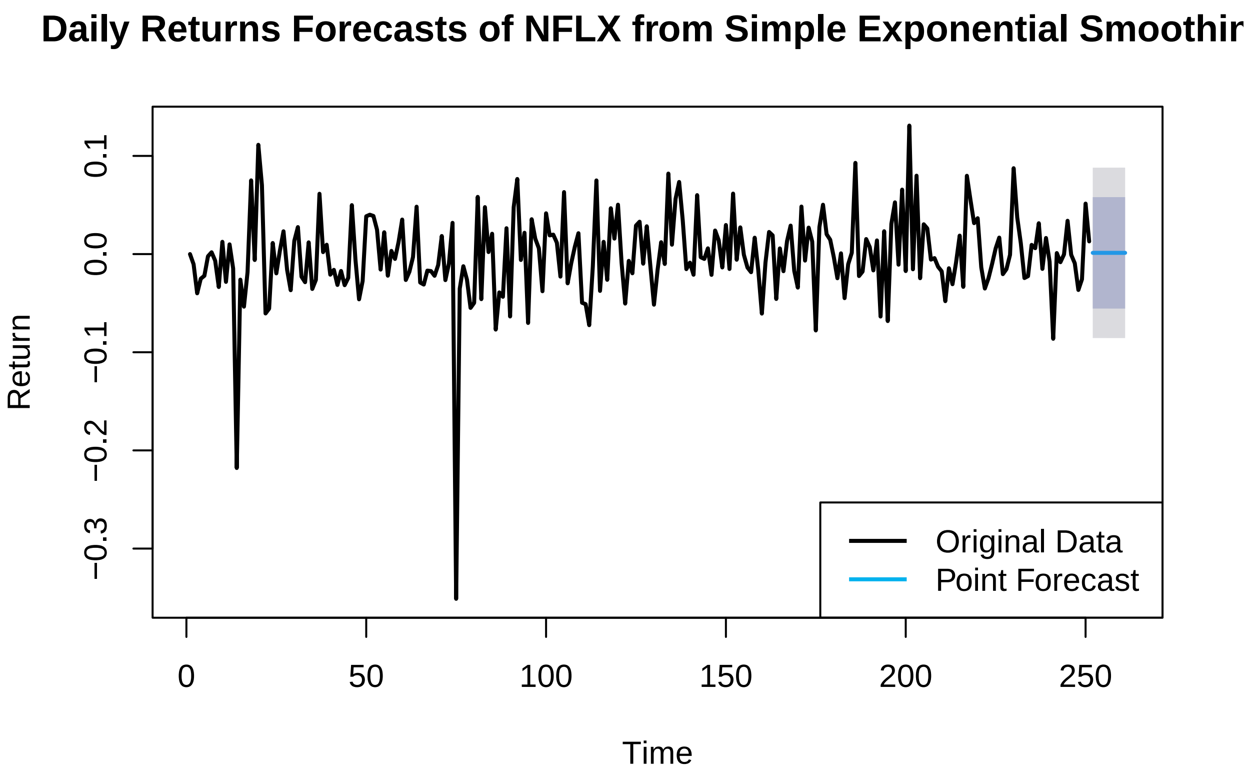
```r
plot(nflx.r$date, nflx.r$daily.returns,
     main = "Daily Returns on NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Return",
     type = 'l',
     lwd = 2.0,
     col = "blue")
lines(nflx.r$date,nflx.r.ses3$fitted, lwd = 2.0, col = "green3")

legend(x = "bottomright",
       legend = c("Original Data", "SES alpha = 0.0113 "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("blue", "green3"))
```



The estimated smoothing parameter $\alpha = 0.0113$ minimizes the sum of the squared forecast errors. This estimated value is unsurprising as the level of the time series is extremely stable (i.e., the mean of the series is not changing very much at all!). Therefore new observations are not given much weight because they don't appear to contribute much new information about the level of the series.

If we apply the plot command to the object generated by the **ses** function, we are able to plot out the point and interval forecasts. Note how the forecasts are flat!

```
plot(nflx.r.ses3,
     main = "Daily Returns Forecasts of NFLX from Simple Exponential Smoothing",
     xlab = "Time",
     ylab ="Return",
     lwd = 2.0)

legend(x = "bottomright",
       legend = c("Original Data", "Point Forecast "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue2"))
```



Now let's suppose we were to apply the simple exponential smoothing model to the price level of NFLX stock. The estimated smoothing parameter is $\alpha = 0.9999$. Compared to the daily returns, we can see that the level of the price series is changing dramatically as we move along the sample. Hence new observations contain lots of information about the level of the series and are thus given a great deal of weight!

```
nflx.p.ses1 <- ses(nflx.p$adjusted, h = 30, initial = "optimal")

summary(nflx.p.ses1)
```

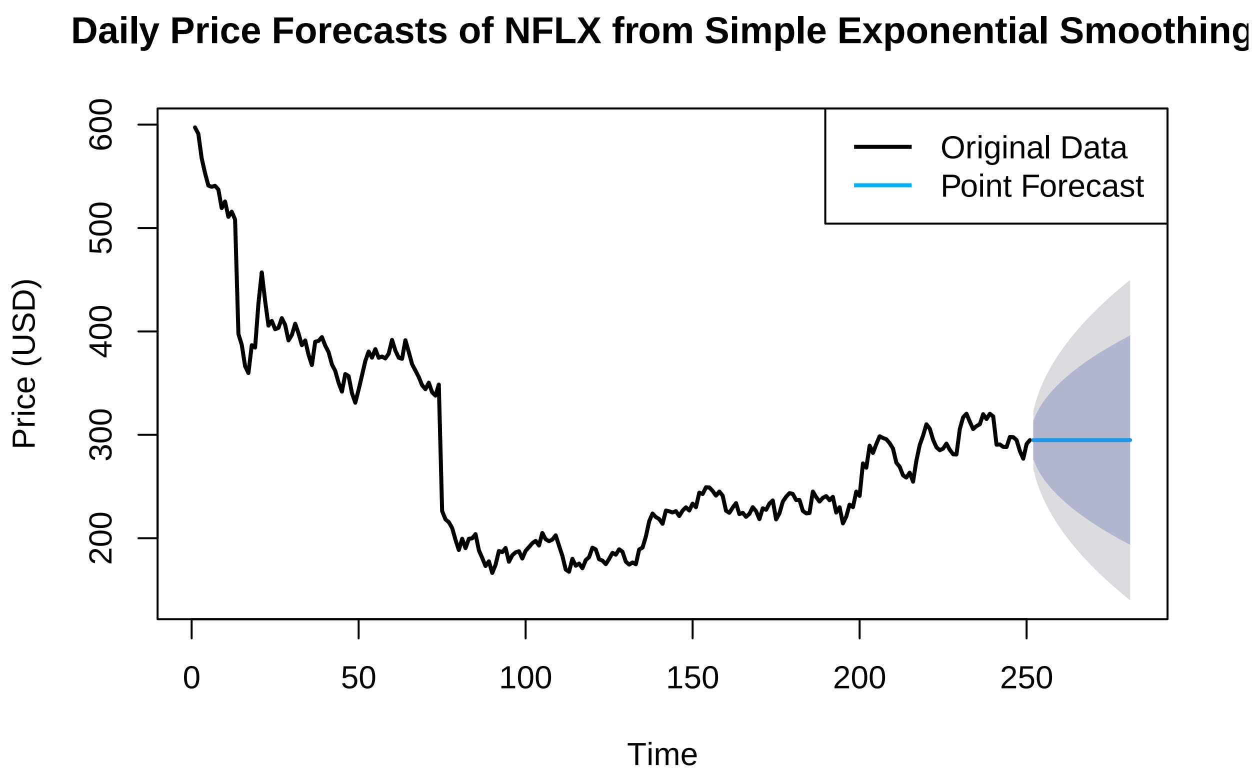
```
##
## Forecast method: Simple exponential smoothing
##
## Model Information:
```

```
## Simple exponential smoothing
##
## Call:
##  ses(y = nflx.p$adjusted, h = 30, initial = "optimal")
##
##   Smoothing parameters:
##     alpha = 0.9999
##
##   Initial states:
##     l = 597.4531
##
##   sigma:  14.4295
##
##      AIC     AICc      BIC
## 2730.858 2730.955 2741.434
##
## Error measures:
##                     ME     RMSE      MAE        MPE     MAPE      MASE
## Training set -1.205593 14.37194 8.516465 -0.3996204 3.090294 0.9960636
##                    ACF1
## Training set 0.05542689
##
## Forecasts:
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 252       294.8796 276.3874 313.3718 266.5982 323.1610
## 253       294.8796 268.7290 321.0302 254.8857 334.8735
## 254       294.8796 262.8523 326.9069 245.8981 343.8611
## 255       294.8796 257.8980 331.8612 238.3211 351.4381
## 256       294.8796 253.5331 336.2261 231.6456 358.1136
## 257       294.8796 249.5870 340.1723 225.6105 364.1488
## 258       294.8796 245.9581 343.8012 220.0606 369.6987
## 259       294.8796 242.5804 347.1789 214.8948 374.8644
## 260       294.8796 239.4080 350.3513 210.0430 379.7162
## 261       294.8796 236.4074 353.3518 205.4541 384.3051
## 262       294.8796 233.5535 356.2057 201.0894 388.6698
## 263       294.8796 230.8267 358.9326 196.9191 392.8402
## 264       294.8796 228.2112 361.5480 192.9191 396.8402
## 265       294.8796 225.6946 364.0647 189.0702 400.6890
## 266       294.8796 223.2663 366.4929 185.3566 404.4027
## 267       294.8796 220.9178 368.8415 181.7647 407.9945
## 268       294.8796 218.6415 371.1177 178.2835 411.4757
## 269       294.8796 216.4313 373.3280 174.9033 414.8560
## 270       294.8796 214.2816 375.4776 171.6157 418.1436
## 271       294.8796 212.1879 377.5714 168.4135 421.3458
## 272       294.8796 210.1458 379.6134 165.2904 424.4688
## 273       294.8796 208.1518 381.6074 162.2409 427.5184
## 274       294.8796 206.2027 383.5566 159.2599 430.4994
## 275       294.8796 204.2954 385.4638 156.3430 433.4162
## 276       294.8796 202.4275 387.3317 153.4863 436.2729
## 277       294.8796 200.5966 389.1626 150.6862 439.0730
## 278       294.8796 198.8006 390.9586 147.9394 441.8198
## 279       294.8796 197.0376 392.7217 145.2431 444.5162
## 280       294.8796 195.3057 394.4535 142.5945 447.1648
## 281       294.8796 193.6035 396.1558 139.9911 449.7681
```

```
plot(nflx.p$date, nflx.p$adjusted,
     main = "Daily Price of NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Price (USD)",
     type = 'l',
     lwd = 2.0,
     col = "darkgreen")
lines(nflx.p$date,nflx.p.ses1$fitted, lwd = 2.0, col = "red3")

legend(x = "topright",
       legend = c("Original Data", "SES alpha = 0.9999 "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("darkgreen", "red3"))
```

However, the simple exponential smoothing model is clearly not an appropriate forecasting model as it would be unreasonable to assume that the level of series would remain flat over a long time horizon:

```
plot(nflx.p.ses1,
     main = "Daily Price Forecasts of NFLX from Simple Exponential Smoothing",
     xlab = "Time",
     ylab ="Price (USD)",
     lwd = 2.0)

legend(x = "topright",
       legend = c("Original Data", "Point Forecast "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue2"))
```



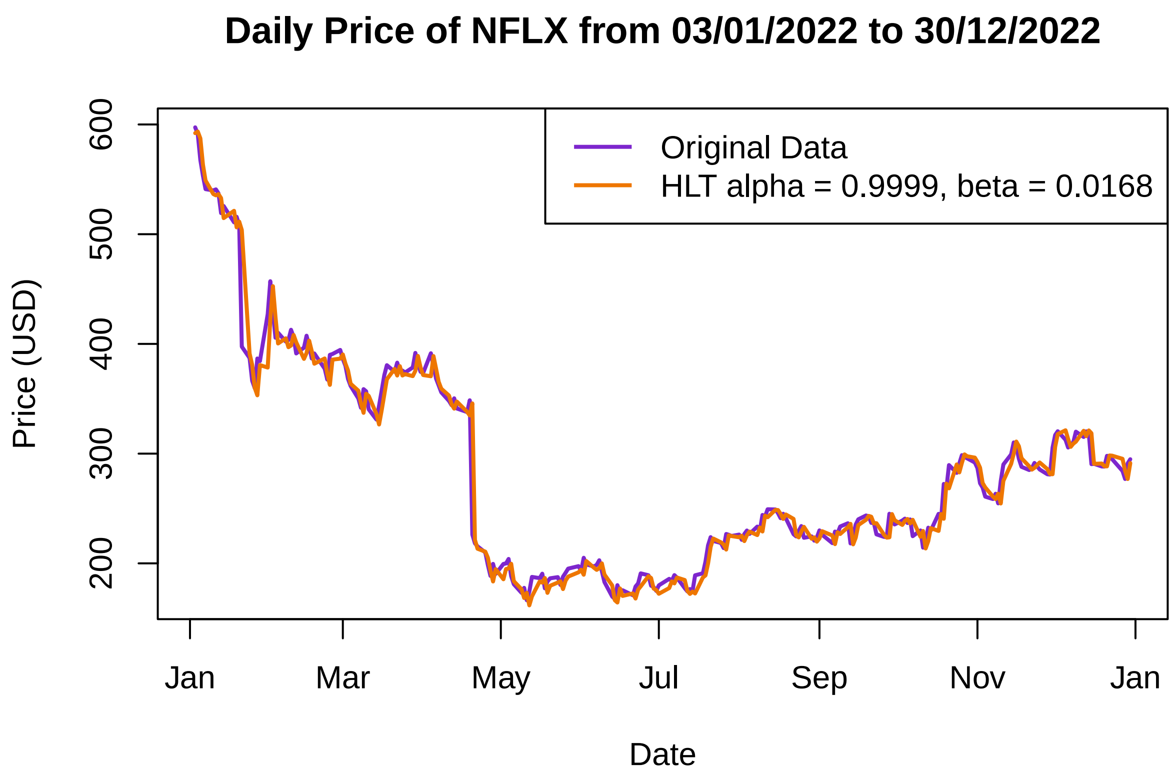
Let's try applying Holt's linear trend model to the price data. To generate the smoothed series we now use the **holt()** function:

```
nflx.p.hlt <- holt(nflx.p$adjusted, h = 30, initial = "optimal")

summary(nflx.p.hlt)
```

```
##
## Forecast method: Holt's method
##
## Model Information:
## Holt's method
##
## Call:
##  holt(y = nflx.p$adjusted, h = 30, initial = "optimal")
##
##   Smoothing parameters:
##     alpha = 0.9999
##     beta  = 0.0168
##
##   Initial states:
##     l = 596.0382
##     b = -3.9072
##
##   sigma:  14.4239
##
##      AIC     AICc      BIC
## 2732.639 2732.884 2750.266
##
## Error measures:
```

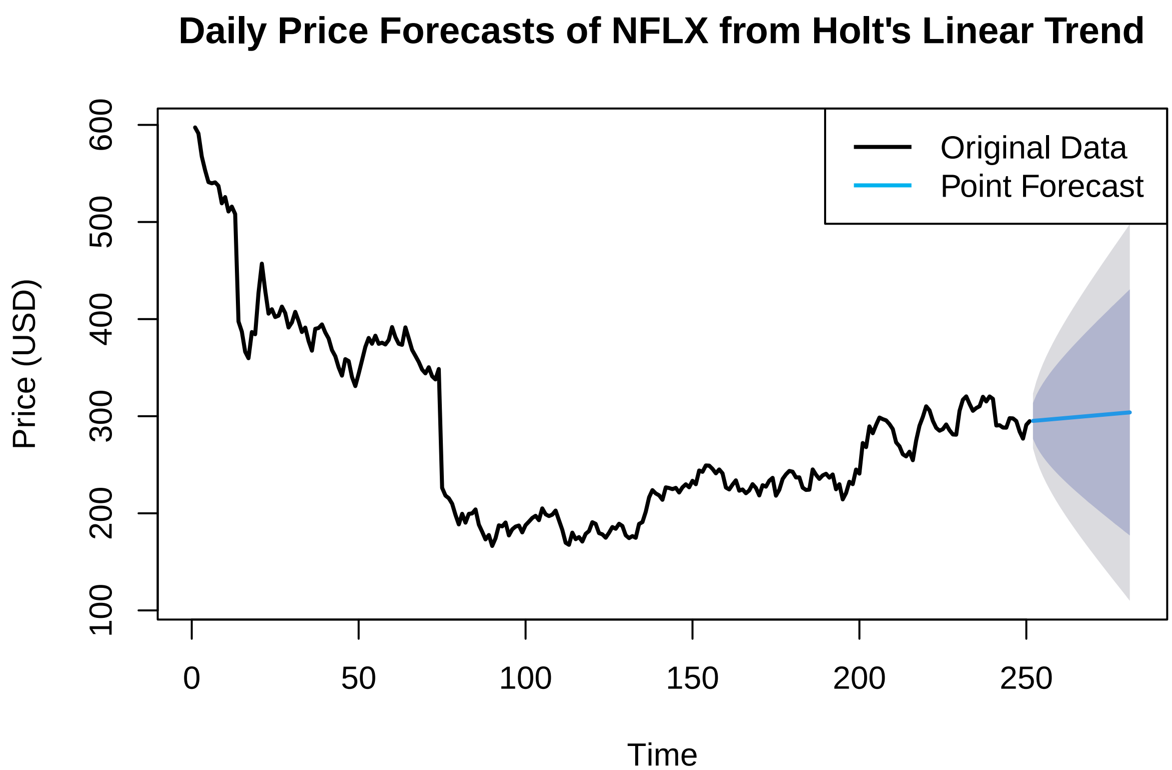
```
##                       ME     RMSE      MAE       MPE     MAPE      MASE
## Training set 0.9985284 14.30855 8.443147 0.4197628 3.084325 0.9874886
##                     ACF1
## Training set 0.03284402
##
## Forecasts:
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 252       295.1809 276.6958 313.6659 266.9104 323.4513
## 253       295.4821 269.1212 321.8429 255.1666 335.7975
## 254       295.7833 263.2278 328.3387 245.9940 345.5725
## 255       296.0845 258.1797 333.9892 238.1141 354.0548
## 256       296.3857 253.6559 339.1155 231.0361 361.7353
## 257       296.6869 249.4932 343.8805 224.5104 368.8633
## 258       296.9881 245.5957 348.3805 218.3902 375.5860
## 259       297.2893 241.9011 352.6774 212.5804 381.9981
## 260       297.5905 238.3669 356.8140 207.0159 388.1651
## 261       297.8917 234.9622 360.8212 201.6493 394.1341
## 262       298.1929 231.6638 364.7220 196.4454 399.9404
## 263       298.4941 228.4539 368.5342 191.3769 405.6112
## 264       298.7953 225.3187 372.2718 186.4226 411.1680
## 265       299.0965 222.2468 375.9462 181.5650 416.6279
## 266       299.3977 219.2290 379.5664 176.7903 422.0051
## 267       299.6989 216.2577 383.1400 172.0866 427.3111
## 268       300.0001 213.3266 386.6736 167.4444 432.5558
## 269       300.3013 210.4301 390.1724 162.8552 437.7473
## 270       300.6025 207.5638 393.6411 158.3121 442.8928
## 271       300.9037 204.7237 397.0837 153.8090 447.9984
## 272       301.2049 201.9062 400.5036 149.3406 453.0692
## 273       301.5061 199.1083 403.9038 144.9022 458.1099
## 274       301.8073 196.3275 407.2870 140.4899 463.1247
## 275       302.1085 193.5614 410.6556 136.1001 468.1169
## 276       302.4097 190.8079 414.0115 131.7295 473.0899
## 277       302.7109 188.0652 417.3566 127.3754 478.0464
## 278       303.0121 185.3316 420.6926 123.0352 482.9889
## 279       303.3133 182.6056 424.0210 118.7067 487.9199
## 280       303.6145 179.8858 427.3432 114.3878 492.8412
## 281       303.9157 177.1712 430.6602 110.0767 497.7547
```

```r
plot(nflx.p$date, nflx.p$adjusted,
     main = "Daily Price of NFLX from 03/01/2022 to 30/12/2022",
     xlab = "Date",
     ylab = "Price (USD)",
     type = 'l',
     lwd = 2.0,
     col = "purple3")
lines(nflx.p$date,nflx.p.hlt$fitted, lwd = 2.0, col = "darkorange2")

legend(x = "topright",
       legend = c("Original Data", "HLT alpha = 0.9999, beta = 0.0168"),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("purple3", "darkorange2"))
```



Here we can see that the point forecasts are now trending upwards over the forecast horizon $h = 30$:

```
plot(nflx.p.hlt,
     main = "Daily Price Forecasts of NFLX from Holt's Linear Trend",
     xlab = "Time",
     ylab ="Price (USD)",
     lwd = 2.0)

legend(x = "topright",
       legend = c("Original Data", "Point Forecast "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue2"))
```



To illustrate the exponential trend model, let's revisit the US GDP data from FRED. Again, we can download this directly using the **tq_get()** function.

```
us.gdp <- tq_get("GDP", get = "economic.data", from ="1960-01-01", to = "2022-12-01")
```

Then, using the **holt()** function, we can compute the smoothed series with exponential trend:

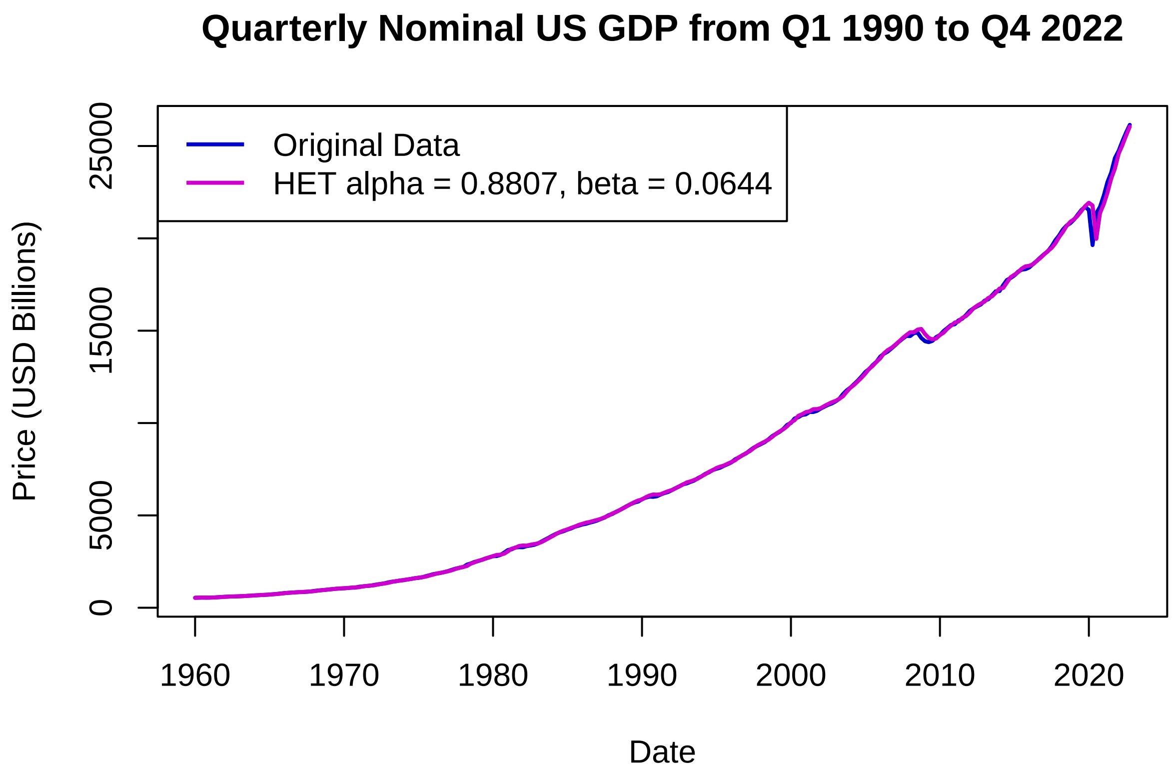
```
us.gdp.het <- holt(us.gdp$price, h = 30, exponential = TRUE)

summary(us.gdp.het)
```

```
##
## Forecast method: Holt's method with exponential trend
##
## Model Information:
## Holt's method with exponential trend
##
## Call:
##  holt(y = us.gdp$price, h = 30, exponential = TRUE)
##
##   Smoothing parameters:
##     alpha = 0.8864
##     beta  = 0.0462
##
##   Initial states:
##     l = 522.7116
##     b = 1.0167
##
##   sigma:  0.0118
##
##      AIC     AICc      BIC
```

```
## 3440.518 3440.762 3458.165
##
## Error measures:
##                     ME     RMSE      MAE        MPE      MAPE      MASE
## Training set -1.454583 187.7633 62.93335 -0.0398656 0.7194722 0.5121312
##                     ACF1
## Training set -0.01899033
##
## Forecasts:
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 253       26490.57 26073.59 26891.52 25875.77 27093.57
## 254       26851.70 26294.55 27413.29 25984.24 27718.58
## 255       27217.76 26530.67 27921.19 26193.60 28252.39
## 256       27588.81 26781.40 28413.81 26382.04 28871.47
## 257       27964.92 27020.64 28929.14 26589.66 29465.71
## 258       28346.15 27300.37 29433.92 26799.20 30063.17
## 259       28732.59 27547.68 29960.95 26933.78 30596.96
## 260       29124.29 27832.97 30496.62 27164.16 31208.11
## 261       29521.33 28120.60 31035.51 27367.18 31797.17
## 262       29923.78 28408.33 31561.76 27609.30 32429.11
## 263       30331.72 28679.51 32126.13 27817.48 33100.98
## 264       30745.22 28958.50 32678.11 28082.88 33721.54
## 265       31164.36 29263.05 33259.23 28315.66 34378.18
## 266       31589.21 29530.17 33802.51 28562.09 34986.04
## 267       32019.85 29828.87 34401.33 28750.03 35601.69
## 268       32456.37 30100.02 35022.28 28987.59 36377.32
## 269       32898.83 30403.24 35614.73 29248.42 37087.05
## 270       33347.33 30670.29 36246.74 29374.13 37902.48
## 271       33801.94 31023.88 36909.70 29577.70 38580.46
## 272       34262.75 31306.62 37537.13 29895.85 39254.20
## 273       34729.84 31606.74 38200.07 30105.96 40034.96
## 274       35203.30 31904.70 38905.92 30261.94 40820.22
## 275       35683.21 32190.18 39618.95 30480.07 41579.83
## 276       36169.67 32510.90 40282.76 30713.51 42363.67
## 277       36662.76 32814.11 41024.55 31023.42 43319.81
## 278       37162.57 33098.72 41713.78 31272.41 44174.68
## 279       37669.19 33403.24 42487.22 31496.96 45120.90
## 280       38182.72 33699.96 43240.20 31716.61 45981.74
## 281       38703.25 34033.85 43960.57 31813.07 47072.56
## 282       39230.87 34353.54 44807.73 32088.83 47936.89
```

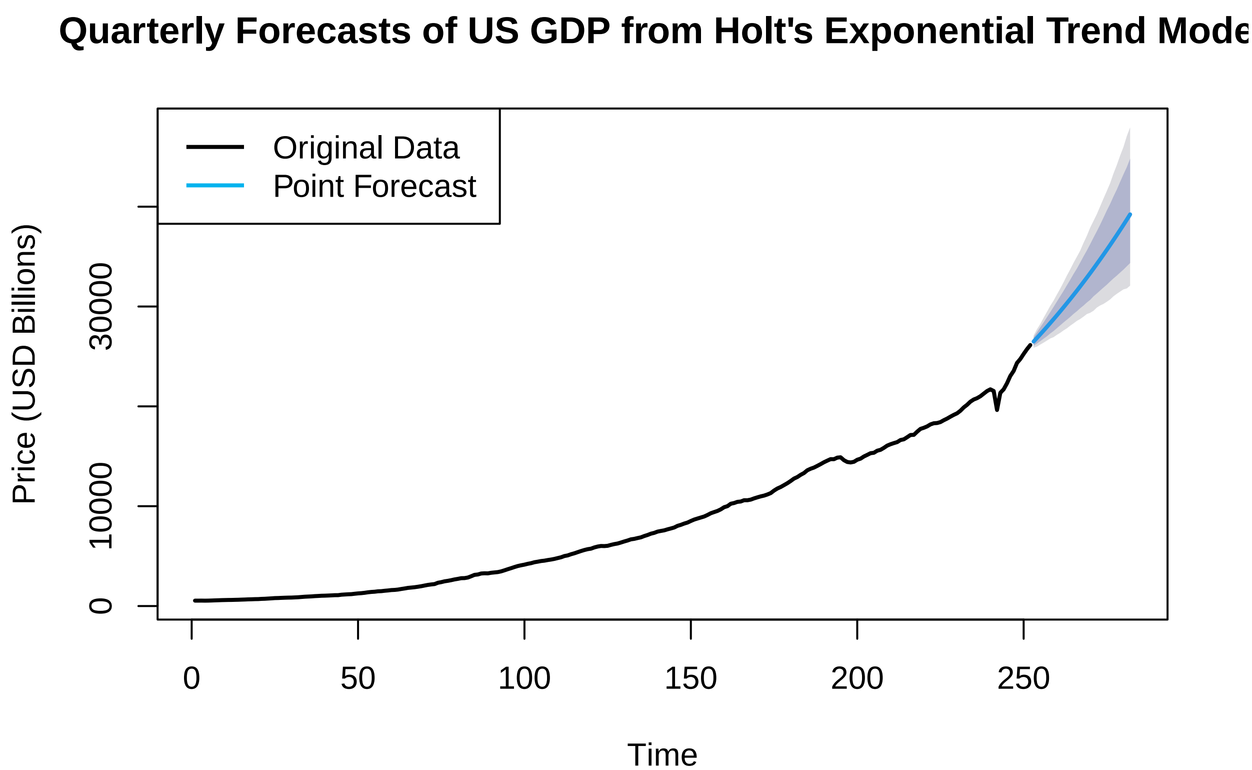
```r
plot(us.gdp$date, us.gdp$price,
     main = "Quarterly Nominal US GDP from Q1 1990 to Q4 2022",
     xlab = "Date",
     ylab = "Price (USD Billions)",
     type = 'l',
     lwd = 2.0,
     col = "blue3")
lines(us.gdp$date,us.gdp.het$fitted, lwd = 2.0, col = "magenta3")

legend(x = "topleft",
       legend = c("Original Data", "HET alpha = 0.8807, beta = 0.0644"),
       lty = c("solid", "solid"),
       lwd = 2.0,
```

```
        col = c("blue3", "magenta3"))
```



Then, we can plot the forecasts for a $h = 20$ step forecast horizon. Note that the forecasts reflect the multiplicative form of the smoothing model.

```
plot(us.gdp.het,
     main = "Quarterly Forecasts of US GDP from Holt's Exponential Trend Model",
     xlab = "Time",
     ylab ="Price (USD Billions)",
     lwd = 2.0)

legend(x = "topleft",
       legend = c("Original Data", "Point Forecast "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue2"))
```



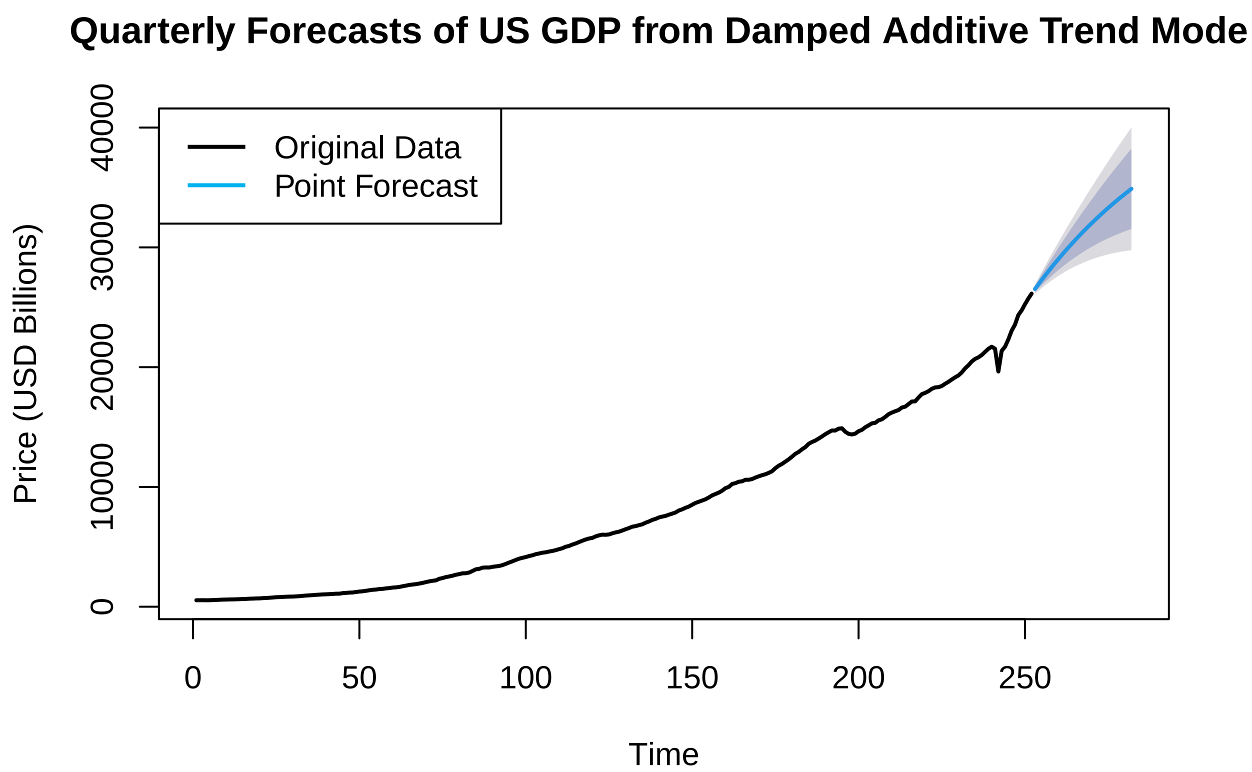
We can compare the above forecasts with those produced by a damped additive trend model:

```
us.gdp.hdt <- holt(us.gdp$price, h = 30, damped = TRUE)

summary(us.gdp.hdt)
```

```
##
## Forecast method: Damped Holt's method
##
## Model Information:
## Damped Holt's method
##
## Call:
##  holt(y = us.gdp$price, h = 30, damped = TRUE)
##
##   Smoothing parameters:
##     alpha = 0.8536
##     beta  = 0.1242
##     phi   = 0.98
##
##   Initial states:
##     l = 522.5493
##     b = 9.9462
##
##   sigma:  193.7921
##
##      AIC     AICc      BIC
## 4054.826 4055.169 4076.003
##
## Error measures:
```

```
##                    ME     RMSE      MAE       MPE      MAPE      MASE
## Training set 25.38016 191.8599 65.67163 0.3590141 0.7715894 0.5344145
##                     ACF1
## Training set -0.02776212
##
## Forecasts:
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 253       26522.49 26274.14 26770.85 26142.67 26902.32
## 254       26899.92 26552.99 27246.84 26369.34 27430.49
## 255       27269.79 26829.03 27710.54 26595.71 27943.86
## 256       27632.26 27098.60 28165.93 26816.09 28448.44
## 257       27987.49 27360.38 28614.60 27028.41 28946.57
## 258       28335.61 27613.89 29057.33 27231.84 29439.38
## 259       28676.77 27859.00 29494.54 27426.09 29927.45
## 260       29011.11 28095.71 29926.50 27611.13 30411.08
## 261       29338.75 28324.16 30353.35 27787.06 30890.45
## 262       29659.85 28544.48 30775.22 27954.04 31365.66
## 263       29974.52 28756.87 31192.17 28112.29 31836.76
## 264       30282.90 28961.53 31604.28 28262.03 32303.77
## 265       30585.12 29158.64 32011.59 28403.51 32766.72
## 266       30881.29 29348.42 32414.15 28536.97 33225.60
## 267       31171.53 29531.06 32812.01 28662.64 33680.42
## 268       31455.97 29706.75 33205.20 28780.76 34131.18
## 269       31734.72 29875.69 33593.76 28891.57 34577.87
## 270       32007.90 30038.06 33977.74 28995.29 35020.51
## 271       32275.61 30194.04 34357.18 29092.13 35459.10
## 272       32537.97 30343.82 34732.12 29182.31 35893.63
## 273       32795.08 30487.56 35102.61 29266.03 36324.13
## 274       33047.05 30625.43 35468.68 29343.49 36750.61
## 275       33293.98 30757.58 35830.38 29414.89 37173.07
## 276       33535.97 30884.18 36187.76 29480.41 37591.54
## 277       33773.12 31005.38 36540.87 29540.22 38006.03
## 278       34005.53 31121.31 36889.75 29594.50 38416.56
## 279       34233.29 31232.14 37234.45 29643.42 38823.16
## 280       34456.50 31337.98 37575.01 29687.14 39225.86
## 281       34675.24 31438.98 37911.49 29725.82 39624.66
## 282       34889.61 31535.27 38243.94 29759.60 40019.61
```

```
plot(us.gdp.hdt,
     main = "Quarterly Forecasts of US GDP from Damped Additive Trend Model",
     xlab = "Time",
     ylab ="Price (USD Billions)",
     lwd = 2.0)

legend(x = "topleft",
       legend = c("Original Data", "Point Forecast "),
       lty = c("solid", "solid"),
       lwd = 2.0,
       col = c("black", "deepskyblue2"))
```



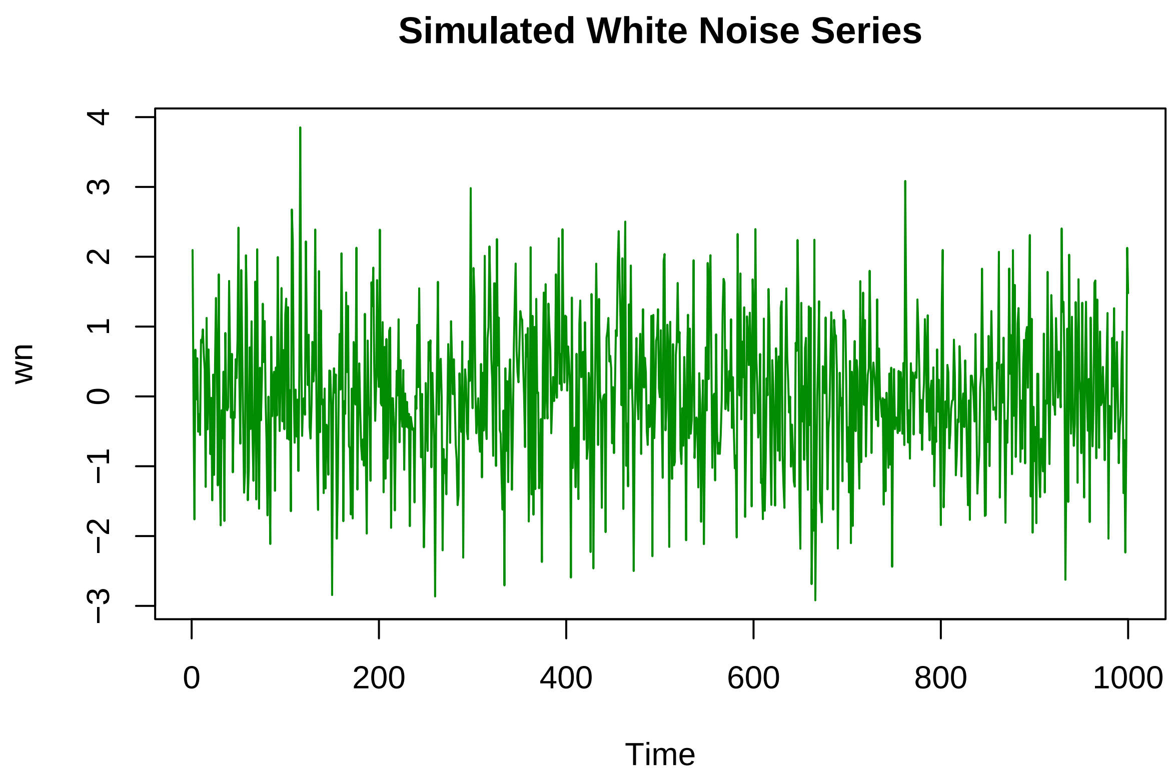
Covariance Stationary Time Series - Autoregressions

Let's first clear our workspace.

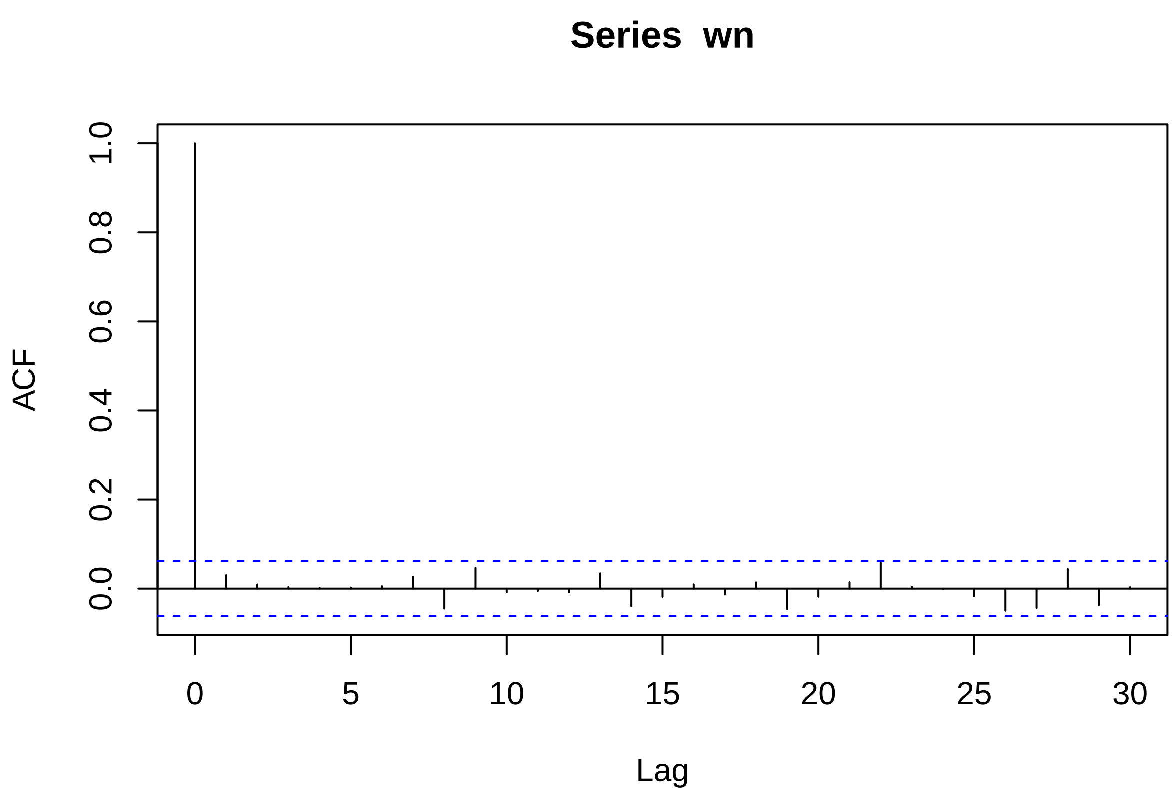
```
rm(list = ls())
```

The first thing we're going to do is generate a white noise series. These are a sequence of iid draws from a standard normal random variable. Then, using the **acf()** function, we are going to compute the sample autocorrelations associated with this series and store it in an object called **wn.acf**

```
T = 1000
wn <- rnorm(T, mean = 0, sd = 1)

plot(wn,
     main = "Simulated White Noise Series",
     xlab = "Time",
     ylab = "wn",
     type = 'l',
     lwd = 1.0,
     col = "green4")
```

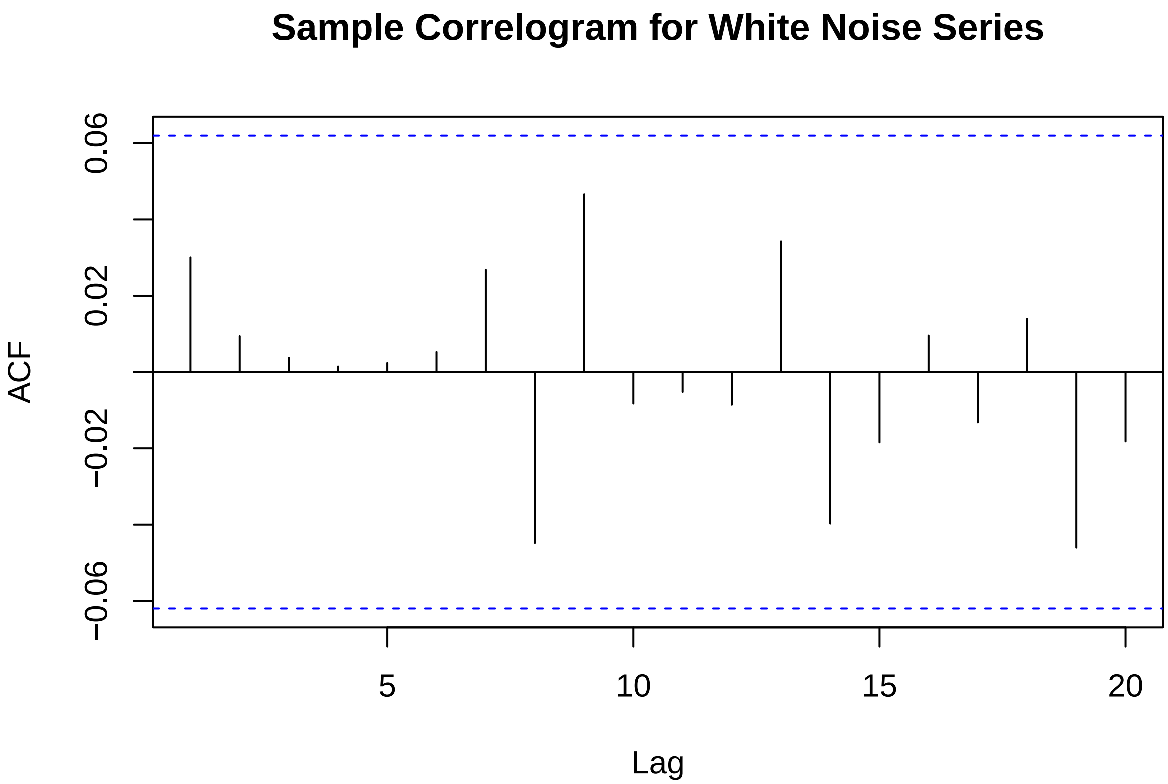



```r
wn.acf <- acf(wn, plot = TRUE)
```



Notice that by setting plot = TRUE in the argument of the function it will automatically generate a plot of the sample autocorrelations (i.e., the correlogram). However, this automatically generated correlogram will always include the 0-th order autocorrelation which is always equal to 1 (i.e., something is always perfectly correlated with itself). This can sometimes make the rest of the correlogram difficult to read, especially if the remaining sample autocorrelations are much smaller. So instead, we should set plot = FALSE, and then plot the correlogram directly using the **plot()** function:

```
plot(wn.acf[1:20], main = "Sample Correlogram for White Noise Series")
```



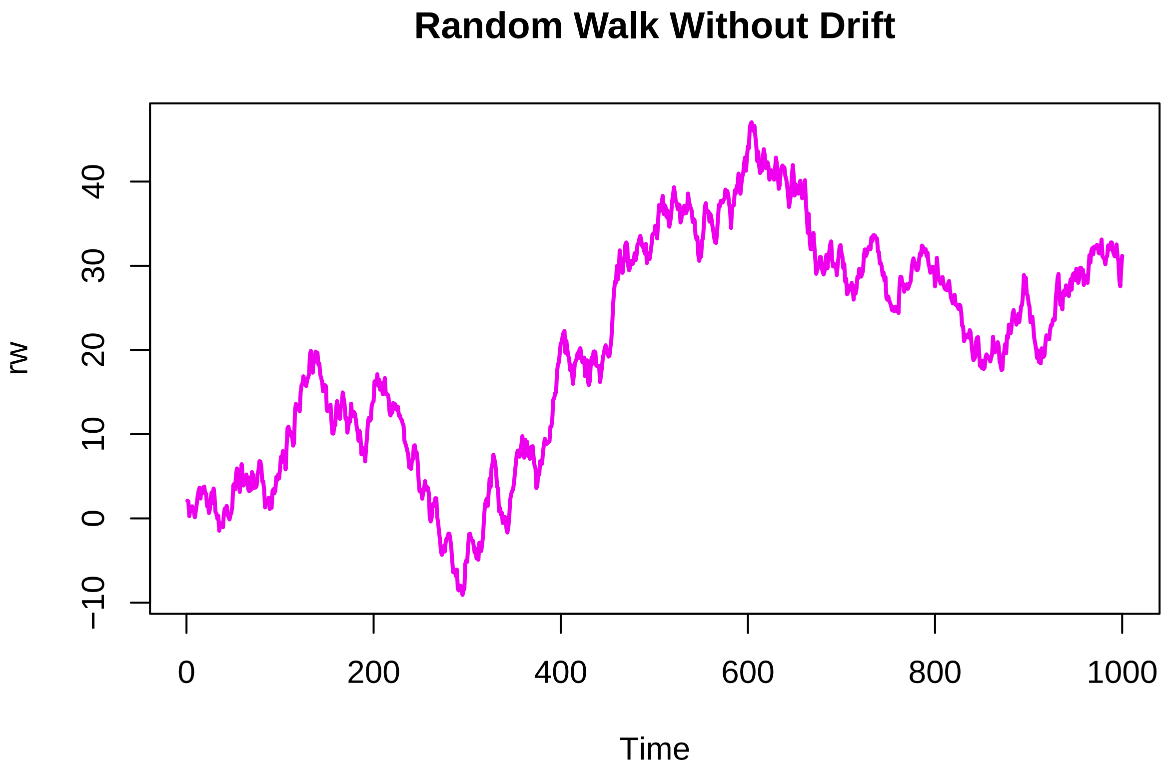
From the correlogram we can see that the sample autocorrelations are small in magnitude and almost none are statistically different from zero.

Now, we can use our white noise series to create a random walk. Recall that a random walk (without drift) is defined as:

$$Y_t = \sum_{i=1}^{t} \epsilon_i$$

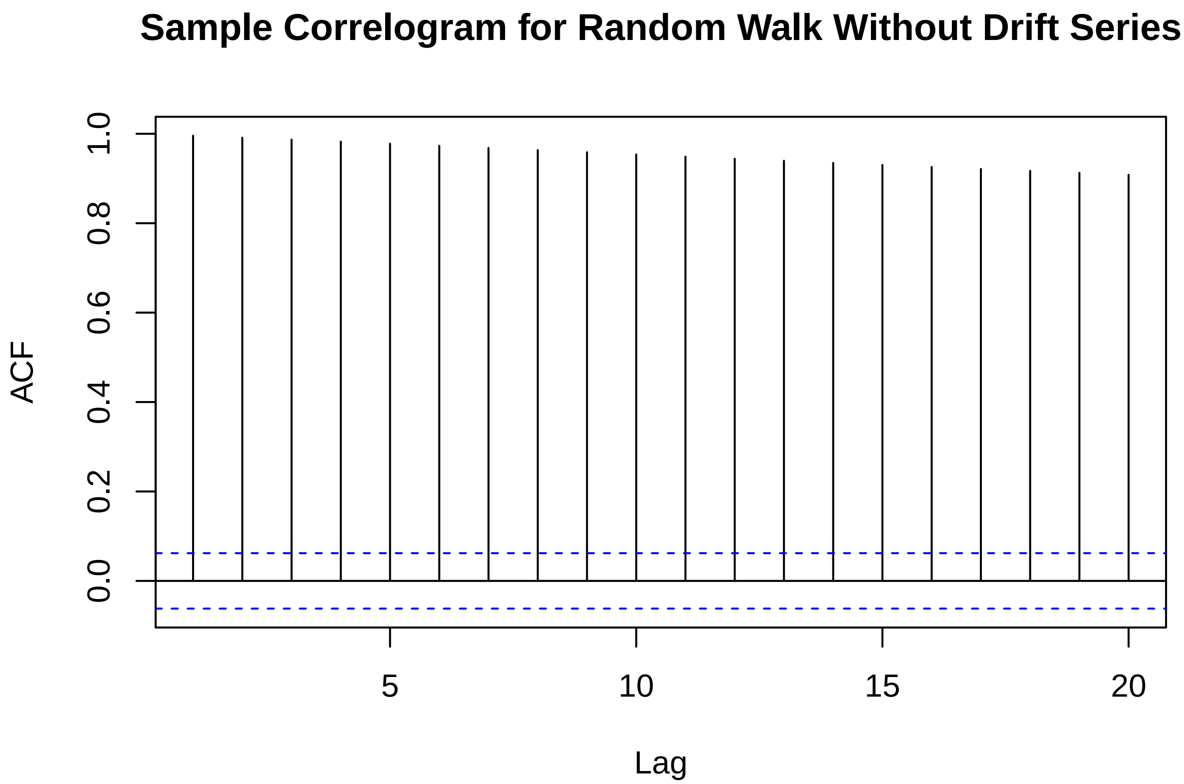
Thus, a random walk without drift can be constructed as the cumulative sum of white noise errors. So, using the **cumsum()** function, we compute:

```
rw <- cumsum(wn)

plot(rw,
     main = "Random Walk Without Drift",
     xlab = "Time",
     ylab = "rw",
     type = 'l',
     lwd = 2.0,
     col = "magenta2")
```



Now let's have a look at the sample correlogram for our random walk series. In contrast with the sample correlogram of the white noise series, all the sample autocorrelations are large in magnitude and statistically different from zero. Also notice that they are decaying very slowly.

```
rw.acf <- acf(rw, plot = FALSE)

plot(rw.acf[1:20], main = "Sample Correlogram for Random Walk Without Drift Series")
```



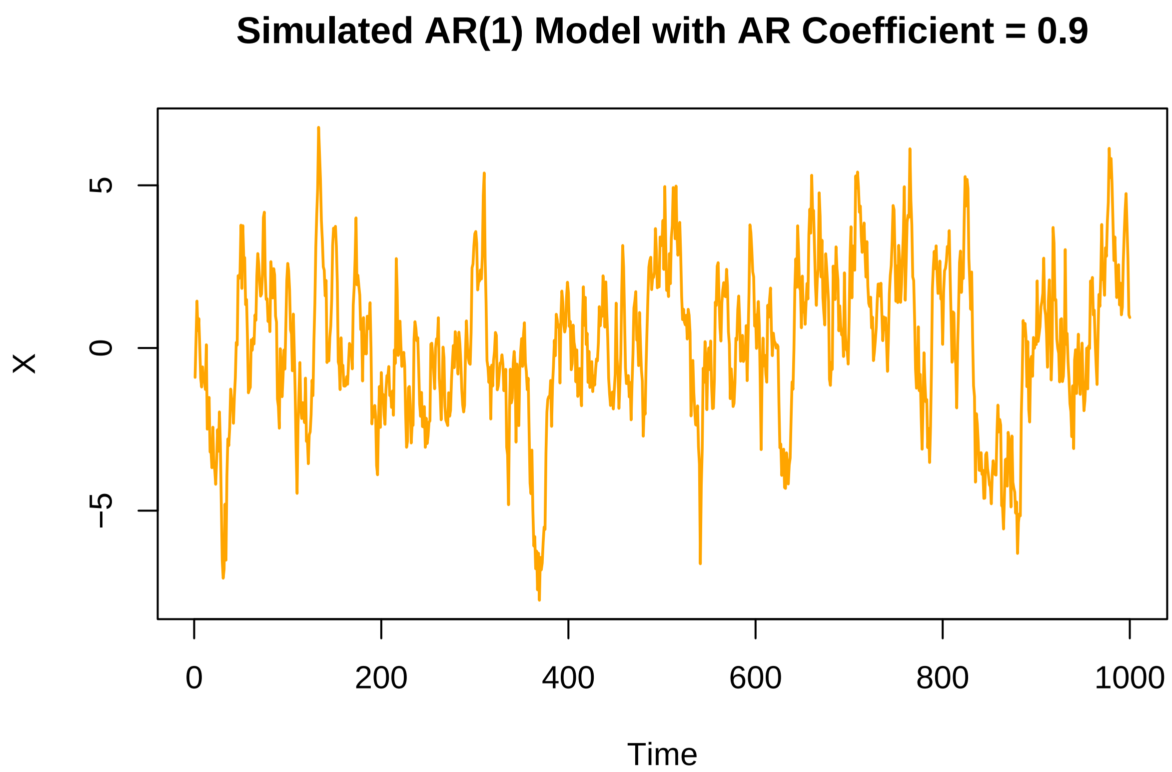
Let's now simulate a pair of AR(1) processes defined by

$$X_t = 0.9X_{t-1} + \epsilon_t \qquad \epsilon_t \sim i.i.d.N(0,1)$$

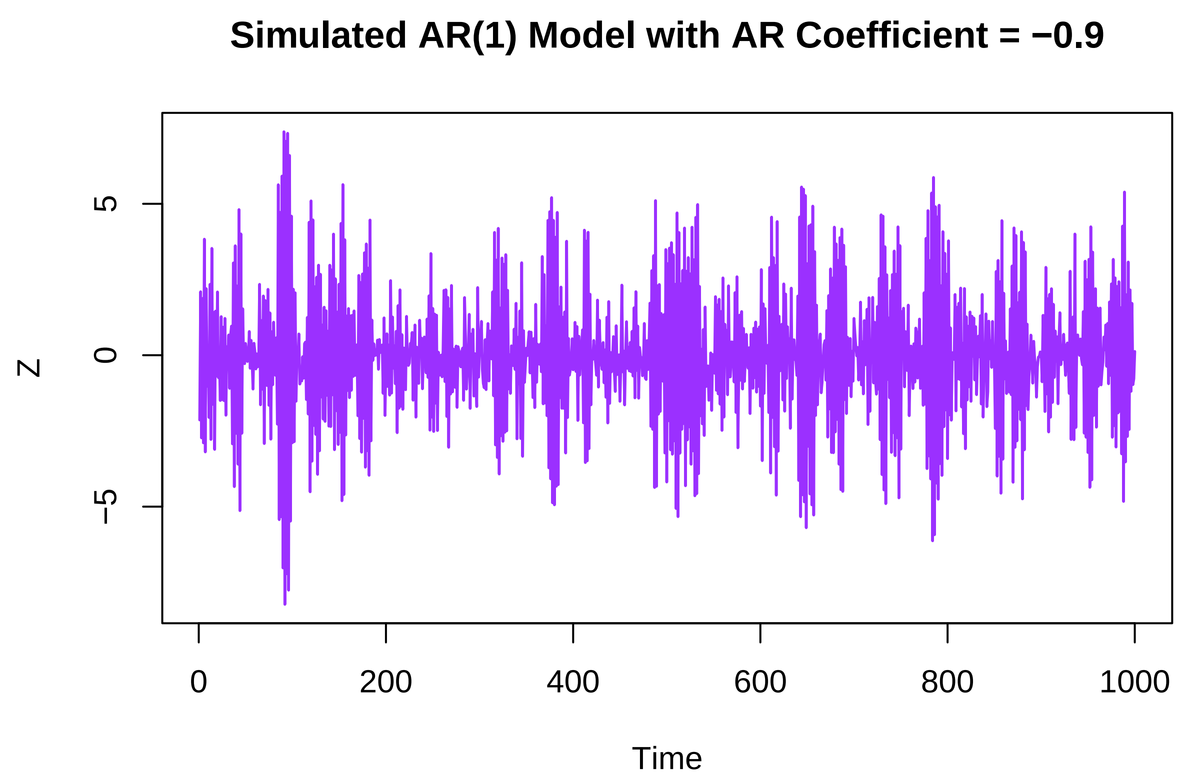
$$Z_t = -0.9Z_{t-1} + \upsilon_t \qquad \upsilon_t \sim i.i.d.N(0,1)$$

We can achieve this using the **arima.sim()** function.

```
x <- arima.sim(model=list(ar=c(0.9)),n = T)
z <- arima.sim(model=list(ar=c(-0.9)), n = T)

plot(x,
    main = "Simulated AR(1) Model with AR Coefficient = 0.9",
    xlab = "Time",
    ylab = "X",
    type = 'l',
    lwd = 1.4,
    col = "orange")
```

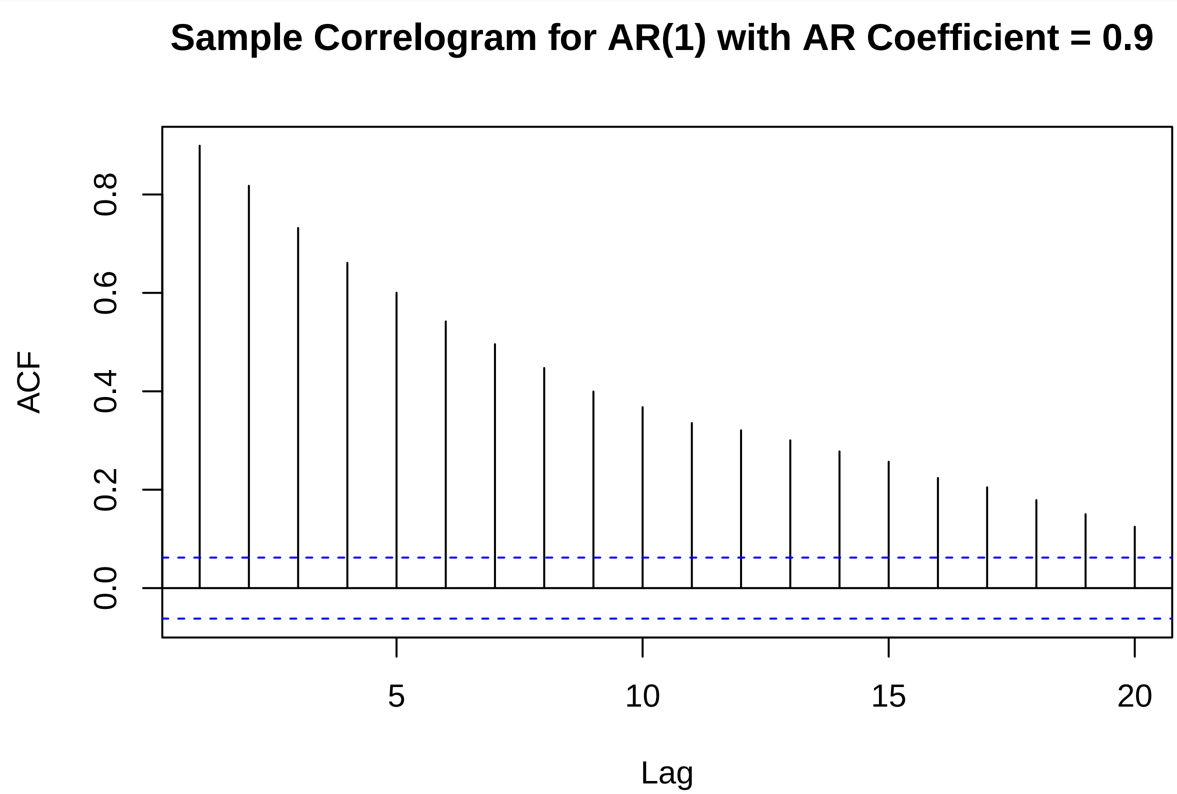



```
plot(z,
    main = "Simulated AR(1) Model with AR Coefficient = -0.9",
    xlab = "Time",
    ylab = "Z",
    type = 'l',
    lwd = 1.4,
    col = "purple1")
```

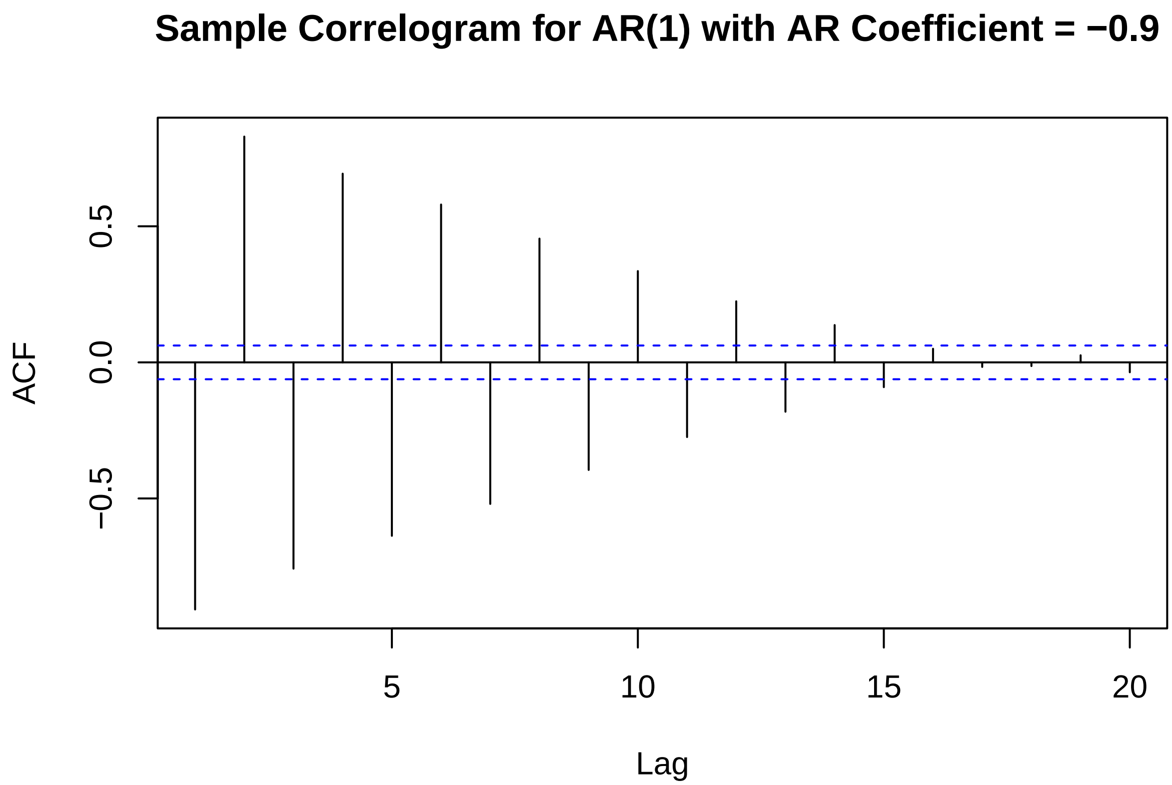


Now, let's have a look at their sample correlograms:

```
x.acf <- acf(x, plot = FALSE)
plot(x.acf[1:20], main = "Sample Correlogram for AR(1) with AR Coefficient = 0.9 ")
```



```r
z.acf <- acf(z, plot = FALSE)
plot(z.acf[1:20], main = "Sample Correlogram for AR(1) with AR Coefficient = -0.9 ")
```



To perform Box-Pierce and Ljung-Box tests on each of our series, we can use the **Box.test()** function. We can see that we would reject the null hypothesis that all the sample autocorrelations are jointly equal to zero for the random walk and AR(1) series.

```r
m = sqrt(T)

Box.test(wn, lag = m, type = "Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  wn
## X-squared = 24.848, df = 31.623, p-value = 0.7984
```

```r
Box.test(rw, lag = m, type = "Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  rw
## X-squared = 26682, df = 31.623, p-value < 2.2e-16
```

```r
Box.test(x, lag = m, type = "Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  x
```

```
## X-squared = 4481.3, df = 31.623, p-value < 2.2e-16
```

```r
Box.test(z, lag = m, type = "Box-Pierce")
```

```
##
##  Box-Pierce test
##
## data:  z
## X-squared = 4325.4, df = 31.623, p-value < 2.2e-16
```

```r
Box.test(wn, lag = m, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  wn
## X-squared = 25.383, df = 31.623, p-value = 0.7757
```

```r
Box.test(rw, lag = m, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  rw
## X-squared = 27150, df = 31.623, p-value < 2.2e-16
```

```r
Box.test(x, lag = m, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  x
## X-squared = 4514.8, df = 31.623, p-value < 2.2e-16
```

```r
Box.test(z, lag = m, type = "Ljung-Box")
```

```
##
##  Box-Ljung test
##
## data:  z
## X-squared = 4354.7, df = 31.623, p-value < 2.2e-16
```