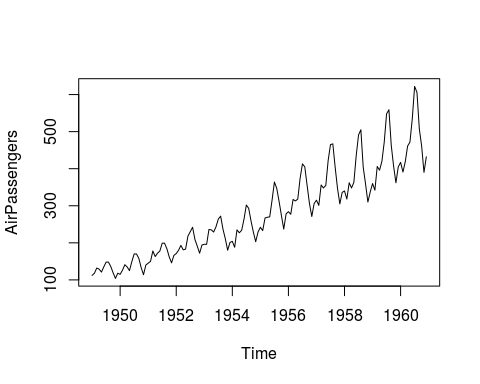
TimeSeries

Sergio Solano

9 de marzo de 2017

## Pruebas series de tiempo de Time Series Analysis and its applications

# Plot AirPassengers  
plot(AirPassengers)



# View the start and end dates of AirPassengers  
start(AirPassengers)

## [1] 1949 1

end(AirPassengers)

## [1] 1960 12

# Use time(), deltat(), frequency(), and cycle() with AirPassengers   
time(AirPassengers)

## Jan Feb Mar Apr May Jun Jul  
## 1949 1949.000 1949.083 1949.167 1949.250 1949.333 1949.417 1949.500  
## 1950 1950.000 1950.083 1950.167 1950.250 1950.333 1950.417 1950.500  
## 1951 1951.000 1951.083 1951.167 1951.250 1951.333 1951.417 1951.500  
## 1952 1952.000 1952.083 1952.167 1952.250 1952.333 1952.417 1952.500  
## 1953 1953.000 1953.083 1953.167 1953.250 1953.333 1953.417 1953.500  
## 1954 1954.000 1954.083 1954.167 1954.250 1954.333 1954.417 1954.500  
## 1955 1955.000 1955.083 1955.167 1955.250 1955.333 1955.417 1955.500  
## 1956 1956.000 1956.083 1956.167 1956.250 1956.333 1956.417 1956.500  
## 1957 1957.000 1957.083 1957.167 1957.250 1957.333 1957.417 1957.500  
## 1958 1958.000 1958.083 1958.167 1958.250 1958.333 1958.417 1958.500  
## 1959 1959.000 1959.083 1959.167 1959.250 1959.333 1959.417 1959.500  
## 1960 1960.000 1960.083 1960.167 1960.250 1960.333 1960.417 1960.500  
## Aug Sep Oct Nov Dec  
## 1949 1949.583 1949.667 1949.750 1949.833 1949.917  
## 1950 1950.583 1950.667 1950.750 1950.833 1950.917  
## 1951 1951.583 1951.667 1951.750 1951.833 1951.917  
## 1952 1952.583 1952.667 1952.750 1952.833 1952.917  
## 1953 1953.583 1953.667 1953.750 1953.833 1953.917  
## 1954 1954.583 1954.667 1954.750 1954.833 1954.917  
## 1955 1955.583 1955.667 1955.750 1955.833 1955.917  
## 1956 1956.583 1956.667 1956.750 1956.833 1956.917  
## 1957 1957.583 1957.667 1957.750 1957.833 1957.917  
## 1958 1958.583 1958.667 1958.750 1958.833 1958.917  
## 1959 1959.583 1959.667 1959.750 1959.833 1959.917  
## 1960 1960.583 1960.667 1960.750 1960.833 1960.917

deltat(AirPassengers)

## [1] 0.08333333

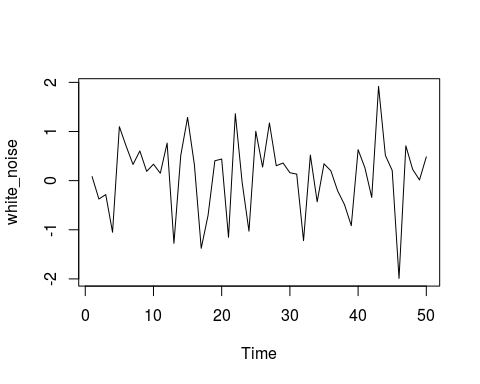
frequency(AirPassengers)

## [1] 12

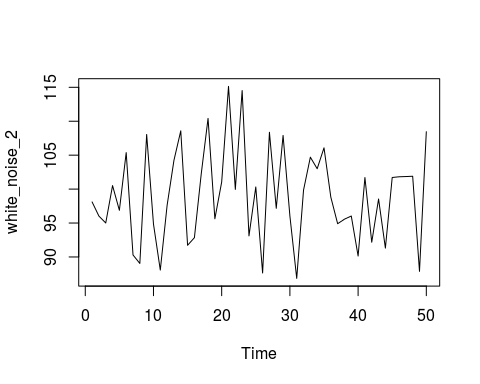
cycle(AirPassengers)

## Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec  
## 1949 1 2 3 4 5 6 7 8 9 10 11 12  
## 1950 1 2 3 4 5 6 7 8 9 10 11 12  
## 1951 1 2 3 4 5 6 7 8 9 10 11 12  
## 1952 1 2 3 4 5 6 7 8 9 10 11 12  
## 1953 1 2 3 4 5 6 7 8 9 10 11 12  
## 1954 1 2 3 4 5 6 7 8 9 10 11 12  
## 1955 1 2 3 4 5 6 7 8 9 10 11 12  
## 1956 1 2 3 4 5 6 7 8 9 10 11 12  
## 1957 1 2 3 4 5 6 7 8 9 10 11 12  
## 1958 1 2 3 4 5 6 7 8 9 10 11 12  
## 1959 1 2 3 4 5 6 7 8 9 10 11 12  
## 1960 1 2 3 4 5 6 7 8 9 10 11 12

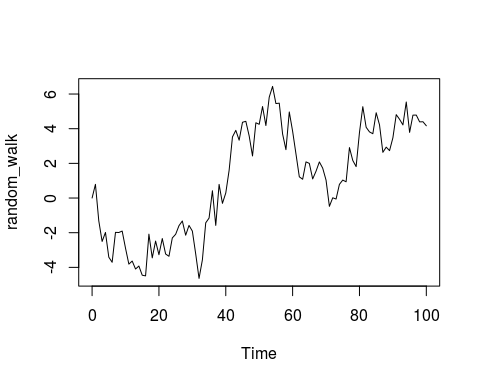
# Simulate a WN model with list(order = c(0, 0, 0))  
white\_noise <- arima.sim(model = list(order = c(0,0,0)), n = 50)  
  
# Plot your white\_noise data  
  
ts.plot(white\_noise)



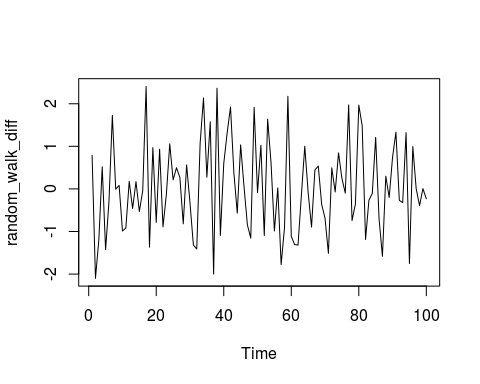
# Simulate from the WN model with: mean = 100, sd = 10  
white\_noise\_2 <- arima.sim(model = list(order = c(0,0,0)), n = 50, mean = 100, sd = 10)  
  
# Plot your white\_noise\_2 data  
ts.plot(white\_noise\_2)

 Los cambios en una serie de tiempo de Random Walk siguen un comportamiento de White noise.

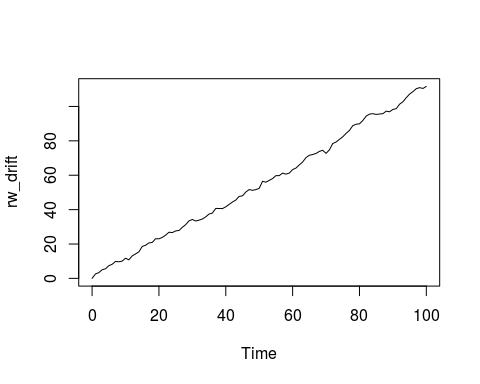
# Generate a RW model using arima.sim  
random\_walk <- arima.sim(model = list(order = c(0, 1, 0)) , n = 100)  
  
# Plot random\_walk  
ts.plot (random\_walk)



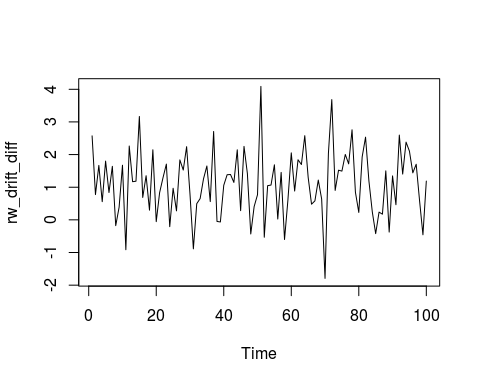
# Calculate the first difference series  
random\_walk\_diff <- diff(random\_walk)   
  
# Plot random\_walk\_diff  
  
ts.plot(random\_walk\_diff)



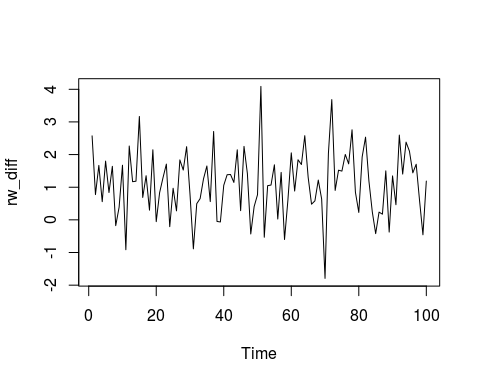
# RANDOM WALK WITH DRIFT  
  
# Generate a RW model with a drift uing arima.sim  
rw\_drift <- arima.sim(model = list(order = c(0, 1, 0)), n = 100, mean = 1)  
  
# Plot rw\_drift  
ts.plot(rw\_drift)



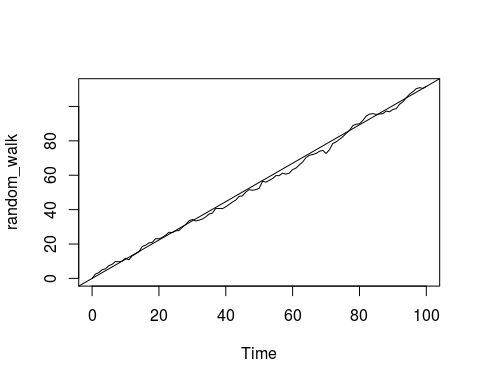
# Calculate the first difference series  
rw\_drift\_diff <- diff(rw\_drift)   
  
# Plot rw\_drift\_diff  
ts.plot(rw\_drift\_diff)



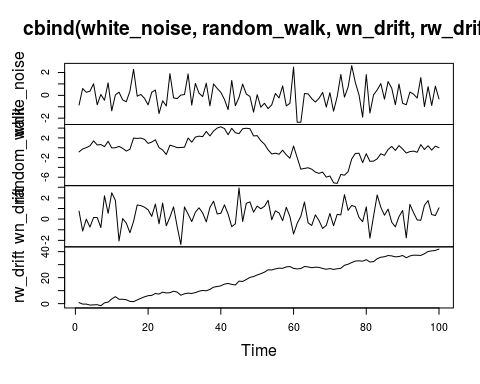
random\_walk <- rw\_drift  
  
# Difference your random\_walk data  
rw\_diff <- diff(random\_walk)  
  
  
# Plot rw\_diff  
plot.ts(rw\_diff)



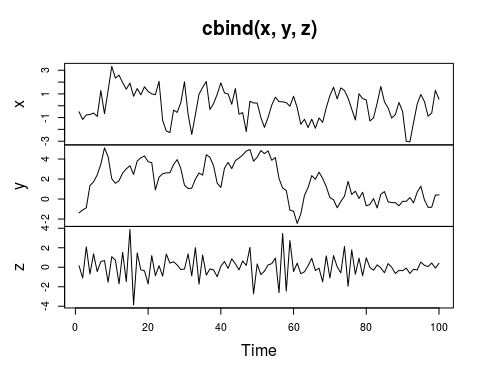
# Now fit the WN model to the differenced data  
model\_wn <- arima(rw\_diff,order = c(0, 0, 0))  
  
# Store the value of the estimated time trend (intercept)  
int\_wn <- model\_wn$coef  
  
# Plot the original random\_walk data  
ts.plot(random\_walk)  
  
# Use abline(0, ...) to add time trend to the figure  
abline(0,int\_wn)



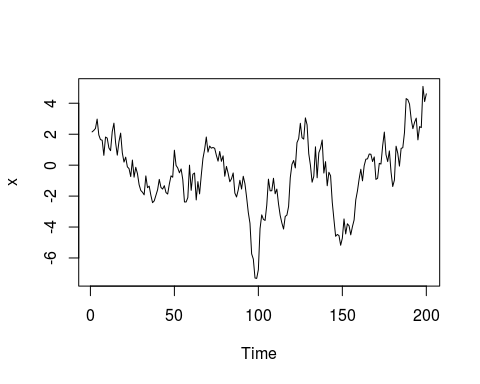
# RANDOM WALK CON Y SIN DRIFT (Estationarity)  
#   
# The white noise (WN) and random walk (RW) models are very closely related. However, only the RW is always non-stationary, both with and without a drift term. This is a simulation exercise to highlight the differences.  
#   
# Recall that if we start with a mean zero WN process and compute its running or cumulative sum, the result is a RW process. The cumsum() function will make this transformation for you. Similarly, if we create a WN process, but change its mean from zero, and then compute its cumulative sum, the result is a RW process with a drift.  
  
# Use arima.sim() to generate WN data  
white\_noise <- arima.sim(model = list(order = c(0, 0, 0)), n = 100)   
   
# Use cumsum() to convert your WN data to RW  
random\_walk <- cumsum(white\_noise)  
   
# Use arima.sim() to generate WN drift data  
wn\_drift <- arima.sim(model = list(order = c(0, 0, 0)),mean=0.4, n = 100)  
   
# Use cumsum() to convert your WN drift data to RW  
rw\_drift <- cumsum(wn\_drift)  
  
# Plot all four data objects  
plot.ts(cbind(white\_noise, random\_walk, wn\_drift, rw\_drift))



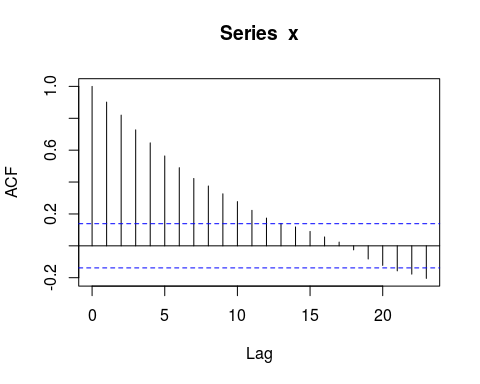
#   
# # Generate means from eu\_percentreturns  
# colMeans(eu\_percentreturns)  
#   
# # Use apply to calculate sample variance from eu\_percentreturns  
# apply(eu\_percentreturns, MARGIN = 2, FUN = var)  
#   
# # Use apply to calculate standard deviation from eu\_percentreturns  
# apply(eu\_percentreturns, MARGIN = 2, FUN = sd)  
#   
# # Display a histogram of percent returns for each index  
# par(mfrow = c(2,2))  
# apply(eu\_percentreturns, MARGIN = 2, FUN = hist, main = "", xlab = "Percentage Return")  
#   
# # Display normal quantile plots of percent returns for each index  
# par(mfrow = c(2,2))  
# apply(eu\_percentreturns, MARGIN = 2, FUN = qqnorm, main = "")  
# qqline(eu\_percentreturns)  
  
# pairs(eu\_stocks)  
  
  
# MODELOS AUTOREGRESIVOS:   
  
# Simulate an AR model with 0.5 slope  
x <- arima.sim(model = list(ar = 0.5), n = 100)  
  
# Simulate an AR model with 0.9 slope  
y <- arima.sim(model = list(ar = 0.9), n = 100)  
  
# Simulate an AR model with -0.75 slope  
z <- arima.sim(model = list(ar = -0.75), n = 100)  
  
# Plot your simulated data  
plot.ts(cbind(x, y, z))



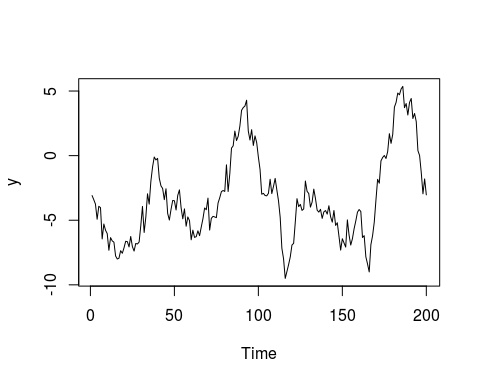
###### COMPARAR MODELOS AR con RandomWalks   
  
# Simulate and plot AR model with slope 0.9   
x <- arima.sim(model = list(ar = 0.9), n = 200)  
ts.plot(x)



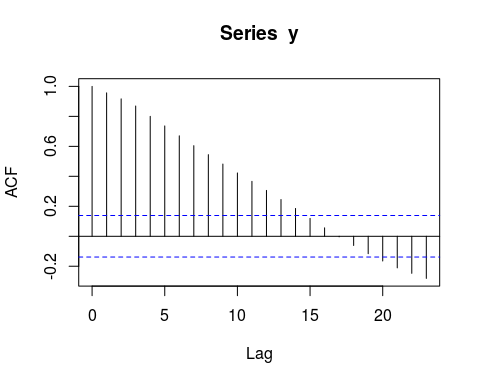
acf(x)



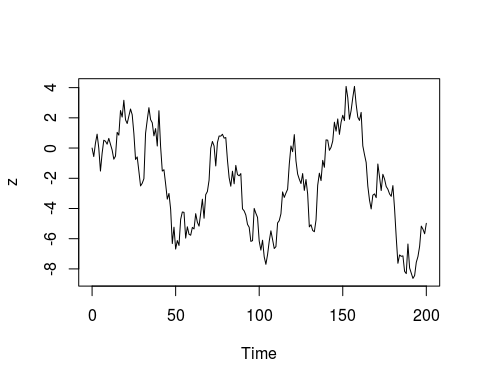
# Simulate and plot AR model with slope 0.98  
y <- arima.sim(model = list(ar = 0.98), n = 200)  
ts.plot(y)



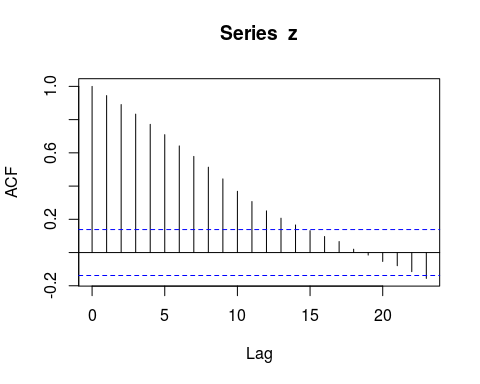
acf(y)



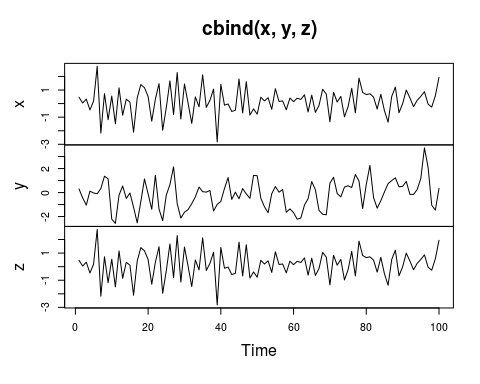
# Simulate and plot RW model  
z <- arima.sim(model = list(order = c(0, 1, 0)), n = 200)  
ts.plot(z)



acf(z)



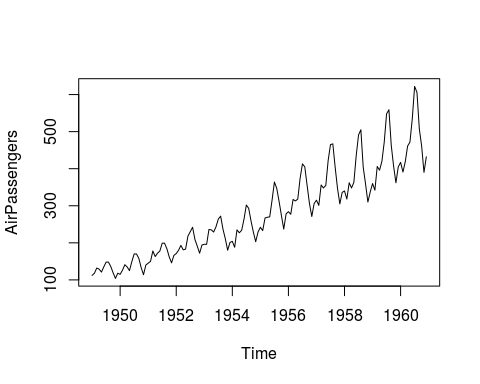
# # AJUSTAR UN AR  
# # Fit an AR model to Nile  
# AR\_fit <-arima(Nile, order = c(1,0,0))  
# print(AR\_fit)  
#   
# # Use predict() to make a 1-step forecast  
# predict\_AR <- predict(AR\_fit)  
#   
# # Obtain the 1-step forecast using $pred[1]  
# predict\_AR$pred[1]  
#   
# # Use predict to make 1-step through 10-step forecasts  
# predict(AR\_fit, n.ahead = 10)  
#   
# # Run to plot the Nile series plus the forecast and 95% prediction intervals  
# ts.plot(Nile, xlim = c(1871, 1980))  
# AR\_forecast <- predict(AR\_fit, n.ahead = 10)$pred  
# AR\_forecast\_se <- predict(AR\_fit, n.ahead = 10)$se  
# points(AR\_forecast, type = "l", col = 2)  
# points(AR\_forecast - 2\*AR\_forecast\_se, type = "l", col = 2, lty = 2)  
# points(AR\_forecast + 2\*AR\_forecast\_se, type = "l", col = 2, lty = 2)  
  
# # AJUSTAR UN MA  
# Generate MA model with slope 0.5  
x <- arima.sim(model = list(ma = 0.5), n = 100)  
  
# Generate MA model with slope 0.9  
y <- x <- arima.sim(model = list(ma = 0.9), n = 100)  
  
# Generate MA model with slope -0.5  
z <- x <- arima.sim(model = list(ma = -0.5), n = 100)  
  
# Plot all three models together  
plot.ts(cbind(x, y, z))



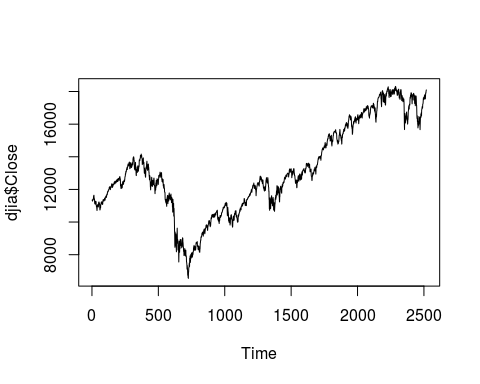
# # # Ajustando un MA   
#   
# # Make a 1-step forecast based on MA  
# predict\_MA <- predict(MA, n.ahead = 1)  
#   
# # Obtain the 1-step forecast using $pred[1]  
# predict\_MA$pred[1]   
#   
# # Make a 1-step through 10-step forecast based on MA  
# predict(MA, n.ahead = 10)  
#   
# # Plot the Nile series plus the forecast and 95% prediction intervals  
# ts.plot(Nile, xlim = c(1871, 1980))  
# MA\_forecasts <- predict(MA, n.ahead = 10)$pred  
# MA\_forecast\_se <- predict(MA, n.ahead = 10)$se  
# points(MA\_forecasts, type = "l", col = 2)  
# points(MA\_forecasts - 2\*MA\_forecast\_se, type = "l", col = 2, lty = 2)  
# points(MA\_forecasts + 2\*MA\_forecast\_se, type = "l", col = 2, lty = 2)

## Pruebas series de tiempo de Time Series Analysis and its applications course

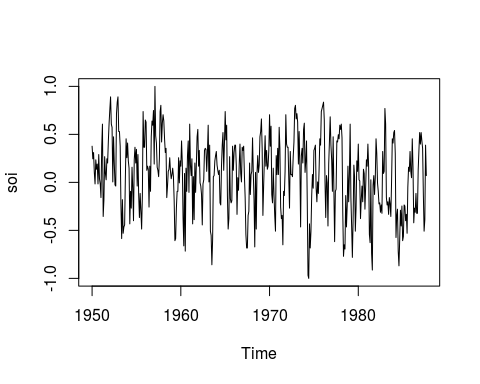
# # R Arima base   
  
# View a detailed description of AirPassengers  
help(AirPassengers)  
  
# Plot AirPassengers  
ts.plot(AirPassengers)



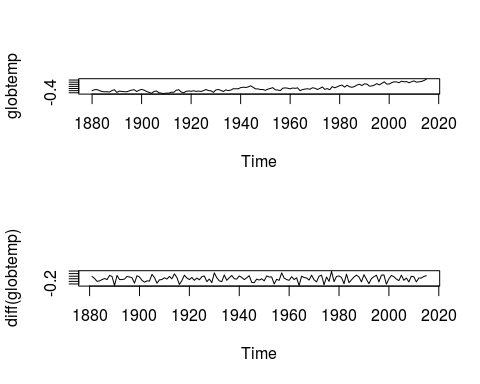
# Plot the DJIA daily closings  
ts.plot(djia$Close)



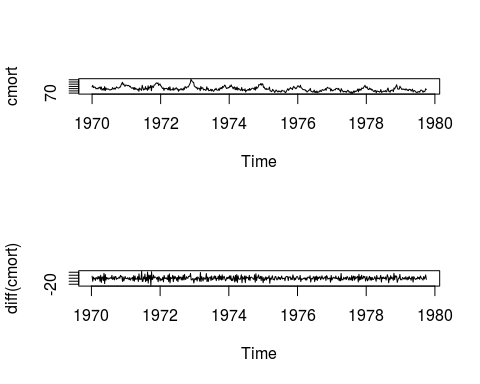
# Plot the Southern Oscillation Index  
plot(soi)



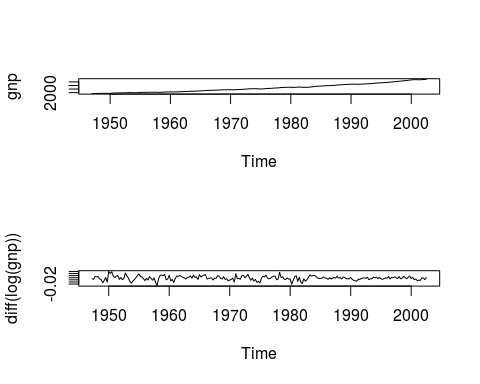
# Plot globtemp and detrended globtemp  
par(mfrow = c(2,1))  
plot(globtemp)   
plot(diff(globtemp))



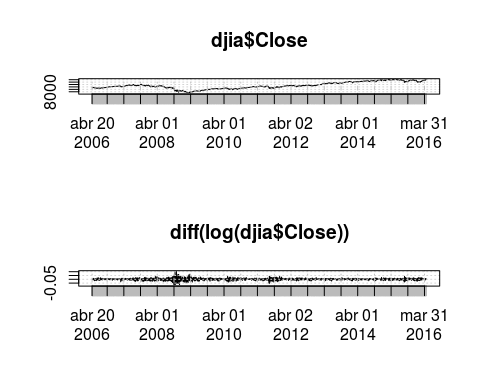
# Plot cmort and detrended cmort  
par(mfrow = c(2,1))  
plot(cmort)  
plot(diff(cmort))



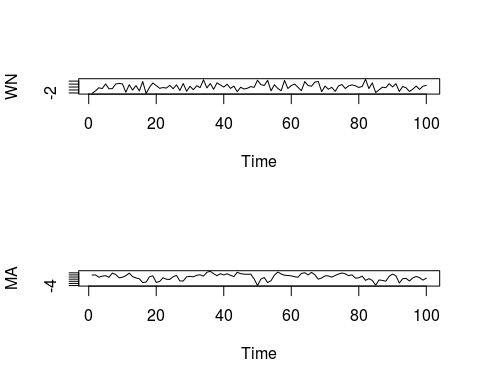
# Plot GNP series (gnp) and its growth rate  
par(mfrow = c(2,1))  
plot(gnp)  
plot(diff(log(gnp)))



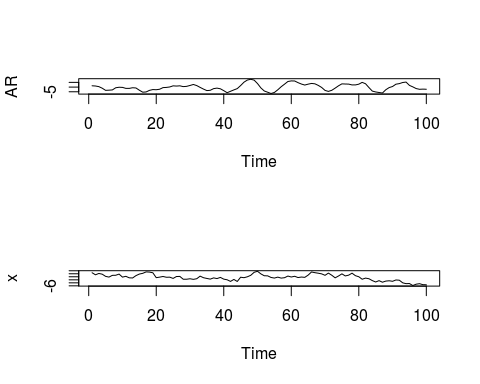
# Plot DJIA closings (djia$Close) and its returns  
par(mfrow = c(2,1))  
plot(djia$Close)  
plot(diff(log(djia$Close)))



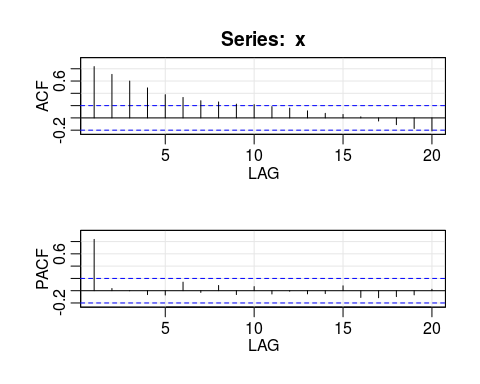
##SIMULAR MA & AR  
  
# Generate and plot white noise  
WN <- arima.sim(model=list(order=c(0,0,0)),n=100)  
plot(WN)  
# Generate and plot an MA(1) with parameter .9   
MA <- arima.sim(model=list(order=c(0,0,1), ma = 0.9),n=100)  
plot(MA)



# Generate and plot an AR(2) with parameters 1.5 and -.75  
AR <- arima.sim(model=list(order=c(2,0,0), ar =c(1.5,-0.75)),n=100)  
plot(AR)  
  
# # SIMULAR Y AJUSTAR AR  
  
  
# Generate 100 observations from the AR(1) model  
x <- arima.sim(model = list(order = c(1, 0, 0), ar = .9), n = 100)   
  
# Plot the generated data   
plot(x)



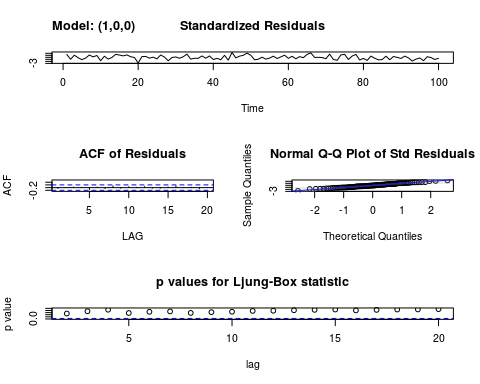
# Plot the sample P/ACF pair  
acf2(x)



## ACF PACF  
## [1,] 0.84 0.84  
## [2,] 0.71 0.04  
## [3,] 0.60 0.00  
## [4,] 0.49 -0.06  
## [5,] 0.38 -0.07  
## [6,] 0.33 0.14  
## [7,] 0.28 -0.03  
## [8,] 0.26 0.08  
## [9,] 0.22 -0.06  
## [10,] 0.22 0.07  
## [11,] 0.18 -0.05  
## [12,] 0.16 -0.01  
## [13,] 0.11 -0.05  
## [14,] 0.07 -0.05  
## [15,] 0.06 0.08  
## [16,] 0.02 -0.11  
## [17,] -0.05 -0.11  
## [18,] -0.11 -0.10  
## [19,] -0.17 -0.07  
## [20,] -0.21 0.02

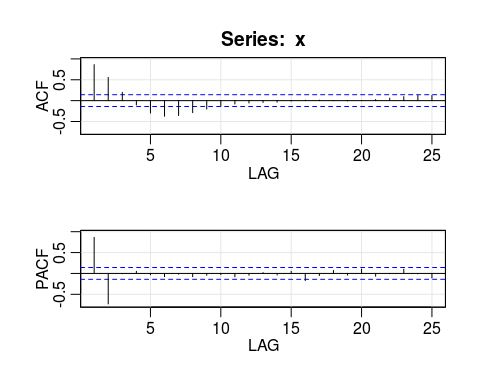
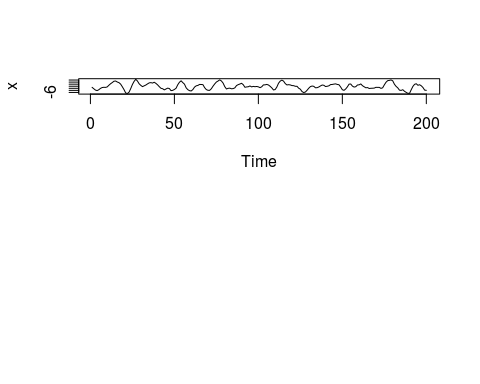
# Fit an AR(1) to the data and examine the -table  
sarima(x, 1, 0, 0)

## initial value 0.731272   
## iter 2 value 0.031859  
## iter 3 value 0.030165  
## iter 4 value 0.029885  
## iter 5 value 0.028436  
## iter 6 value 0.027868  
## iter 7 value 0.027858  
## iter 8 value 0.027858  
## iter 9 value 0.027857  
## iter 10 value 0.027856  
## iter 11 value 0.027855  
## iter 12 value 0.027855  
## iter 13 value 0.027855  
## iter 14 value 0.027855  
## iter 15 value 0.027855  
## iter 16 value 0.027855  
## iter 17 value 0.027855  
## iter 17 value 0.027855  
## iter 17 value 0.027855  
## final value 0.027855   
## converged  
## initial value 0.044746   
## iter 2 value 0.044287  
## iter 3 value 0.043306  
## iter 4 value 0.043159  
## iter 5 value 0.043012  
## iter 6 value 0.042841  
## iter 7 value 0.042785  
## iter 8 value 0.042632  
## iter 9 value 0.042627  
## iter 10 value 0.042624  
## iter 11 value 0.042612  
## iter 12 value 0.042612  
## iter 13 value 0.042612  
## iter 14 value 0.042612  
## iter 15 value 0.042611  
## iter 15 value 0.042611  
## iter 15 value 0.042611  
## final value 0.042611   
## converged



## $fit  
##   
## Call:  
## stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,   
## Q), period = S), xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc,   
## REPORT = 1, reltol = tol))  
##   
## Coefficients:  
## ar1 xmean  
## 0.8922 -0.5344  
## s.e. 0.0488 0.8915  
##   
## sigma^2 estimated as 1.072: log likelihood = -146.15, aic = 298.31  
##   
## $degrees\_of\_freedom  
## [1] 98  
##   
## $ttable  
## Estimate SE t.value p.value  
## ar1 0.8922 0.0488 18.2954 0.0000  
## xmean -0.5344 0.8915 -0.5994 0.5503  
##   
## $AIC  
## [1] 1.109323  
##   
## $AICc  
## [1] 1.131823  
##   
## $BIC  
## [1] 0.1614259

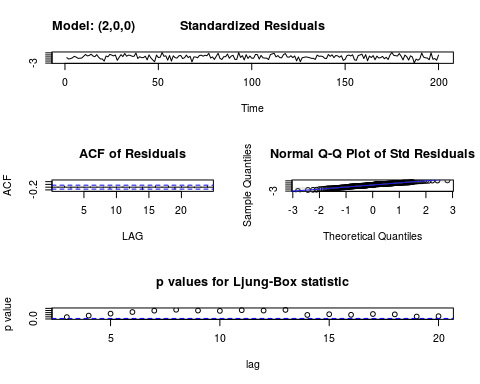
# # aJUSTAR MA(2)  
  
# astsa is preloaded  
x <- arima.sim(model = list(order = c(2, 0, 0), ar = c(1.5, -.75)), n = 200)  
# Plot x  
plot(x)  
  
# Plot the sample P/ACF of x  
acf2(x)



## ACF PACF  
## [1,] 0.86 0.86  
## [2,] 0.56 -0.73  
## [3,] 0.20 -0.01  
## [4,] -0.10 0.05  
## [5,] -0.30 -0.03  
## [6,] -0.37 -0.08  
## [7,] -0.36 -0.02  
## [8,] -0.29 -0.08  
## [9,] -0.20 -0.04  
## [10,] -0.13 -0.03  
## [11,] -0.08 -0.08  
## [12,] -0.06 -0.04  
## [13,] -0.05 0.02  
## [14,] -0.04 -0.04  
## [15,] -0.01 0.05  
## [16,] 0.01 -0.17  
## [17,] 0.01 -0.05  
## [18,] 0.00 0.07  
## [19,] -0.01 -0.04  
## [20,] 0.00 0.10  
## [21,] 0.03 -0.07  
## [22,] 0.06 -0.01  
## [23,] 0.10 0.10  
## [24,] 0.12 -0.01  
## [25,] 0.12 -0.11

# Fit an AR(2) to the data and examine the t-table  
sarima(x,2,0,0)

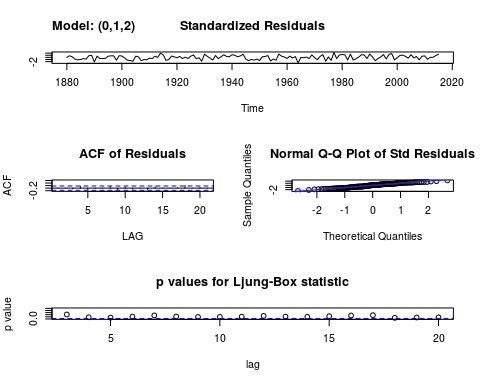
## initial value 1.068757   
## iter 2 value 0.931937  
## iter 3 value 0.503686  
## iter 4 value 0.250712  
## iter 5 value 0.096340  
## iter 6 value -0.038984  
## iter 7 value -0.041718  
## iter 8 value -0.042048  
## iter 9 value -0.042055  
## iter 10 value -0.042069  
## iter 11 value -0.042074  
## iter 12 value -0.042074  
## iter 13 value -0.042075  
## iter 14 value -0.042075  
## iter 14 value -0.042075  
## iter 14 value -0.042075  
## final value -0.042075   
## converged  
## initial value -0.036528   
## iter 2 value -0.036541  
## iter 3 value -0.036550  
## iter 4 value -0.036551  
## iter 5 value -0.036551  
## iter 6 value -0.036551  
## iter 7 value -0.036551  
## iter 7 value -0.036551  
## iter 7 value -0.036551  
## final value -0.036551   
## converged



## $fit  
##   
## Call:  
## stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,   
## Q), period = S), xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc,   
## REPORT = 1, reltol = tol))  
##   
## Coefficients:  
## ar1 ar2 xmean  
## 1.5143 -0.7493 -0.1392  
## s.e. 0.0462 0.0463 0.2879  
##   
## sigma^2 estimated as 0.9155: log likelihood = -276.48, aic = 560.95  
##   
## $degrees\_of\_freedom  
## [1] 197  
##   
## $ttable  
## Estimate SE t.value p.value  
## ar1 1.5143 0.0462 32.7926 0.0000  
## ar2 -0.7493 0.0463 -16.2000 0.0000  
## xmean -0.1392 0.2879 -0.4837 0.6291  
##   
## $AIC  
## [1] 0.9417349  
##   
## $AICc  
## [1] 0.9527605  
##   
## $BIC  
## [1] -0.008790341

# # # SIMULANDO ARIMAS   
  
# Fit an ARIMA(0,1,2) to globtemp and check the fit  
sarima(globtemp, 0, 1, 2)

## initial value -2.220513   
## iter 2 value -2.294887  
## iter 3 value -2.307682  
## iter 4 value -2.309170  
## iter 5 value -2.310360  
## iter 6 value -2.311251  
## iter 7 value -2.311636  
## iter 8 value -2.311648  
## iter 9 value -2.311649  
## iter 9 value -2.311649  
## iter 9 value -2.311649  
## final value -2.311649   
## converged  
## initial value -2.310187   
## iter 2 value -2.310197  
## iter 3 value -2.310199  
## iter 4 value -2.310201  
## iter 5 value -2.310202  
## iter 5 value -2.310202  
## iter 5 value -2.310202  
## final value -2.310202   
## converged

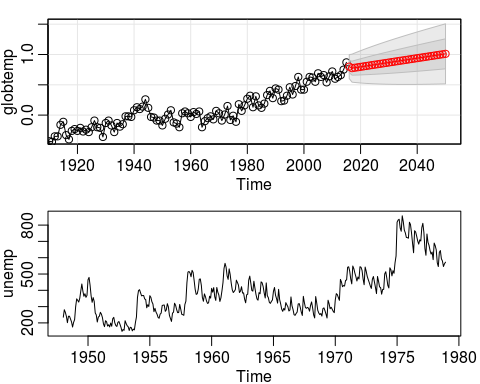


## $fit  
##   
## Call:  
## stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,   
## Q), period = S), xreg = constant, optim.control = list(trace = trc, REPORT = 1,   
## reltol = tol))  
##   
## Coefficients:  
## ma1 ma2 constant  
## -0.3984 -0.2173 0.0072  
## s.e. 0.0808 0.0768 0.0033  
##   
## sigma^2 estimated as 0.00982: log likelihood = 120.32, aic = -232.64  
##   
## $degrees\_of\_freedom  
## [1] 133  
##   
## $ttable  
## Estimate SE t.value p.value  
## ma1 -0.3984 0.0808 -4.9313 0.0000  
## ma2 -0.2173 0.0768 -2.8303 0.0054  
## constant 0.0072 0.0033 2.1463 0.0337  
##   
## $AIC  
## [1] -3.579224  
##   
## $AICc  
## [1] -3.562273  
##   
## $BIC  
## [1] -4.514974

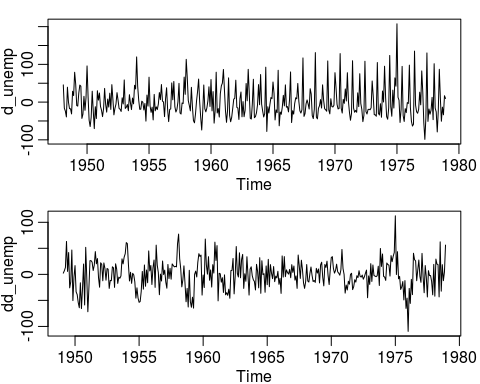
# Forecast data 35 years into the future  
sarima.for (globtemp,n.ahead = 35, p = 0, d = 1, q = 2)

## $pred  
## Time Series:  
## Start = 2016   
## End = 2050   
## Frequency = 1   
## [1] 0.7995567 0.7745381 0.7816919 0.7888457 0.7959996 0.8031534 0.8103072  
## [8] 0.8174611 0.8246149 0.8317688 0.8389226 0.8460764 0.8532303 0.8603841  
## [15] 0.8675379 0.8746918 0.8818456 0.8889995 0.8961533 0.9033071 0.9104610  
## [22] 0.9176148 0.9247687 0.9319225 0.9390763 0.9462302 0.9533840 0.9605378  
## [29] 0.9676917 0.9748455 0.9819994 0.9891532 0.9963070 1.0034609 1.0106147  
##   
## $se  
## Time Series:  
## Start = 2016   
## End = 2050   
## Frequency = 1   
## [1] 0.09909556 0.11564576 0.12175580 0.12757353 0.13313729 0.13847769  
## [7] 0.14361964 0.14858376 0.15338730 0.15804492 0.16256915 0.16697084  
## [13] 0.17125943 0.17544322 0.17952954 0.18352490 0.18743511 0.19126540  
## [19] 0.19502047 0.19870459 0.20232164 0.20587515 0.20936836 0.21280424  
## [25] 0.21618551 0.21951471 0.22279416 0.22602604 0.22921235 0.23235497  
## [31] 0.23545565 0.23851603 0.24153763 0.24452190 0.24747019

### SERIES DE TIEMPO ESTACIONALES   
  
# Plot unemp   
plot(unemp)



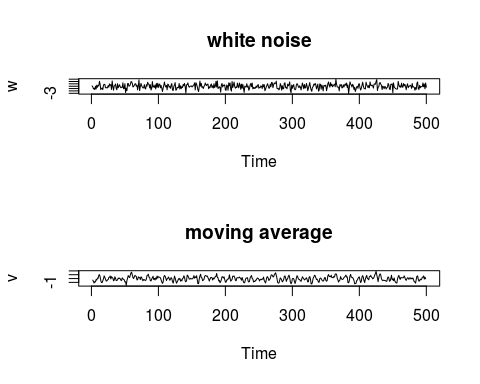
# Difference your data and plot it  
d\_unemp <- diff(unemp)   
plot(d\_unemp)  
  
# Seasonally difference d\_unemp and plot it  
dd\_unemp <- diff(d\_unemp, lag = 12)   
plot(dd\_unemp)



## Pruebas series de tiempo de Time Series Analysis and its applications text

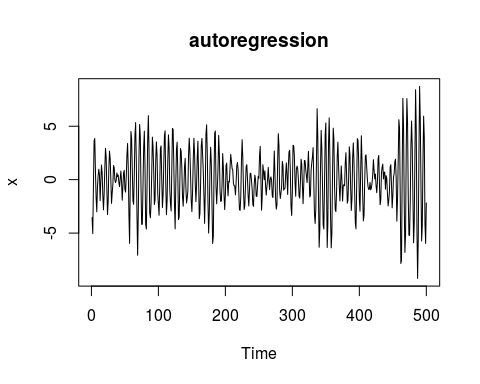
Página 13 Moving Average

w = rnorm (500,0,1)  
v = filter(w, sides = 2, rep(1/3,3))  
par(mfrow=c(2,1))  
plot.ts(w,main = "white noise")  
plot.ts(v, main = "moving average")



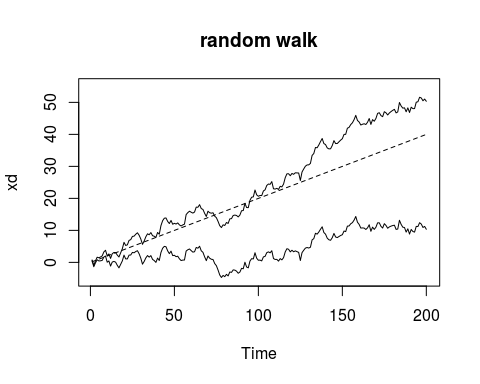
Autoregesivo

w = rnorm(550,0,1)  
x = filter (w, filter = c(1,-0.9), method = "recursive")[-(1:50)]  
plot.ts(x, main = "autoregression")



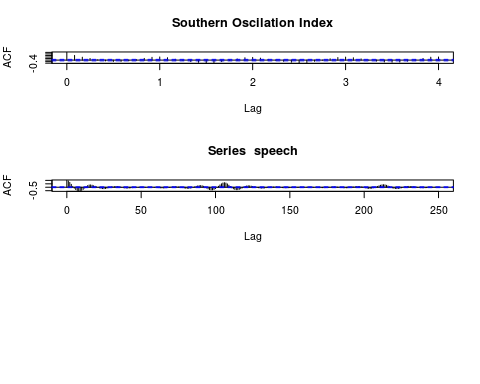
Autoregesivo con drift

set.seed(154)  
w = rnorm(200,0,1); x = cumsum (w)  
wd = w + .2; xd = cumsum(wd)  
plot.ts(xd, ylim = c(-5,55), main = "random walk")  
lines(x); lines(0.2\*(1:200), lty = "dashed")

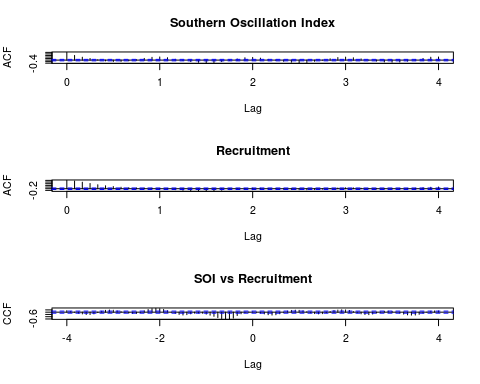


ACF

par(mfrow = c(3,1))  
acf(soi, 48, main = "Southern Oscilation Index")  
acf(speech, 250)  
  
par(mfrow=c(3,1))



acf(soi, 48, main="Southern Oscillation Index")   
acf(rec, 48, main="Recruitment")  
ccf(soi, rec, 48, main="SOI vs Recruitment", ylab="CCF")



# Varios vectores  
  
persp(1:64, 1:36, soiltemp, phi=30, theta=30, scale=FALSE, expand=4, ticktype="detailed", xlab="rows", ylab="cols", zlab="temperature")  
  
plot.ts(rowMeans(soiltemp), xlab="row", ylab="Average Temperature")

