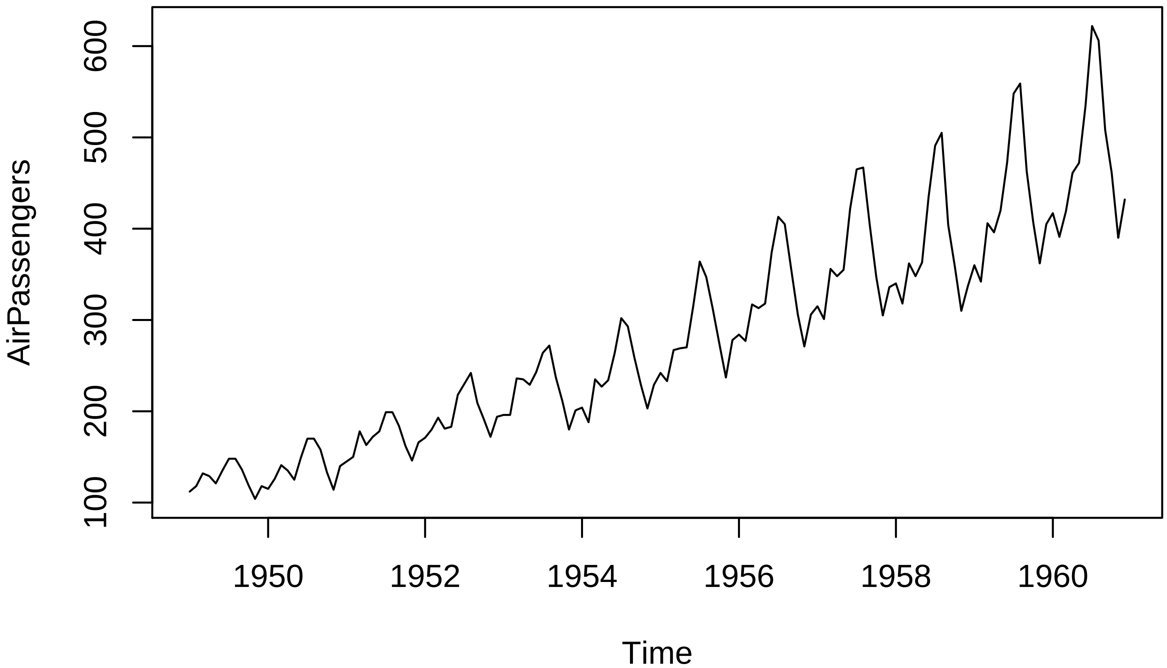
TimeSeries

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
```

```r
deltat(AirPassengers)
```

```
## [1] 0.08333333
```

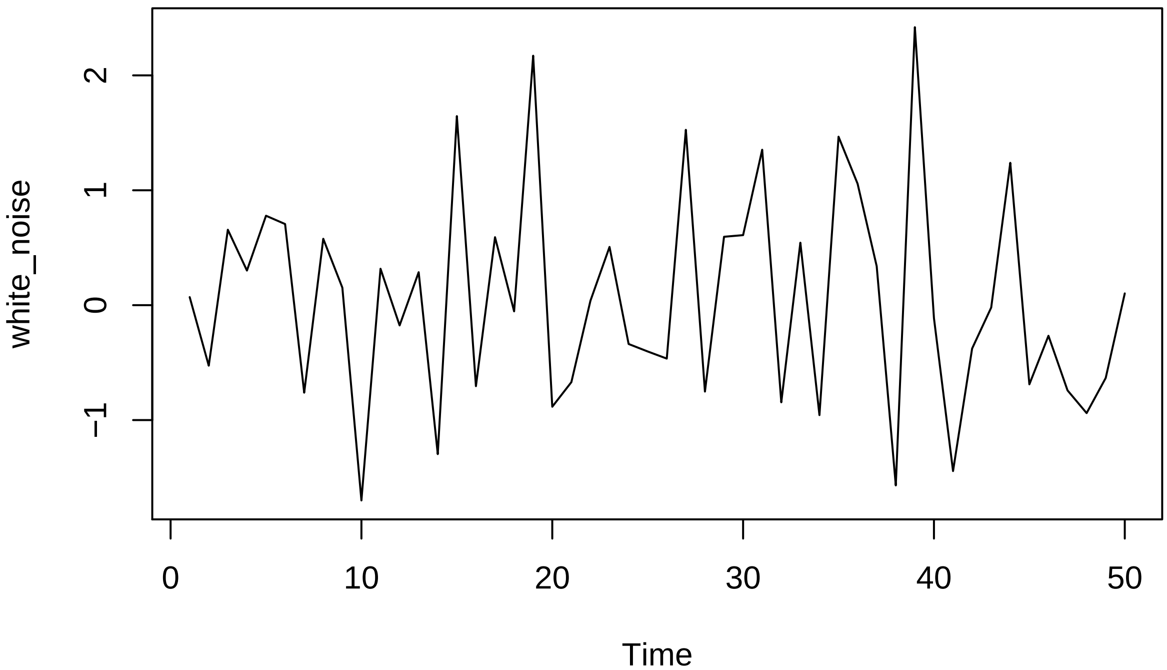
```r
frequency(AirPassengers)
```

```
## [1] 12
```

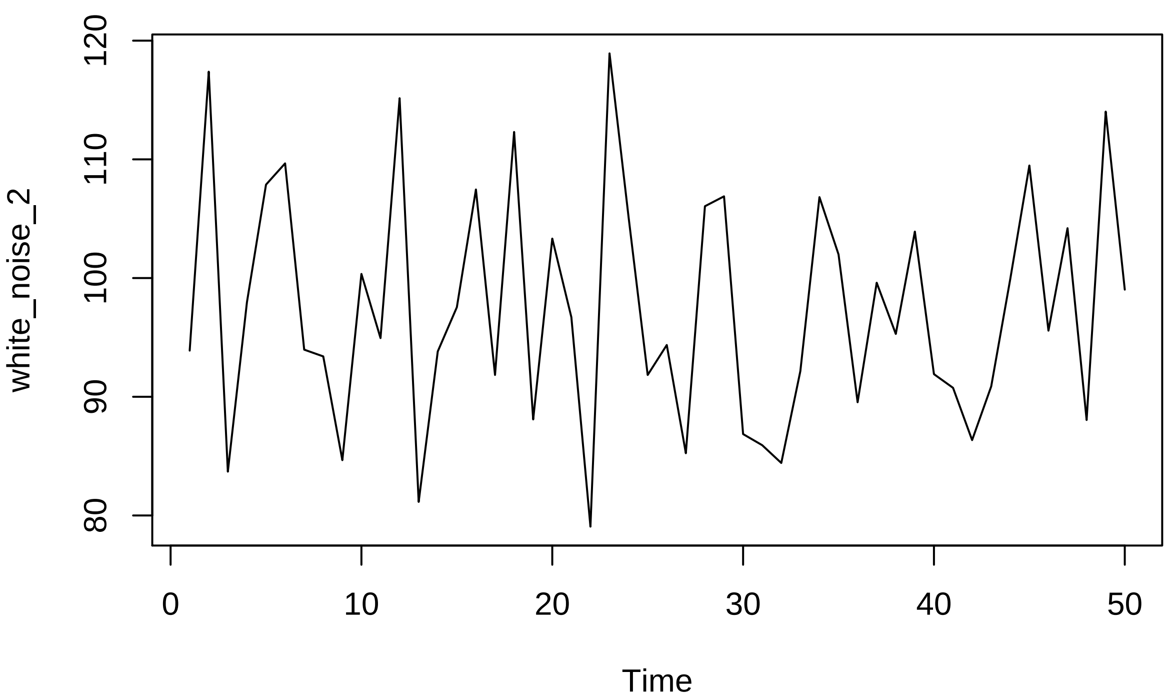
```r
cycle(AirPassengers)
```

```
##      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
```

```r
# 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)
```



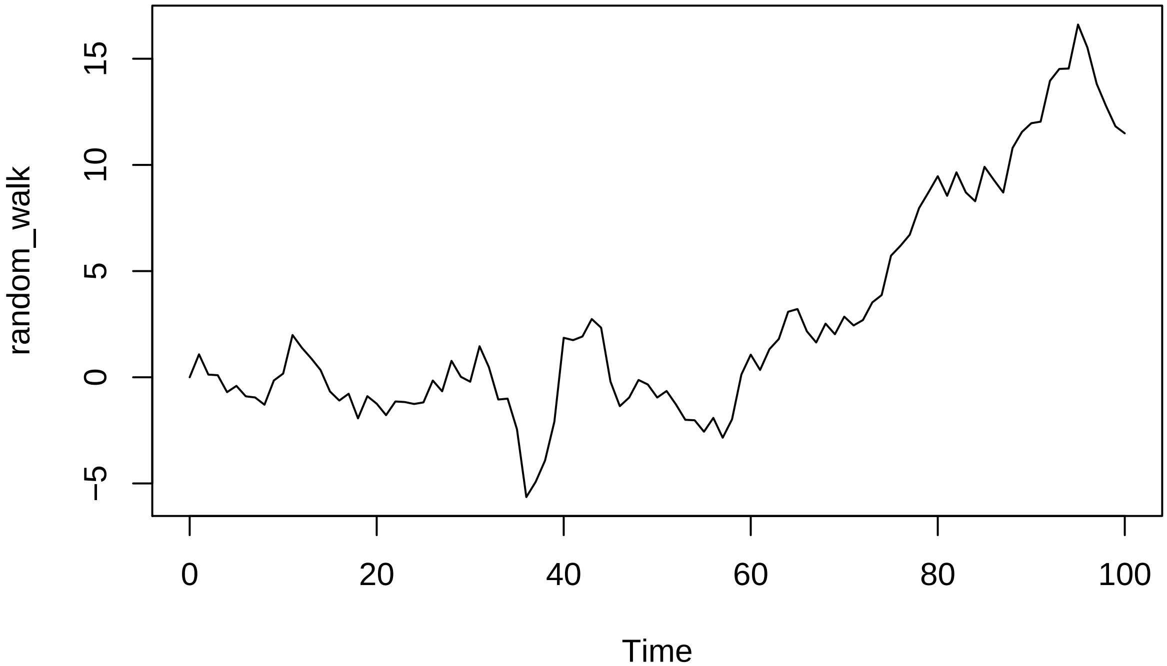

```r
# Simulate from the WN model with: mean = 100, sd = 10
white_noise_2 <- arima.sim(model = list(order = c(0,0,0)), n = 50, mean = 100, sd = 10)

# Plot your white_noise_2 data
ts.plot(white_noise_2)
```



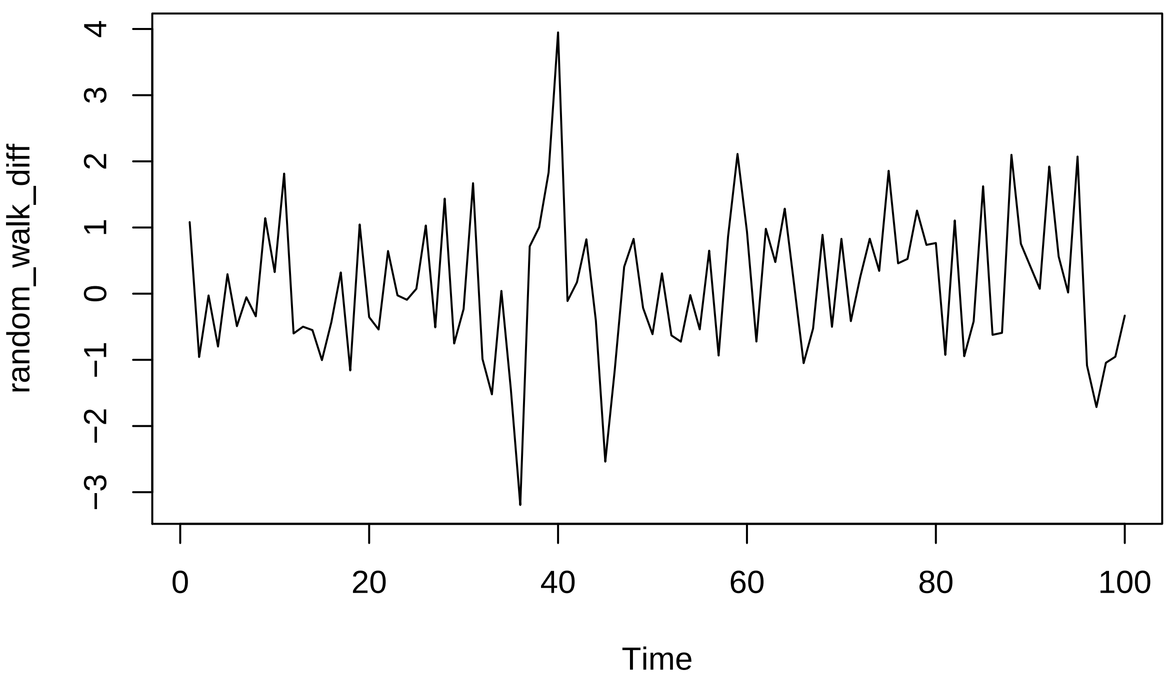
Los cambios en una serie de tiempo de Random Walk siguen un comportamiento de White noise.

```r
# Generate a RW model using arima.sim
random_walk <- arima.sim(model = list(order = c(0, 1, 0)) , n = 100)

# Plot random_walk
```

```r
ts.plot (random_walk)
```

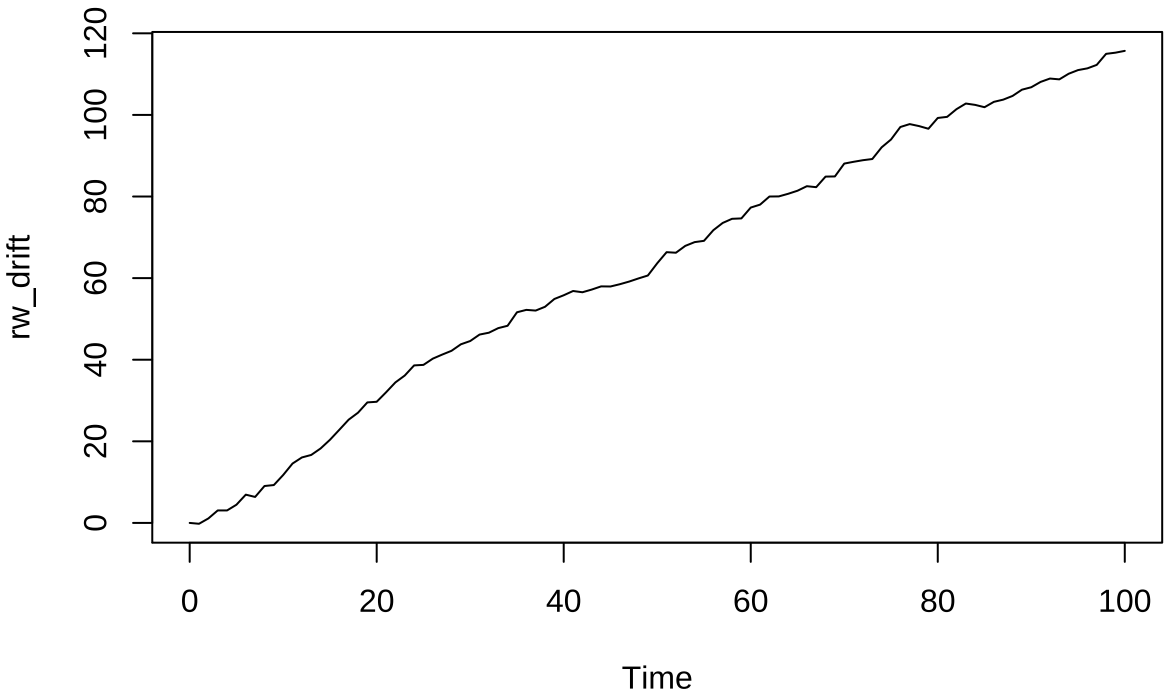



```r
# Calculate the first difference series
random_walk_diff <- diff(random_walk)

# Plot random_walk_diff

ts.plot(random_walk_diff)
```

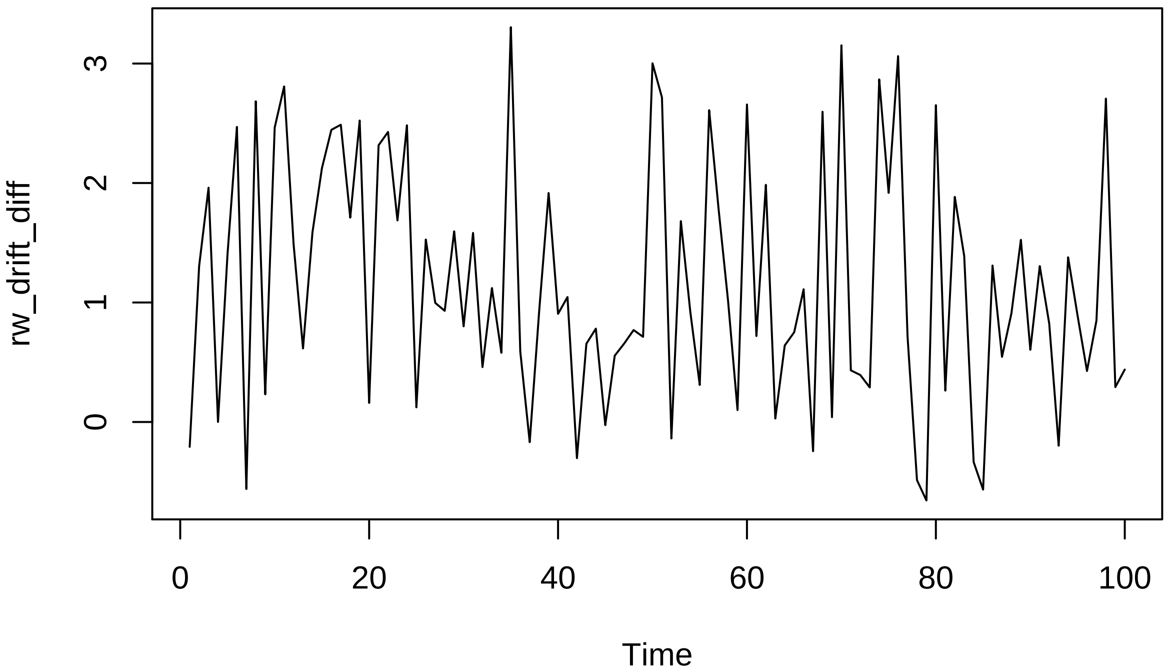



```r
# RANDOM WALK WITH DRIFT
```

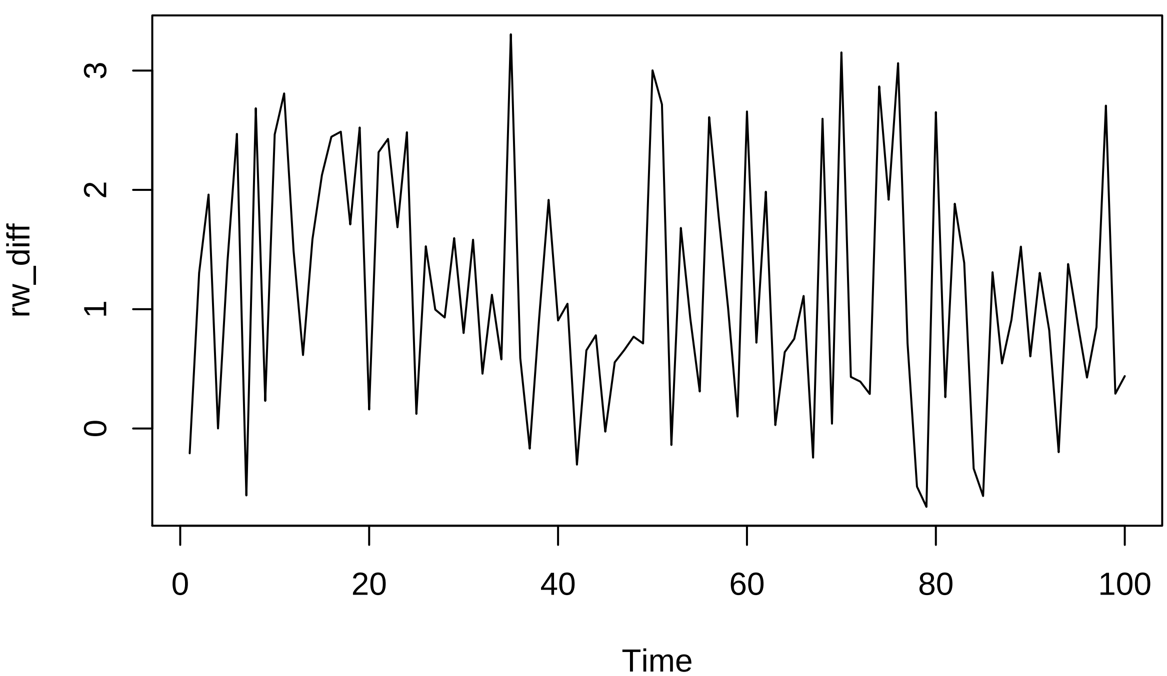
```r
# Generate a RW model with a drift uing arima.sim
rw_drift <- arima.sim(model = list(order = c(0, 1, 0)), n = 100, mean = 1)

# Plot rw_drift
ts.plot(rw_drift)
```




```r
# Calculate the first difference series
rw_drift_diff <- diff(rw_drift)

# Plot rw_drift_diff
ts.plot(rw_drift_diff)
```

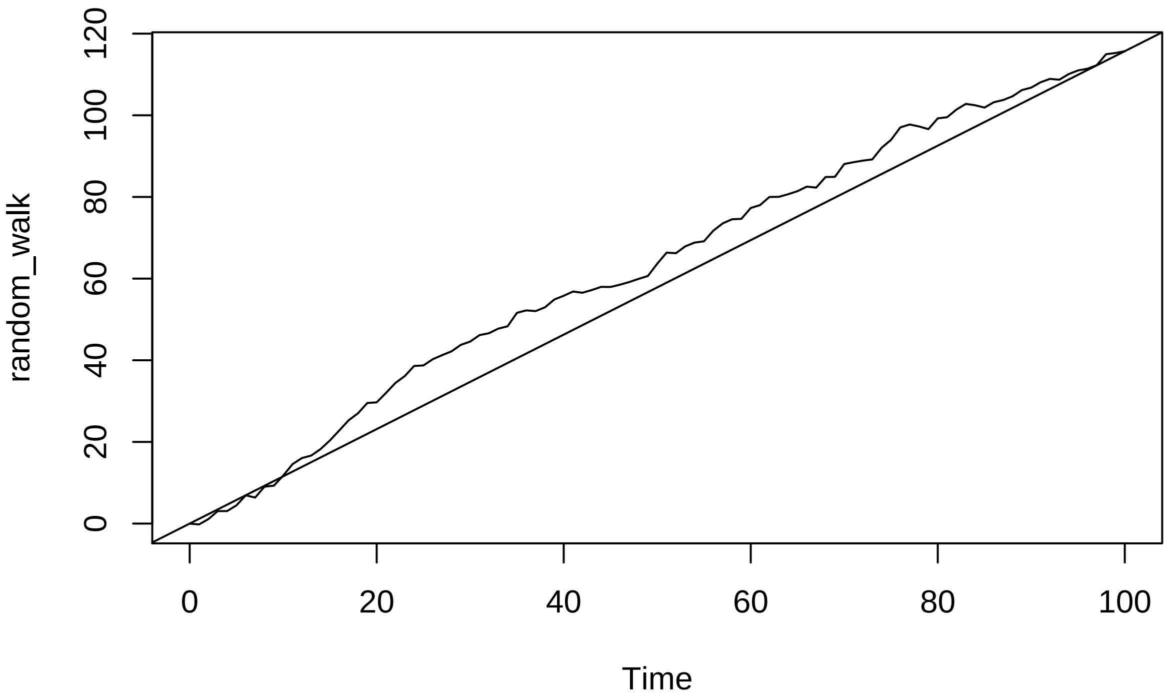



```r
random_walk <- rw_drift

# Difference your random_walk data
rw_diff <- diff(random_walk)


# Plot rw_diff
plot.ts(rw_diff)
```

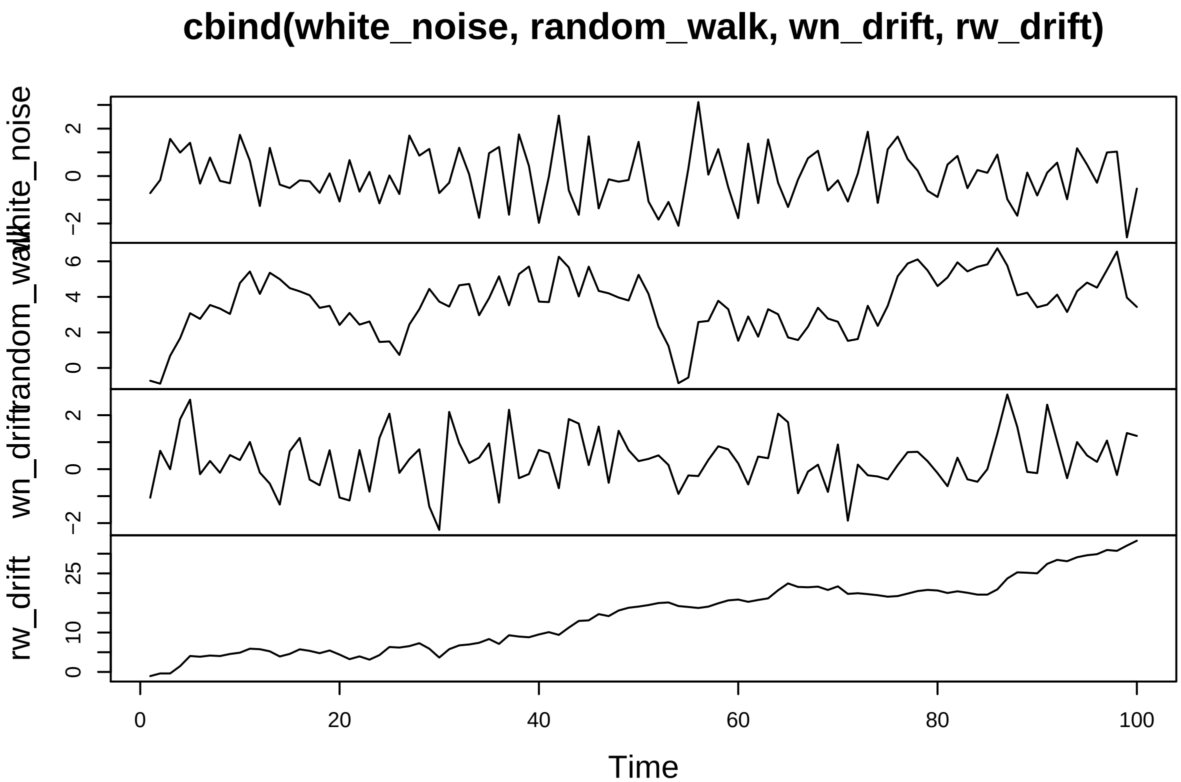



```r
# Now fit the WN model to the differenced data
model_wn <- arima(rw_diff,order = c(0, 0, 0))
```

```r
# 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)
```

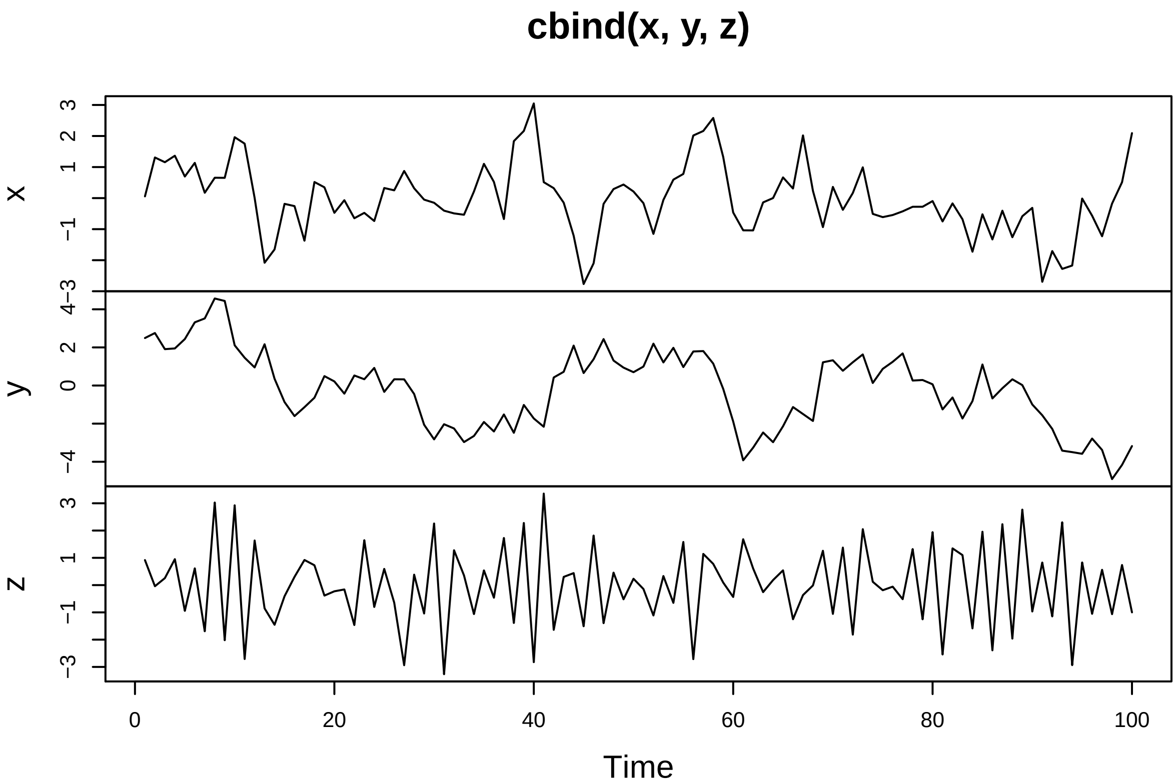



```r
#  RANDOM WALK CON Y SIN DRIFT (Estationarity)
#
# The white noise (WN) and random walk (RW) models are very closely related. However, only the RW is al
#
# Recall that if we start with a mean zero WN process and compute its running or cumulative sum, the re

# 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))
```

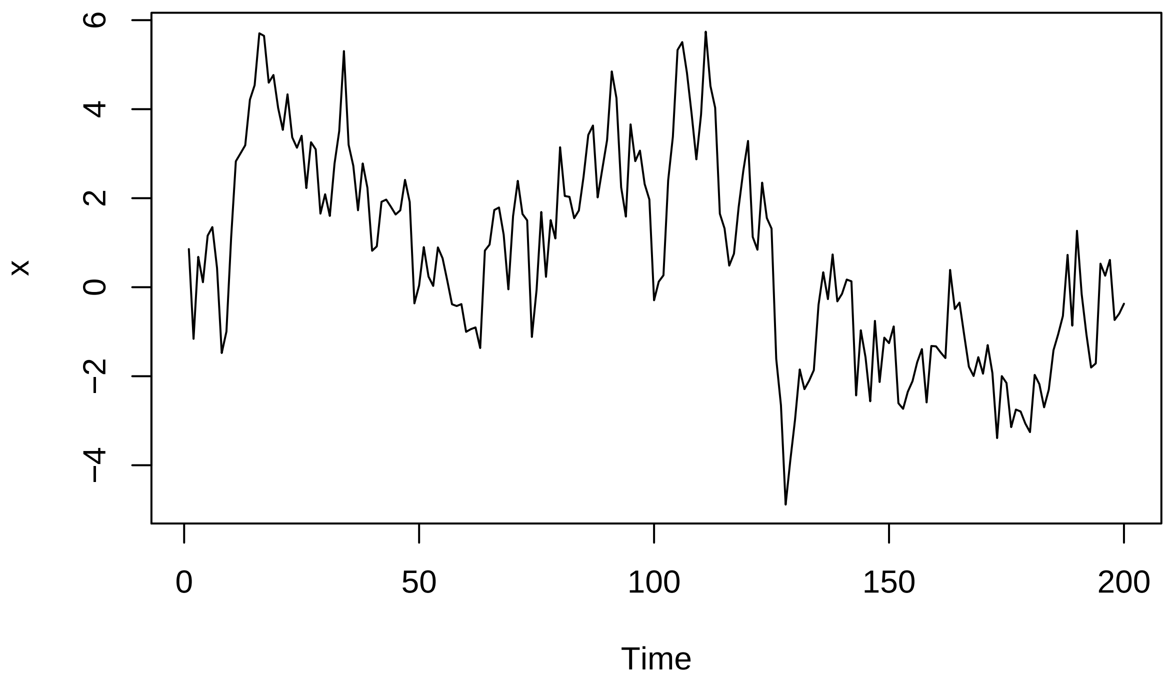



```r
#
# # Generate means from eu_percentreturns
# colMeans(eu_percentreturns)
#
# # Use apply to calculate sample variance from eu_percentreturns
# apply(eu_percentreturns, MARGIN = 2, FUN = var)
#
# # Use apply to calculate standard deviation from eu_percentreturns
# apply(eu_percentreturns, MARGIN = 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)
```

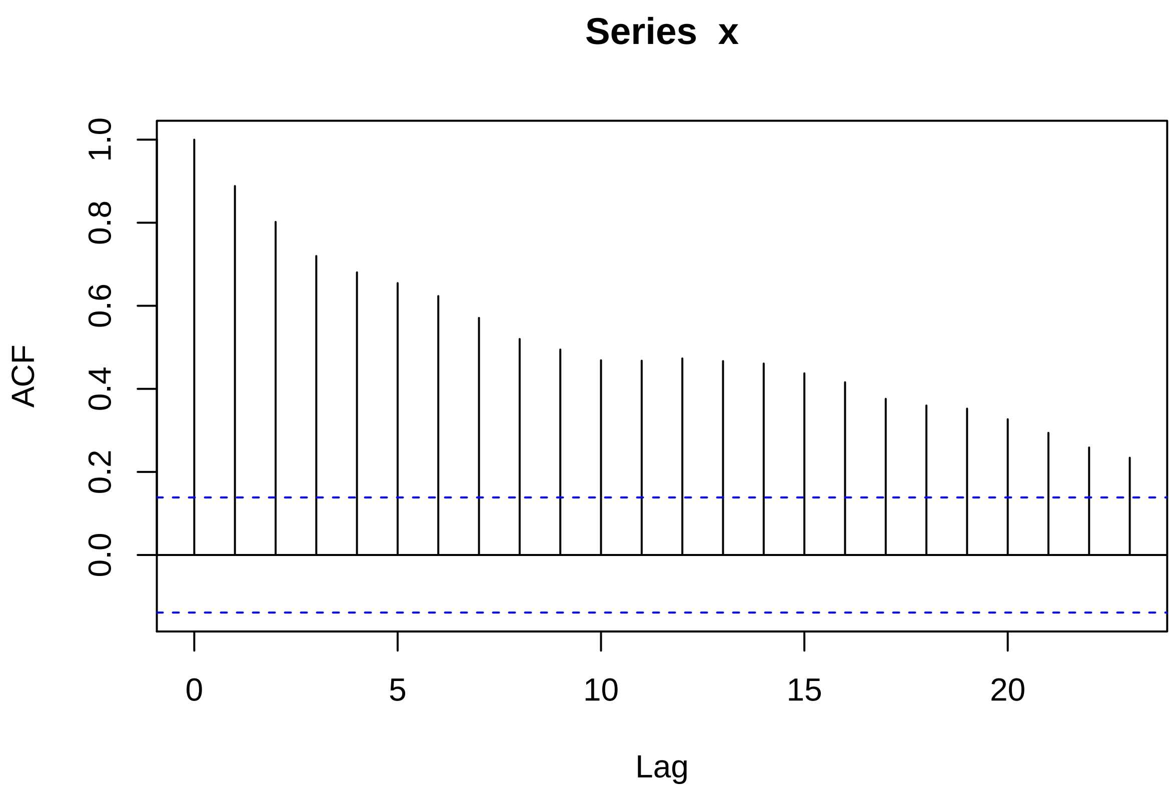
```r
# Simulate an AR model with -0.75 slope
z <- arima.sim(model = list(ar = -0.75), n = 100)

# Plot your simulated data
plot.ts(cbind(x, y, z))
```



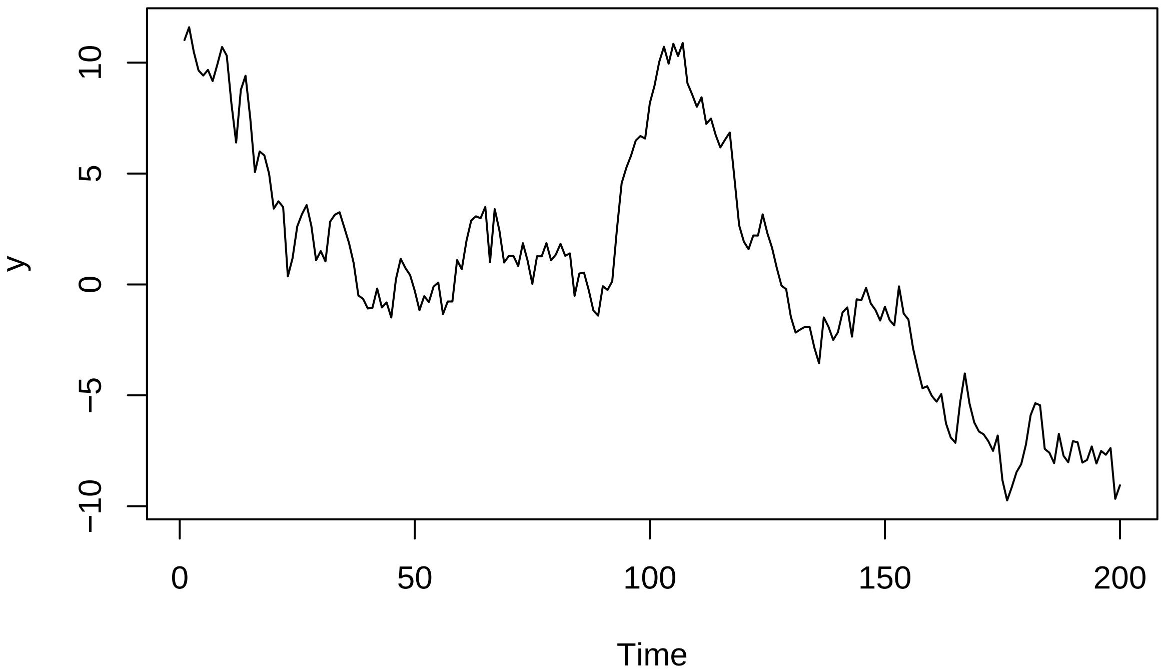

```r
###### COMPARAR MODELOS AR con RandomWalks

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



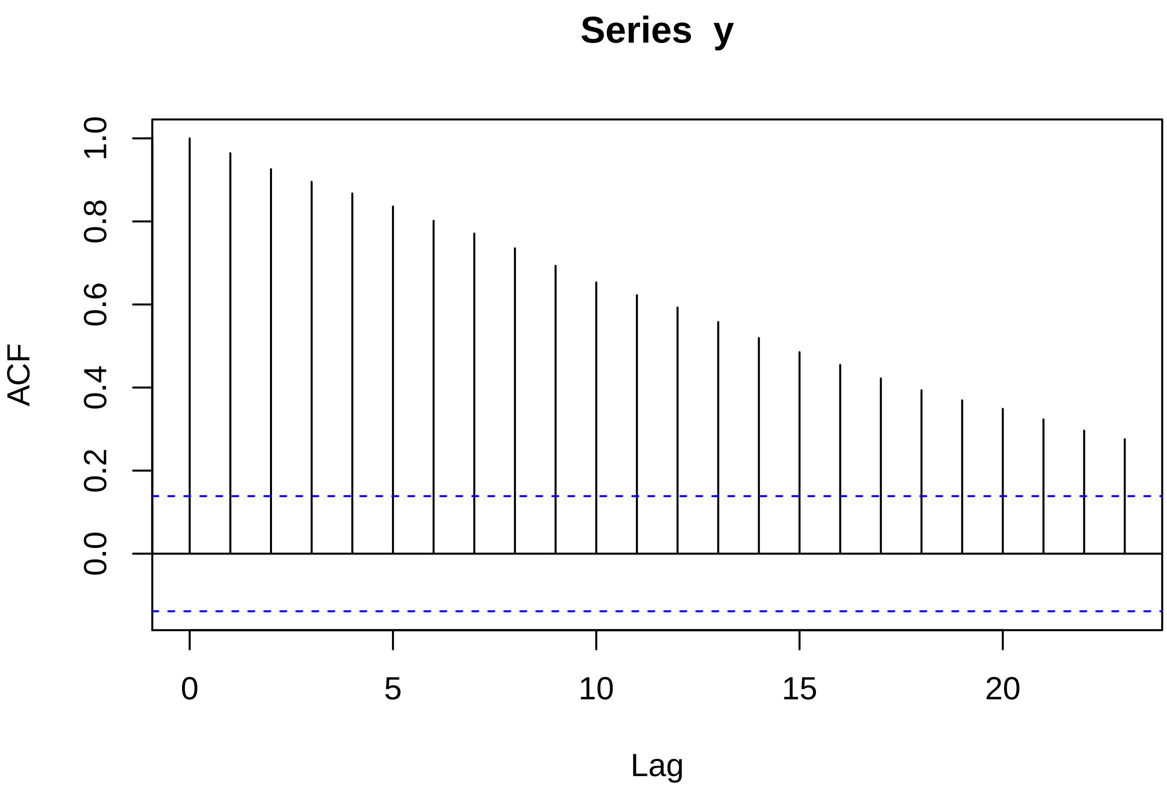

```r
acf(x)
```



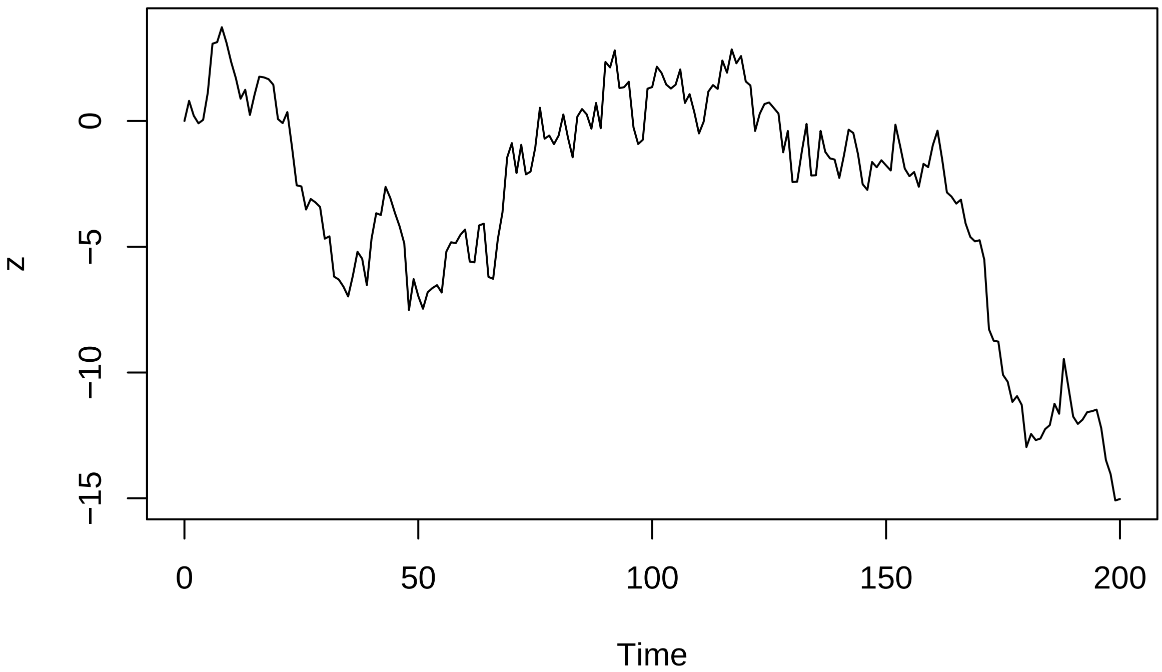

```r
# Simulate and plot AR model with slope 0.98
y <- arima.sim(model = list(ar = 0.98), n = 200)
ts.plot(y)
```



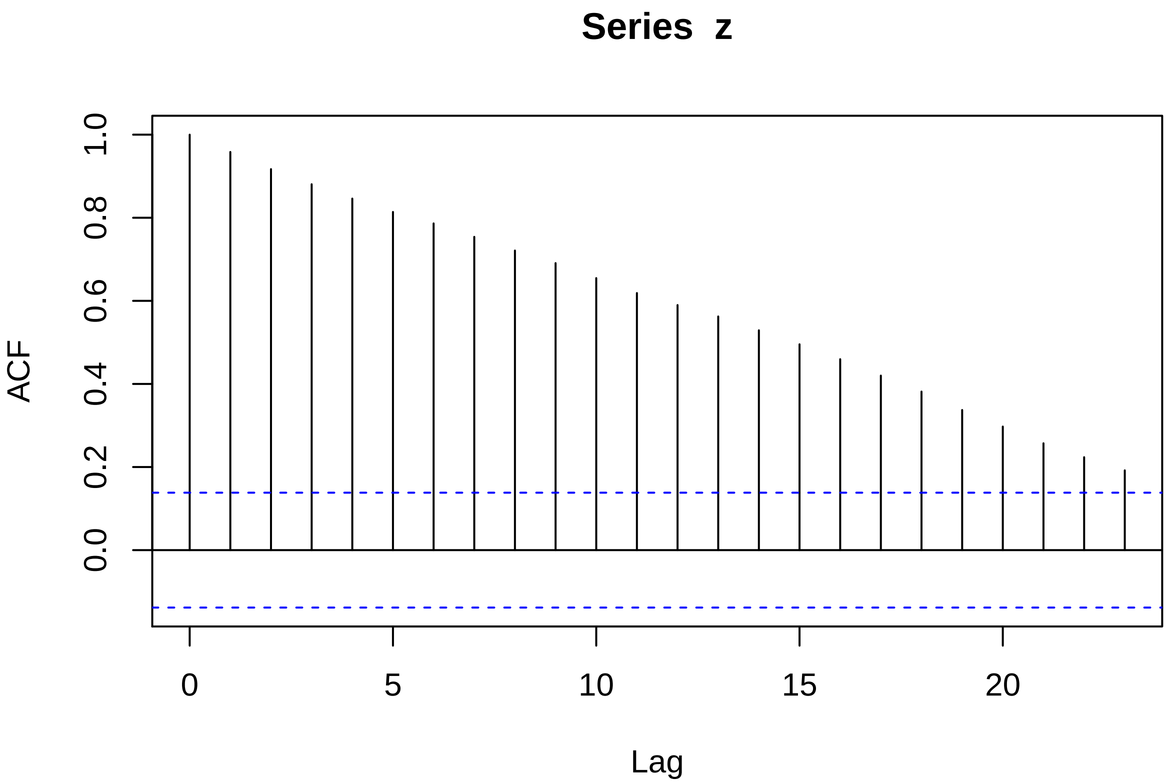

```r
acf(y)
```




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

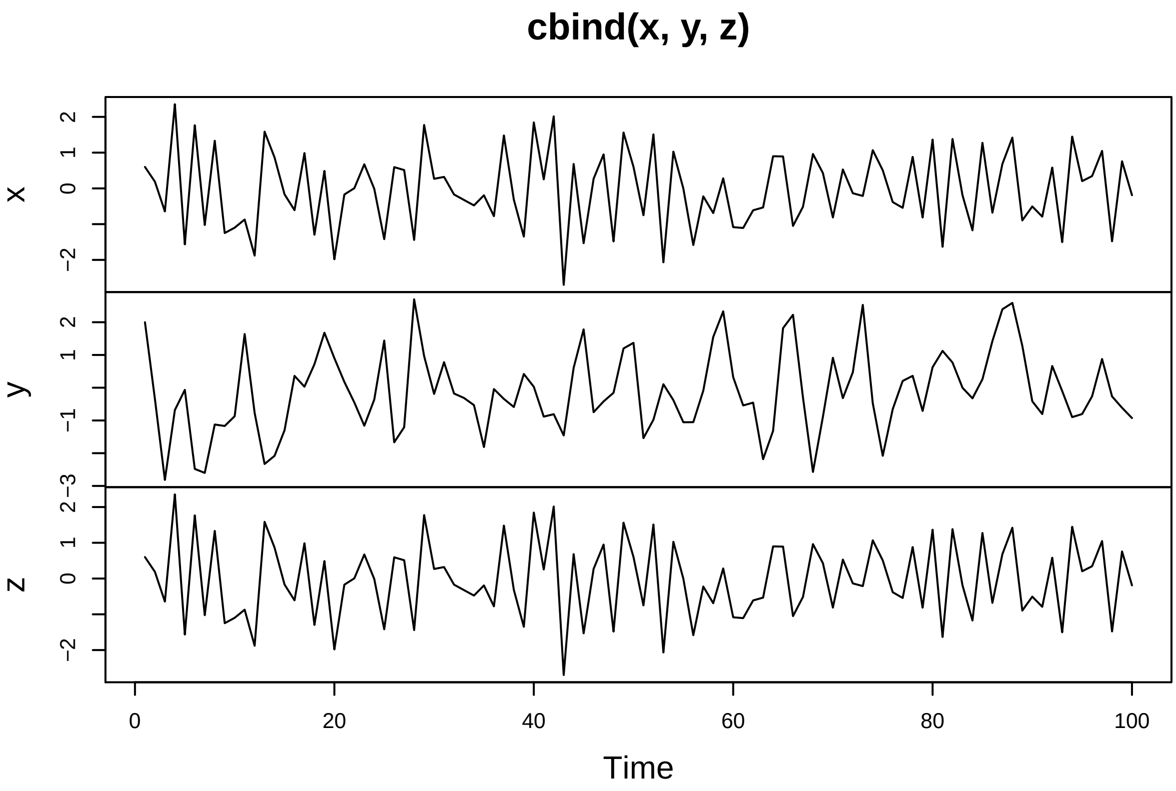



```r
acf(z)
```




```r
# # 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
```

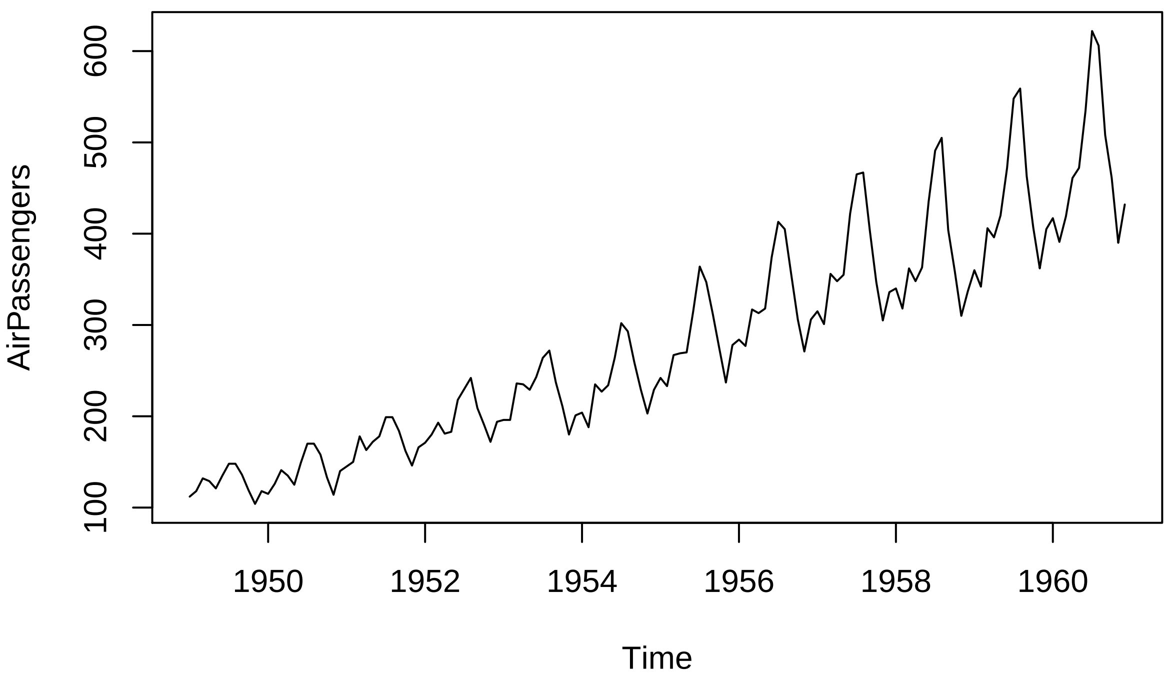
```r
# predict_AR <- predict(AR_fit)
#
# # Obtain the 1-step forecast using $pred[1]
# predict_AR$pred[1]
#
# # Use predict to make 1-step through 10-step forecasts
# predict(AR_fit, n.ahead = 10)
#
# # Run to plot the Nile series plus the forecast and 95% prediction intervals
# ts.plot(Nile, xlim = 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))
```



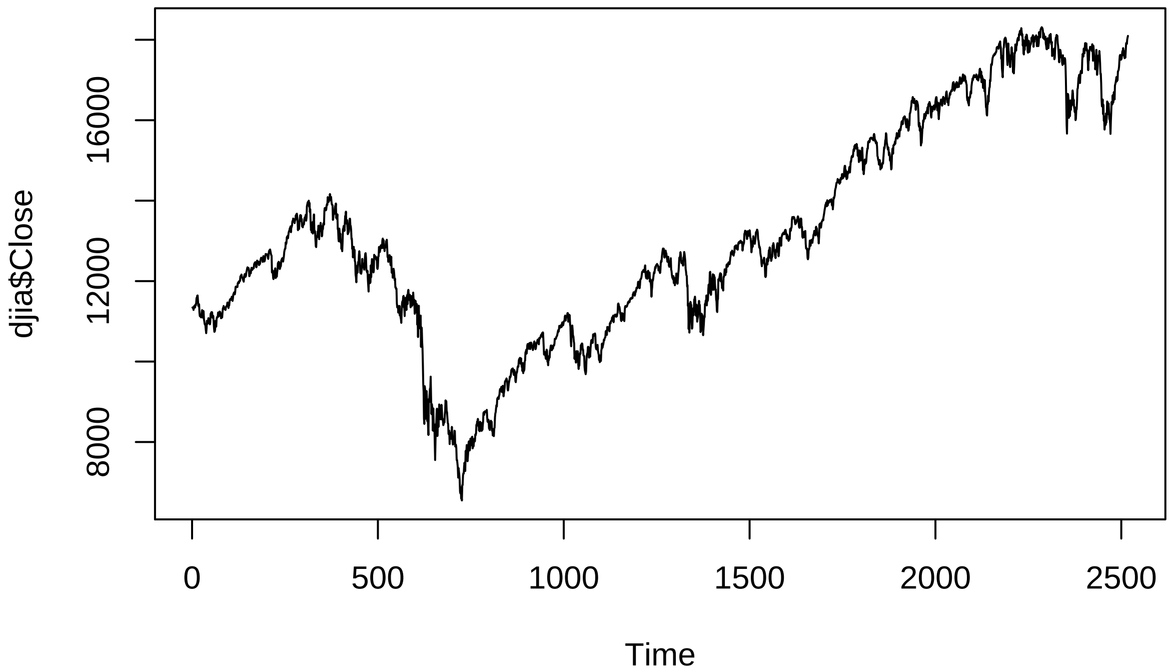
```r
# # # Ajustando un MA
#
# # Make a 1-step forecast based on MA
# predict_MA <- predict(MA, n.ahead = 1)
#
# # Obtain the 1-step forecast using $pred[1]
# predict_MA$pred[1]
#
# # Make a 1-step through 10-step forecast based on MA
# predict(MA, n.ahead = 10)
#
# # Plot the Nile series plus the forecast 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

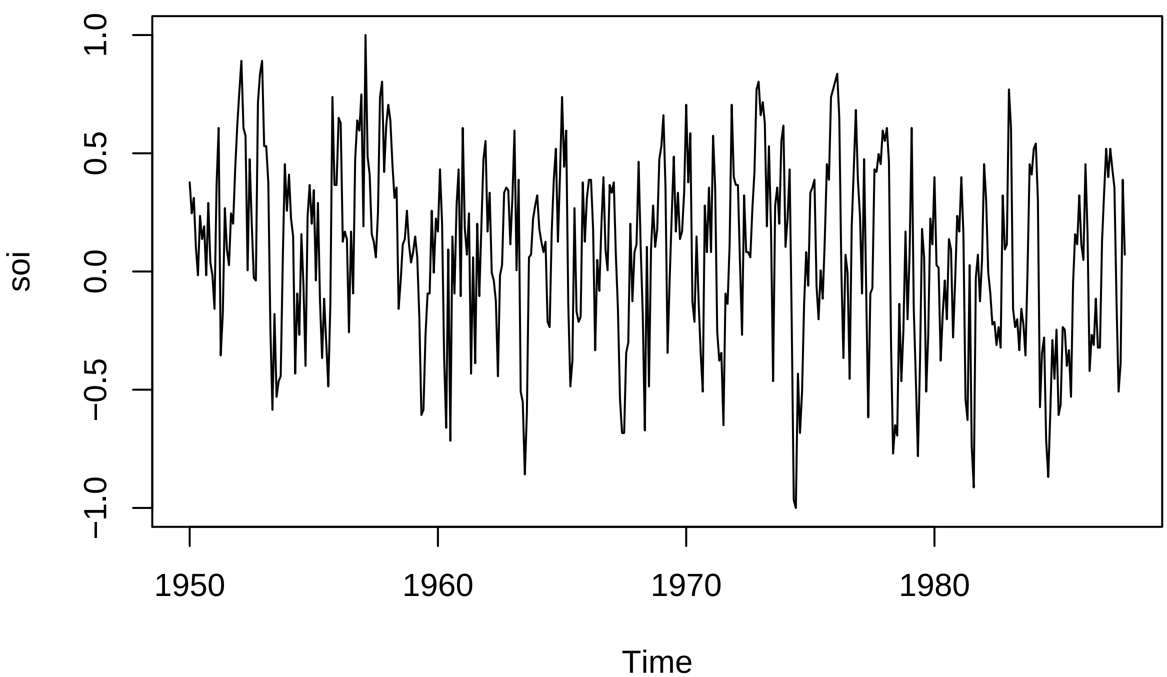
```r
# # R Arima base

# View a detailed description of AirPassengers
help(AirPassengers)

# Plot AirPassengers
ts.plot(AirPassengers)
```



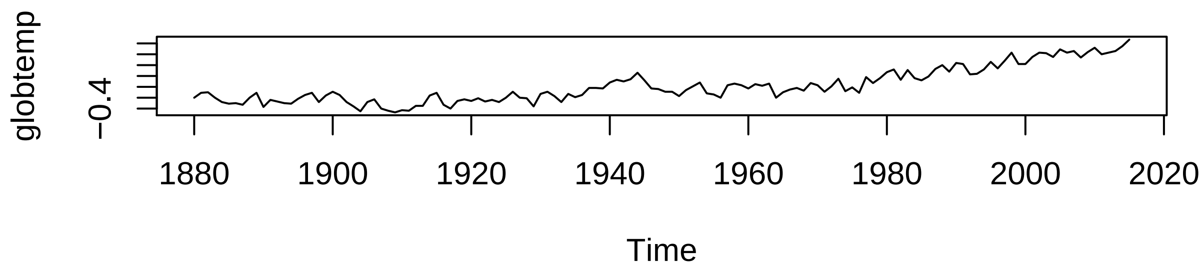

```r
# Plot the DJIA daily closings
ts.plot(djia$Close)
```

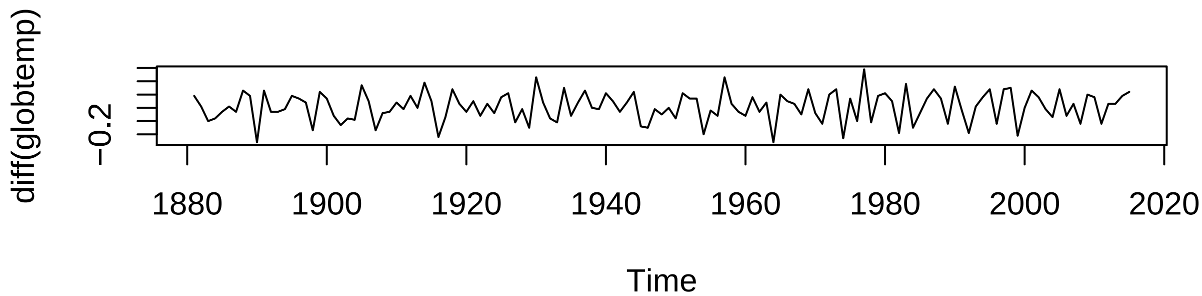



```r
# Plot the Southern Oscillation Index
plot(soi)
```

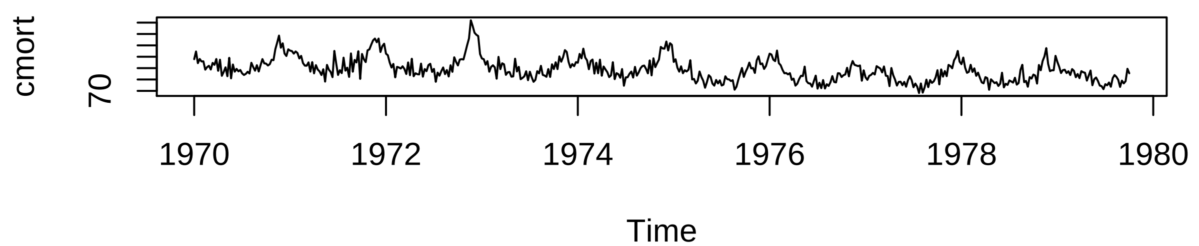


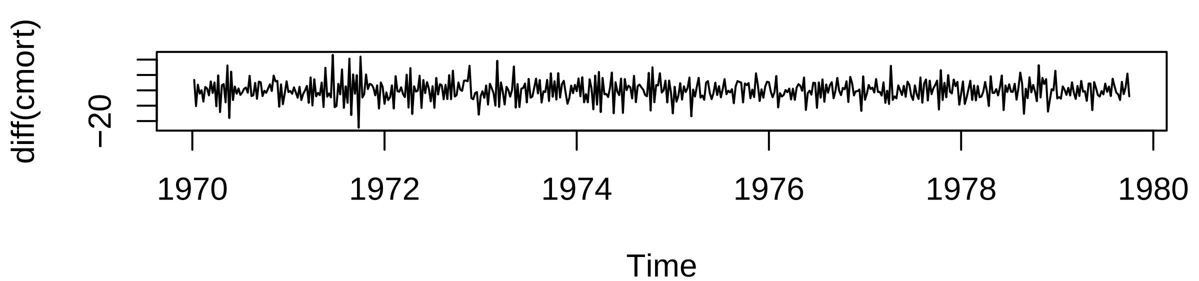

```r
# Plot globtemp and detrended globtemp
par(mfrow = c(2,1))
plot(globtemp)
plot(diff(globtemp))
```



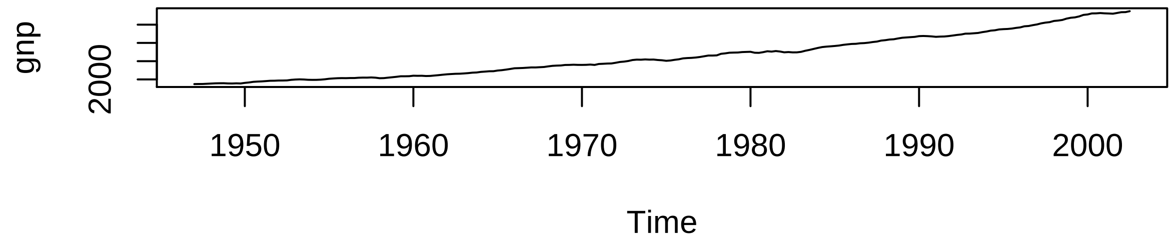


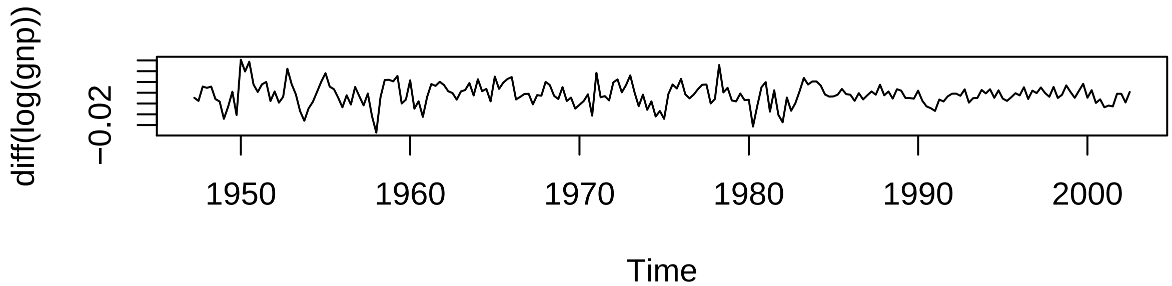

```
# Plot cmort and detrended cmort
par(mfrow = c(2,1))
plot(cmort)
plot(diff(cmort))
```



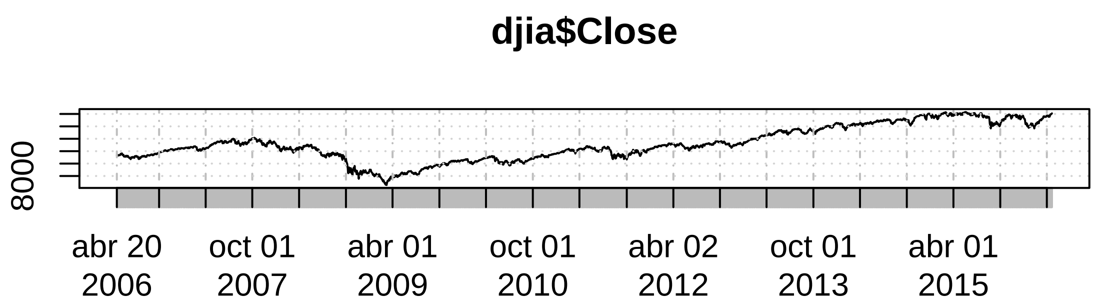


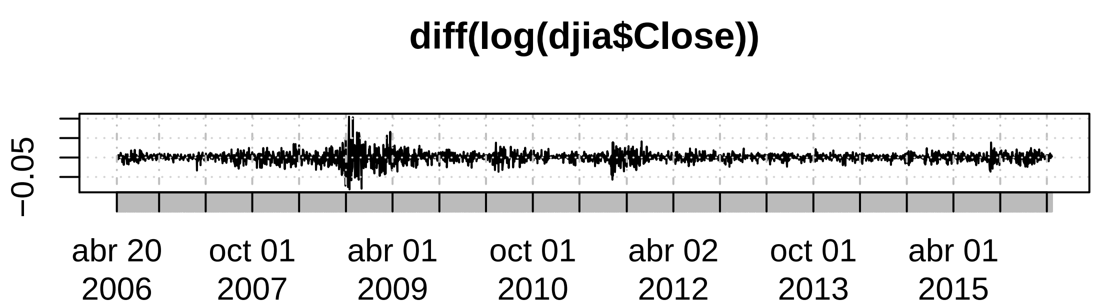

```
# Plot GNP series (gnp) and its growth rate
par(mfrow = c(2,1))
plot(gnp)
plot(diff(log(gnp)))
```





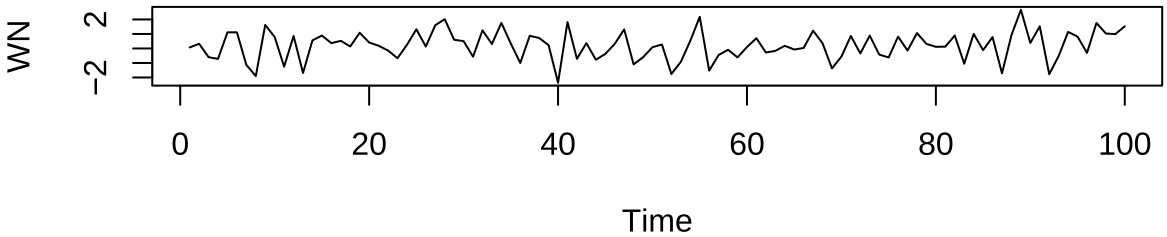

```r
# Plot DJIA closings (djia$Close) and its returns
par(mfrow = c(2,1))
plot(djia$Close)
plot(diff(log(djia$Close)))
```

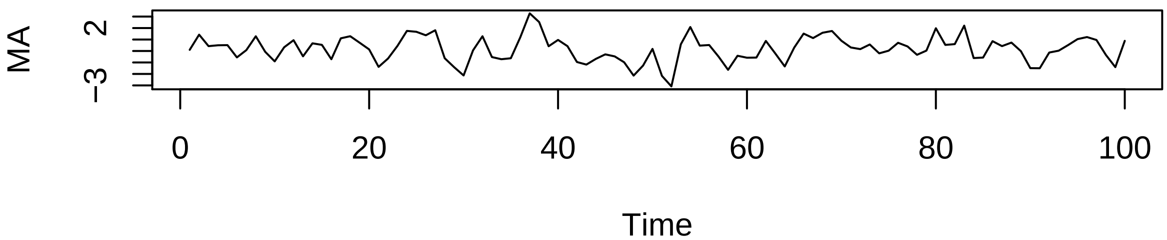




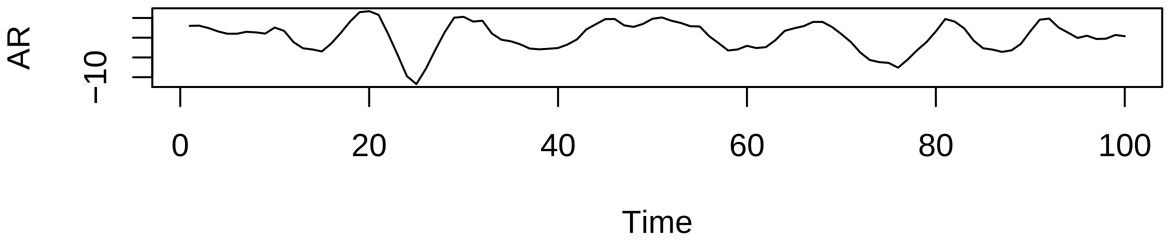

```r
##SIMULAR MA & AR

# Generate and plot white noise
WN <- arima.sim(model=list(order=c(0,0,0)),n=100)
plot(WN)
```

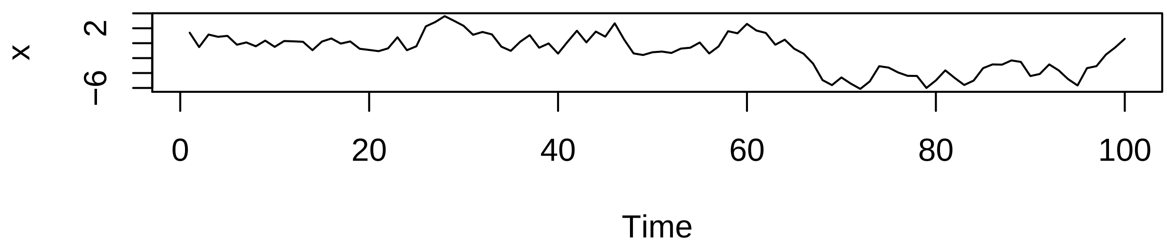
```r
# Generate and plot an MA(1) with parameter .9
MA <- arima.sim(model=list(order=c(0,0,1), ma = 0.9),n=100)
plot(MA)
```



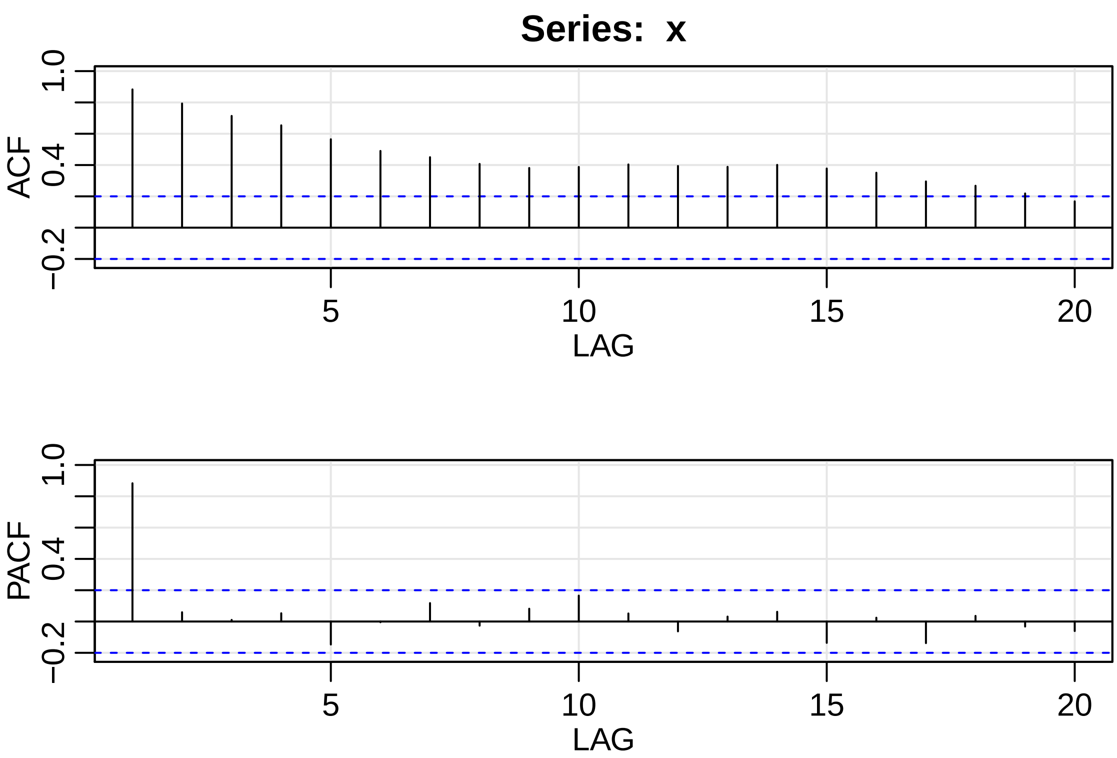



```r
# Generate and plot an AR(2) with parameters 1.5 and -.75
AR <- arima.sim(model=list(order=c(2,0,0), ar =c(1.5,-0.75)),n=100)
plot(AR)

# # 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)
```






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

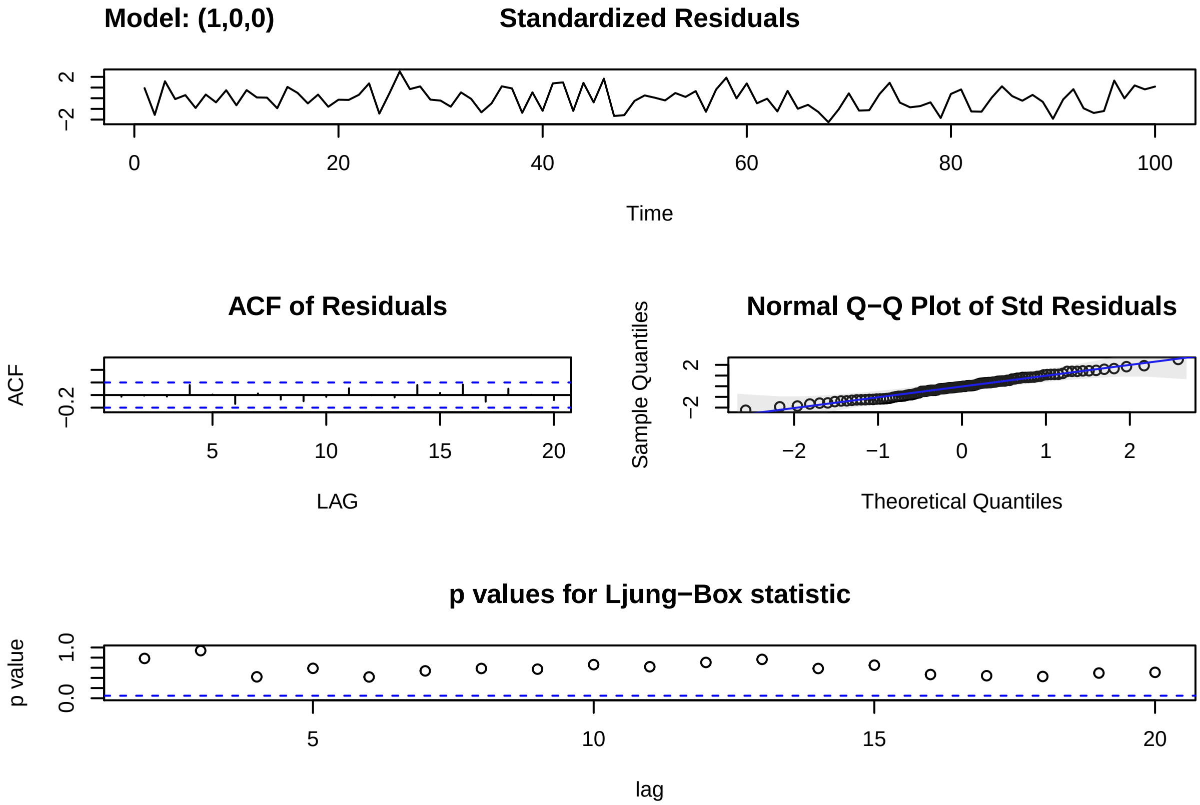



```
##         ACF  PACF
##  [1,] 0.88  0.88
##  [2,] 0.79  0.06
##  [3,] 0.71  0.01
##  [4,] 0.65  0.05
##  [5,] 0.56 -0.15
```

```
##  [6,] 0.49  0.00
##  [7,] 0.45  0.12
##  [8,] 0.41 -0.03
##  [9,] 0.38  0.08
## [10,] 0.39  0.17
## [11,] 0.40  0.05
## [12,] 0.39 -0.06
## [13,] 0.39  0.03
## [14,] 0.40  0.06
## [15,] 0.38 -0.14
## [16,] 0.35  0.02
## [17,] 0.30 -0.14
## [18,] 0.27  0.04
## [19,] 0.22 -0.03
## [20,] 0.17 -0.06
```

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

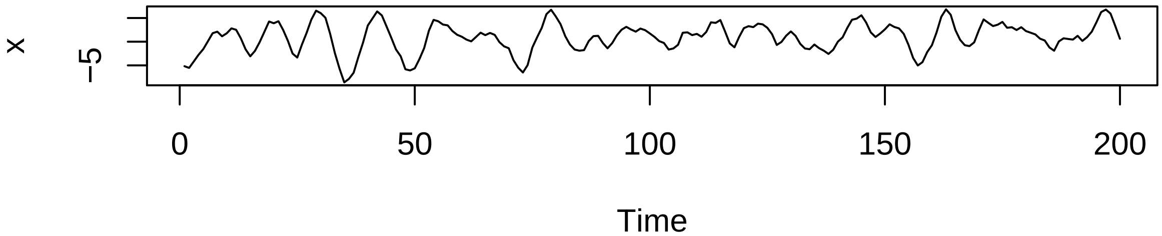
```
## initial  value 0.864080
## iter   2 value 0.076128
## iter   3 value 0.076053
## iter   4 value 0.076050
## iter   5 value 0.076037
## iter   6 value 0.076032
## iter   7 value 0.076031
## iter   8 value 0.076030
## iter   9 value 0.076030
## iter  10 value 0.076030
## iter  11 value 0.076030
## iter  11 value 0.076030
## final  value 0.076030
## converged
## initial  value 0.084971
## iter   2 value 0.084844
## iter   3 value 0.084104
## iter   4 value 0.084084
## iter   5 value 0.084076
## iter   6 value 0.084070
## iter   7 value 0.084070
## iter   8 value 0.084069
## iter   9 value 0.084069
## iter  10 value 0.084069
## iter  11 value 0.084069
## iter  11 value 0.084069
## final  value 0.084069
## converged
```

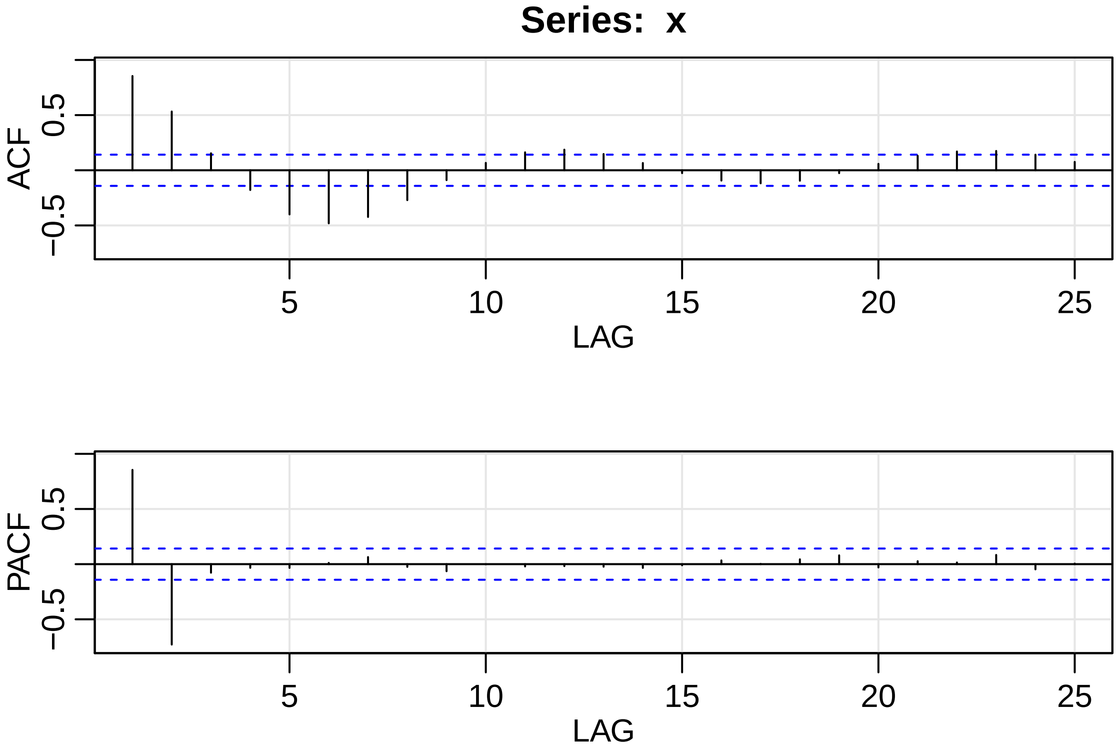



```
## $fit
##
## Call:
## stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,
##     Q), period = S), xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc,
##     REPORT = 1, reltol = tol))
##
## Coefficients:
##          ar1    xmean
##       0.8892  -0.8150
## s.e.  0.0439   0.9109
##
## sigma^2 estimated as 1.165:  log likelihood = -150.3,  aic = 306.6
##
## $degrees_of_freedom
## [1] 98
##
## $ttable
##       Estimate     SE t.value p.value
## ar1     0.8892 0.0439 20.2535  0.0000
## xmean  -0.8150 0.9109 -0.8948  0.3731
##
## $AIC
## [1] 1.192502
##
## $AICc
## [1] 1.215002
##
```

```
## $BIC
## [1] 0.2446051
```

```r
# # 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)
```



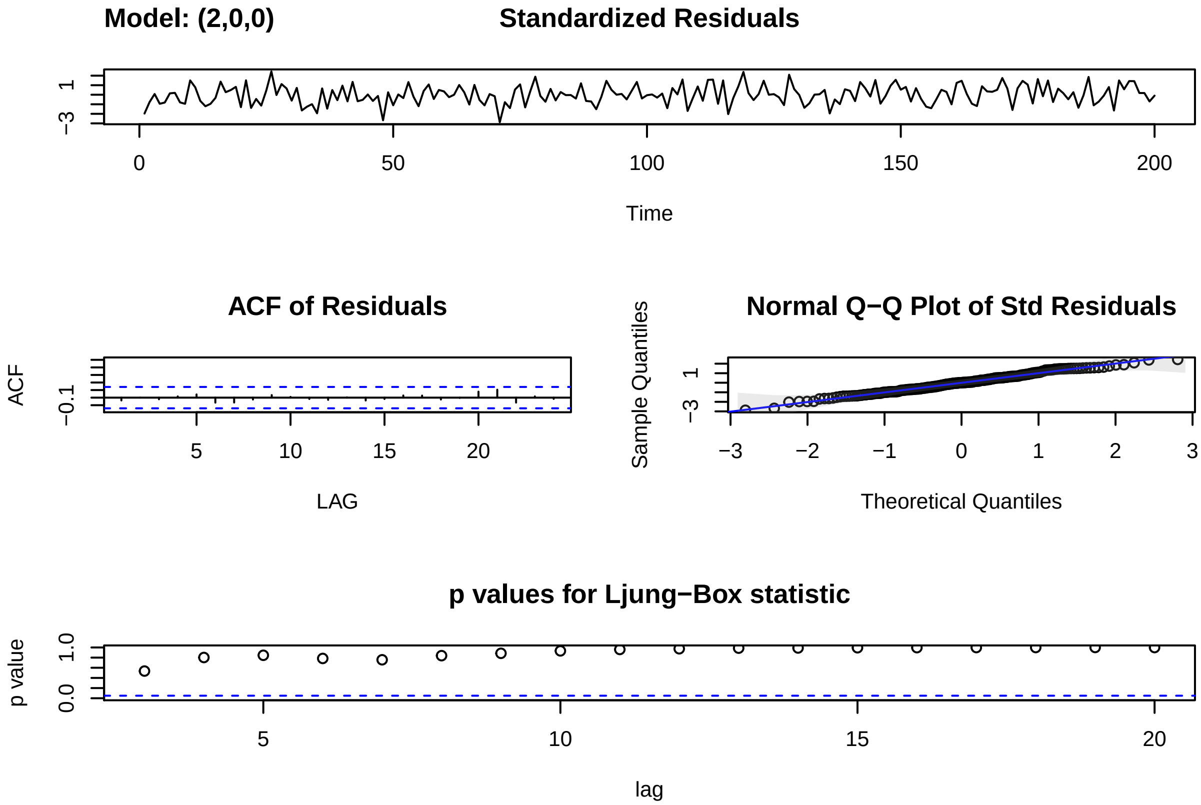



```
##         ACF  PACF
##  [1,]  0.85  0.85
##  [2,]  0.53 -0.73
##  [3,]  0.15 -0.08
##  [4,] -0.18 -0.03
##  [5,] -0.40 -0.03
##  [6,] -0.48  0.01
##  [7,] -0.42  0.06
##  [8,] -0.27 -0.02
##  [9,] -0.09 -0.06
## [10,]  0.07  0.00
```

```
## [11,]  0.16 -0.02
## [12,]  0.19 -0.02
## [13,]  0.15 -0.02
## [14,]  0.07 -0.03
## [15,] -0.03 -0.01
## [16,] -0.09  0.03
## [17,] -0.12  0.00
## [18,] -0.09  0.04
## [19,] -0.03  0.08
## [20,]  0.06 -0.03
## [21,]  0.13  0.03
## [22,]  0.17  0.01
## [23,]  0.18  0.08
## [24,]  0.14 -0.05
## [25,]  0.08  0.01
```

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

```
## initial  value 1.095203
## iter   2 value 1.001045
## iter   3 value 0.540681
## iter   4 value 0.341701
## iter   5 value 0.171896
## iter   6 value -0.019431
## iter   7 value -0.032940
## iter   8 value -0.034805
## iter   9 value -0.034832
## iter  10 value -0.034839
## iter  11 value -0.034841
## iter  12 value -0.034841
## iter  13 value -0.034841
## iter  14 value -0.034841
## iter  14 value -0.034841
## iter  14 value -0.034841
## final  value -0.034841
## converged
## initial  value -0.019789
## iter   2 value -0.019930
## iter   3 value -0.020075
## iter   4 value -0.020112
## iter   5 value -0.020114
## iter   6 value -0.020114
## iter   7 value -0.020114
## iter   8 value -0.020114
## iter   8 value -0.020114
## final  value -0.020114
## converged
```



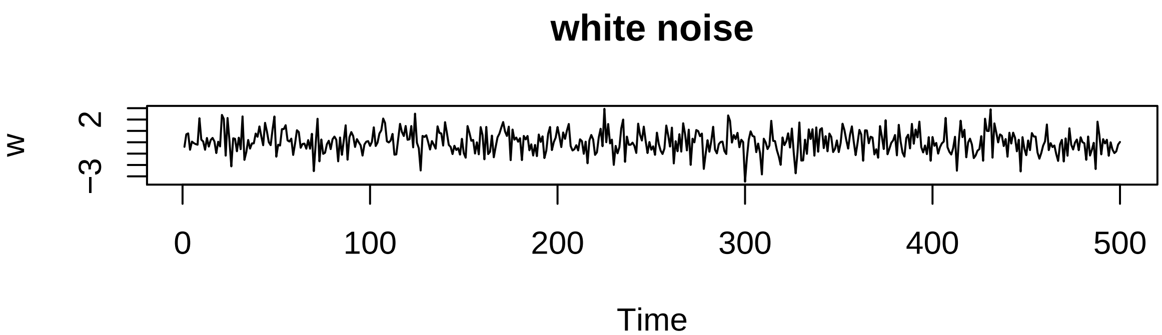

```
## $fit
##
## Call:
## stats::arima(x = xdata, order = c(p, d, q), seasonal = list(order = c(P, D,
##     Q), period = S), xreg = xmean, include.mean = FALSE, optim.control = list(trace = trc,
##     REPORT = 1, reltol = tol))
##
## Coefficients:
##          ar1      ar2   xmean
##       1.5353  -0.7814  0.8849
## s.e.  0.0443   0.0447  0.2797
##
## sigma^2 estimated as 0.9451:  log likelihood = -279.76,  aic = 567.53
##
## $degrees_of_freedom
## [1] 197
##
## $ttable
##       Estimate     SE  t.value p.value
## ar1     1.5353 0.0443  34.6233  0.0000
## ar2    -0.7814 0.0447 -17.4768  0.0000
## xmean   0.8849 0.2797   3.1642  0.0018
##
## $AIC
## [1] 0.9735514
##
## $AICc
## [1] 0.9845771
```

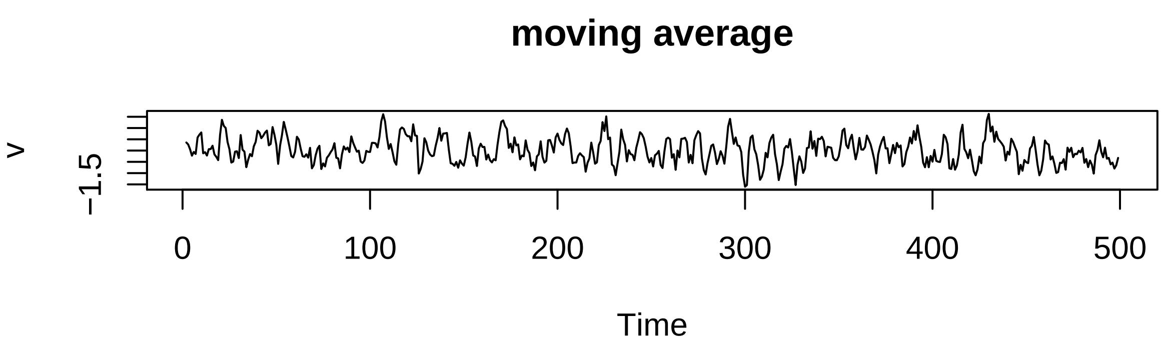
```
##
## $BIC
## [1] 0.0230262
```

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

Página 13 Moving Average

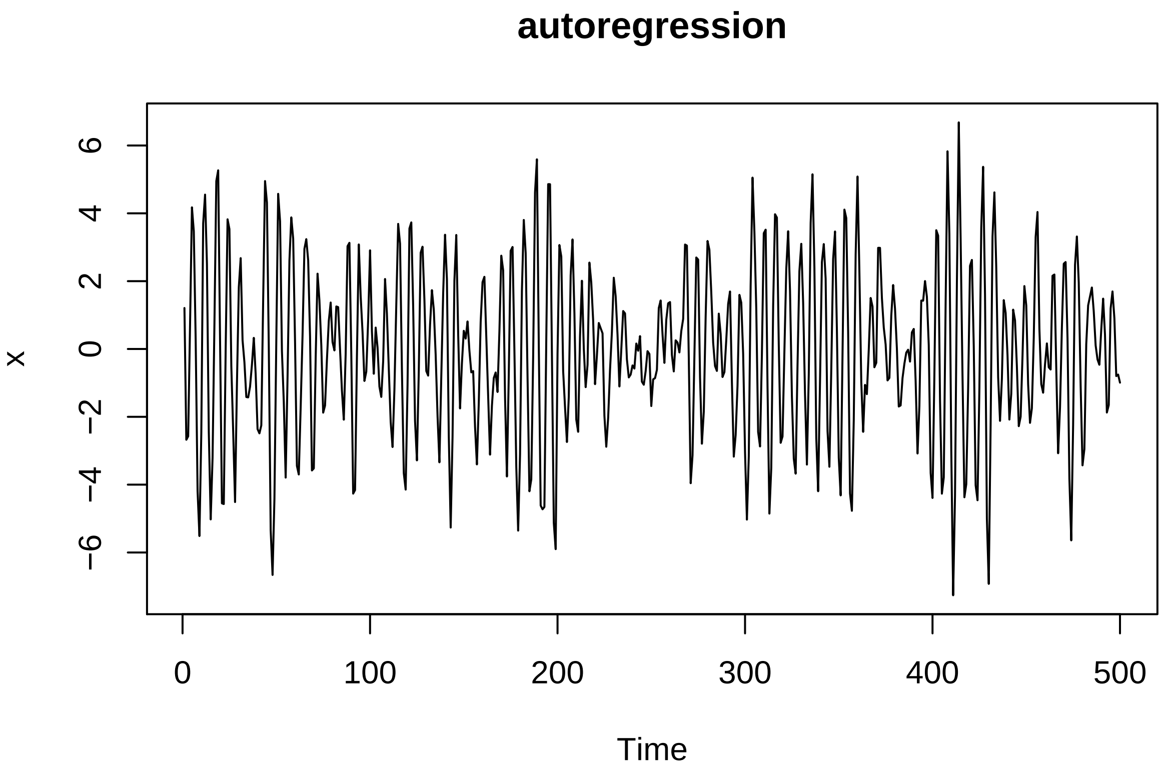
```
w = rnorm (500,0,1)
v = filter(w, sides = 2, rep(1/3,3))
par(mfrow=c(2,1))
plot.ts(w,main = "white noise")
plot.ts(v, main = "moving average")
```





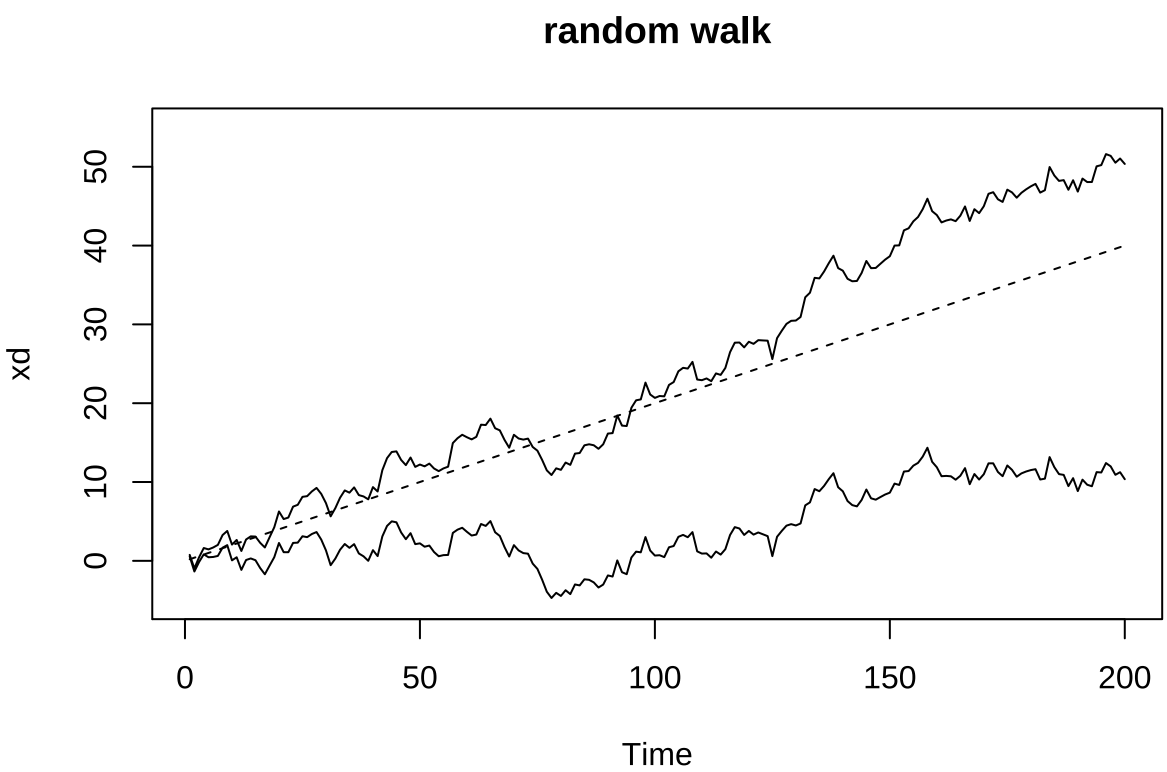
Autoregesivo

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



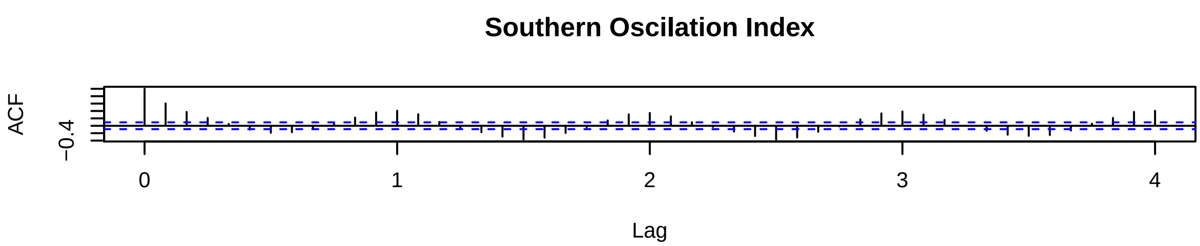
Autoregesivo con drift

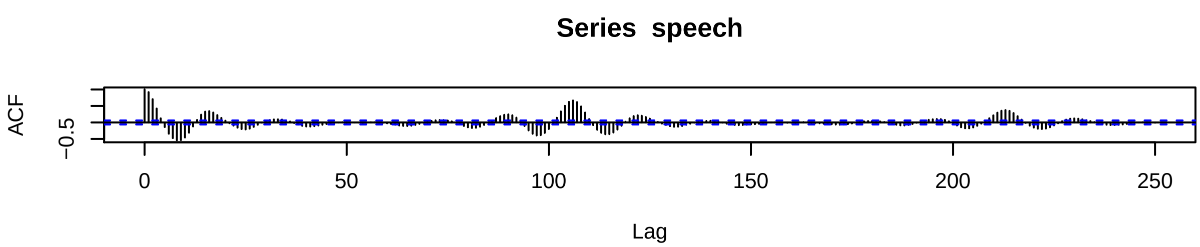
```
set.seed(154)
w = rnorm(200,0,1); x = cumsum (w)
wd = w + .2; xd = cumsum(wd)
plot.ts(xd, ylim = c(-5,55), main = "random walk")
lines(x); lines(0.2*(1:200), lty = "dashed")
```



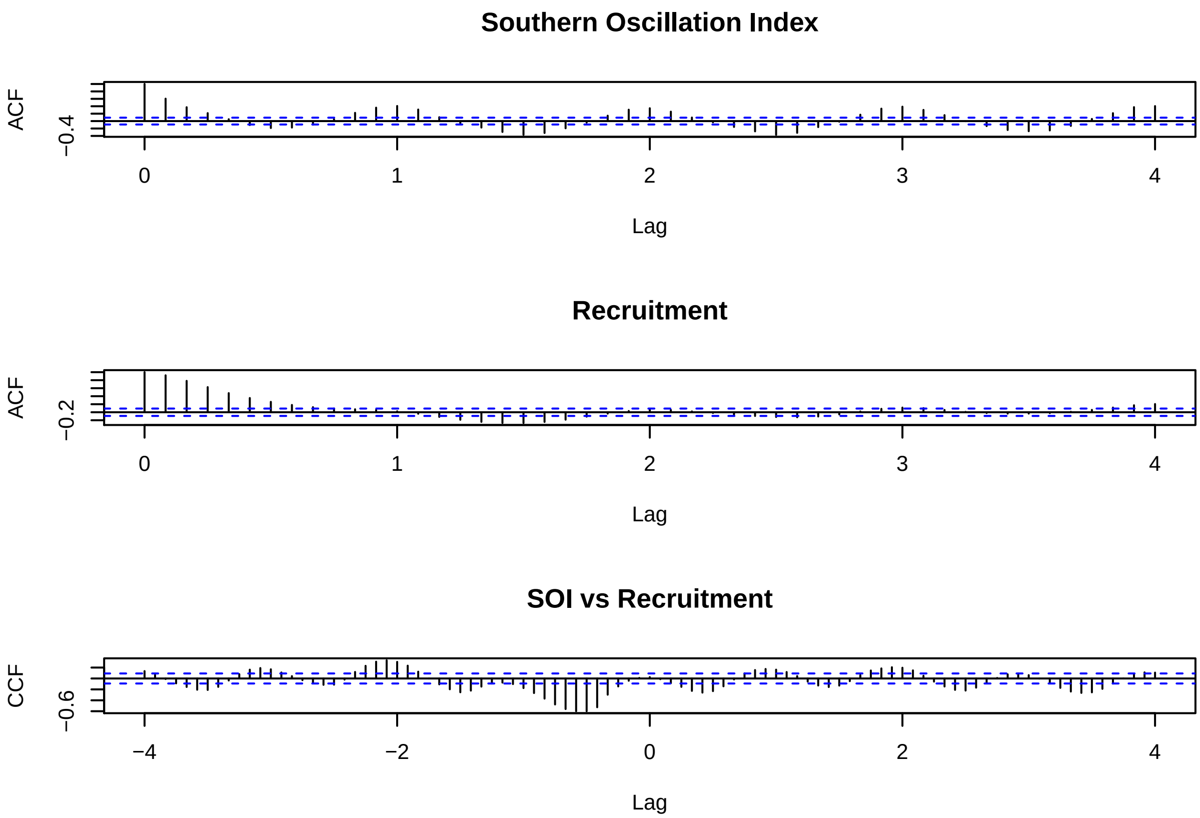
ACF

```
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")
```




```r
# Varios vectores

persp(1:64, 1:36, soiltemp, phi=30, theta=30, scale=FALSE, expand=4, ticktype="detailed", xlab="rows", y

plot.ts(rowMeans(soiltemp), xlab="row", ylab="Average Temperature")
```