Week 2

- Describe a time series via time plot. Analyze time series data through software
- Recognize simple Autocorrelation Functions (ACFs).
- Produce random walk and form simple moving averages for datasets.

Objectives

- Define a time series
- Get familiar with 'astsa' package

Definition - Time series is a dataset collected through time.

Correlation - Sampling adjacent points in time introduce a correlation.

Objectives

- See some examples of time series.
- Produce meaningful time plots.

```
require(astsa)
help(astsa)
```

astsa-package {astsa}	R Documentation

Applied Statistical Time Series Analysis (more than just data)

Description

Contains data sets and scripts for analyzing time series in both the frequency and time domains including state space modeling as well as supporting the texts Time Series Analysis and Its Applications: With R Examples (4th ed, 2017) and Time Series: A Data Analysis Approach Using R, (1st ed, 2019).

Details

Package:	astsa
Type:	Package
Version:	2.0
Date:	2023-01-10
License:	GPL-3
LazyLoad:	yes
LazyData:	yes

Author(s)

David Stoffer <stoffer@pitt.edu>

References

You can find demonstrations of astsa capabilities at FUN WITH ASTSA.

The most recent version of the package can be found at https://github.com/nickpoison/astsa/.

In addition, the News and ChangeLog files are at https://github.com/nickpoison/astsa/blob/master/NEWS.md.

The webpages for the texts and some help on using R for time series analysis can be found at https://nickpoison.github.io/.

[Package *astsa* version 2.0]

```
Loading required package: astsa
```

```
help(jj)
```

jj {astsa}	R Documentation

Johnson and Johnson Quarterly Earnings Per Share

Description

Johnson and Johnson quarterly earnings per share, 84 quarters (21 years) measured from the first quarter of 1960 to the last quarter of 1980.

Format

The format is: Time-Series [1:84] from 1960 to 1981: 0.71 0.63 0.85 0.44 0.61 0.69 0.92 0.55 0.72 0.77 ...

Details

The data were provided (personal communication) by Professor Paul Griffin, `https://gsm.ucdavis.edu/profile/paul-griffin`, of the Graduate School of Management, University of California, Davis. This data set is also included with the R distribution as `JohnsonJohnson`.

References

You can find demonstrations of astsa capabilities at FUN WITH ASTSA.

The most recent version of the package can be found at https://github.com/nickpoison/astsa/.

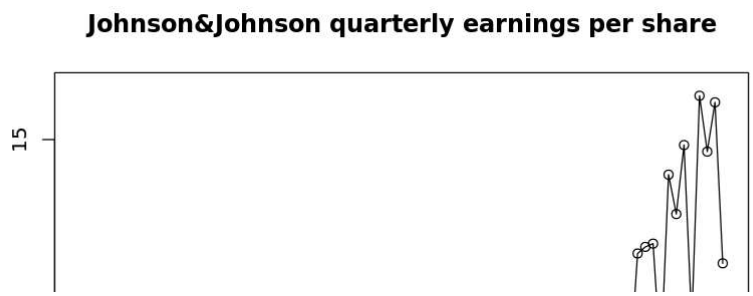
In addition, the News and ChangeLog files are at https://github.com/nickpoison/astsa/blob/master/NEWS.md.

The webpages for the texts and some help on using R for time series analysis can be found at https://nickpoison.github.io/.

[Package *astsa* version 2.0]

```
# already a time series, if not -- plot.ts()
# plotting

plot(jj, type='o', main='Johnson&Johnson quarterly earnings per share',
```

Download




```
help(flu)
```

flu {astsa}	R Documentation

Monthly pneumonia and influenza deaths in the U.S., 1968 to 1978.

Description

Monthly pneumonia and influenza deaths per 10,000 people in the United States for 11 years, 1968 to 1978.

Usage

```
data(flu)
```

Format

The format is: Time-Series [1:132] from 1968 to 1979: 0.811 0.446 0.342 0.277 0.248 ...

References

You can find demonstrations of astsa capabilities at FUN WITH ASTSA.

The most recent version of the package can be found at https://github.com/nickpoison/astsa/.

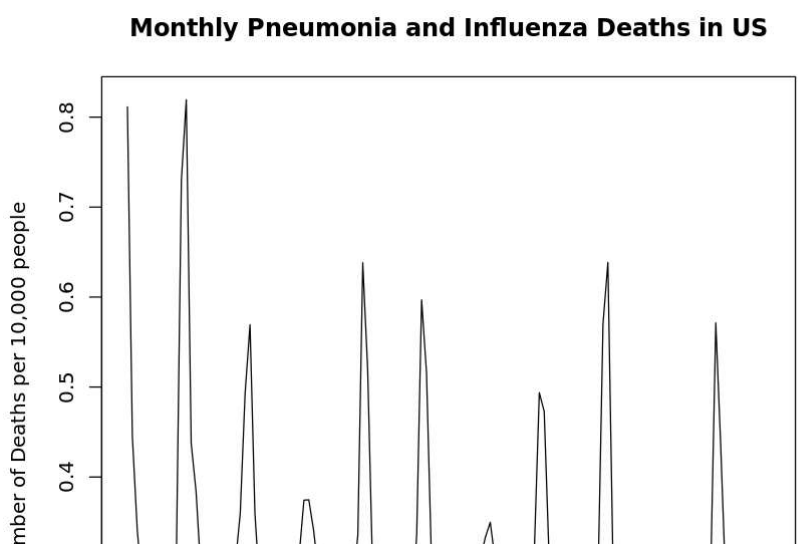
In addition, the News and ChangeLog files are at https://github.com/nickpoison/astsa/blob/master/NEWS.md.

The webpages for the texts and some help on using R for time series analysis can be found at https://nickpoison.github.io/.

[Package *astsa* version 2.0]

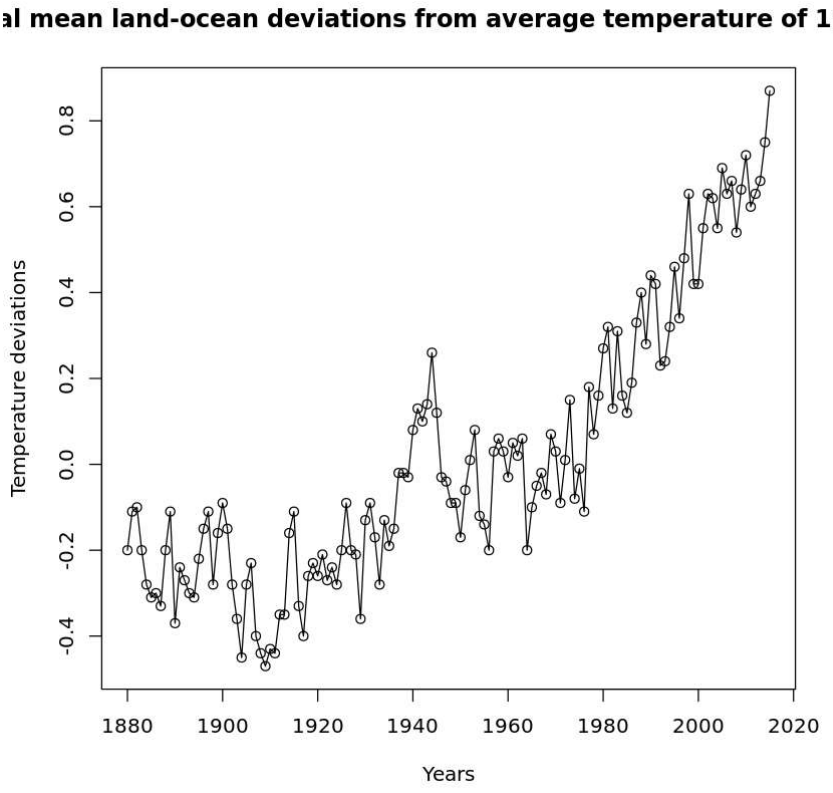
```
plot(flu, main='Monthly Pneumonia and Influenza Deaths in US', ylab='Num
```

Download




```
plot(globtemp, main='Global mean land-ocean deviations from average temp
```

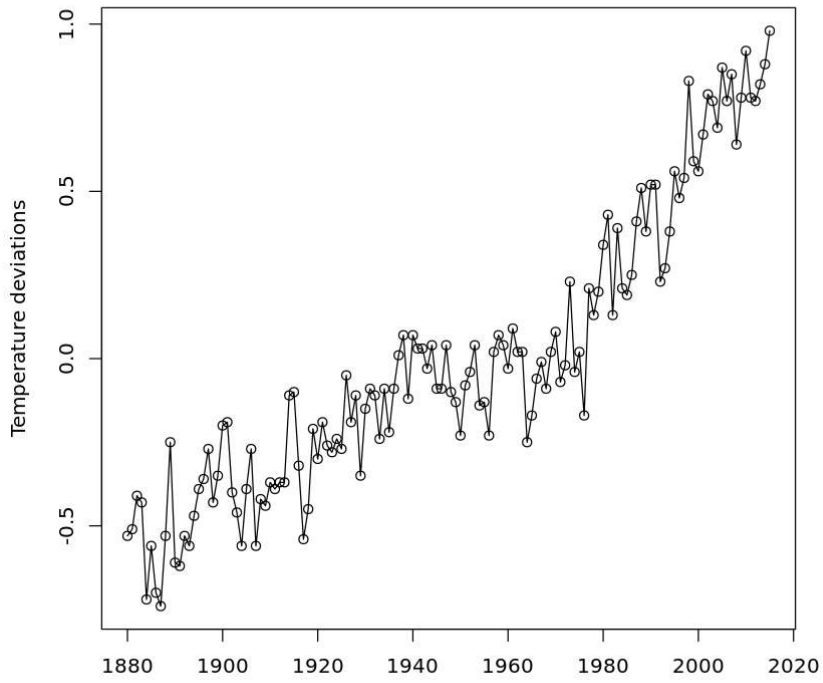
Download




```
plot(globtempl, main='Global mean land(only) deviations from average tem
```

Download

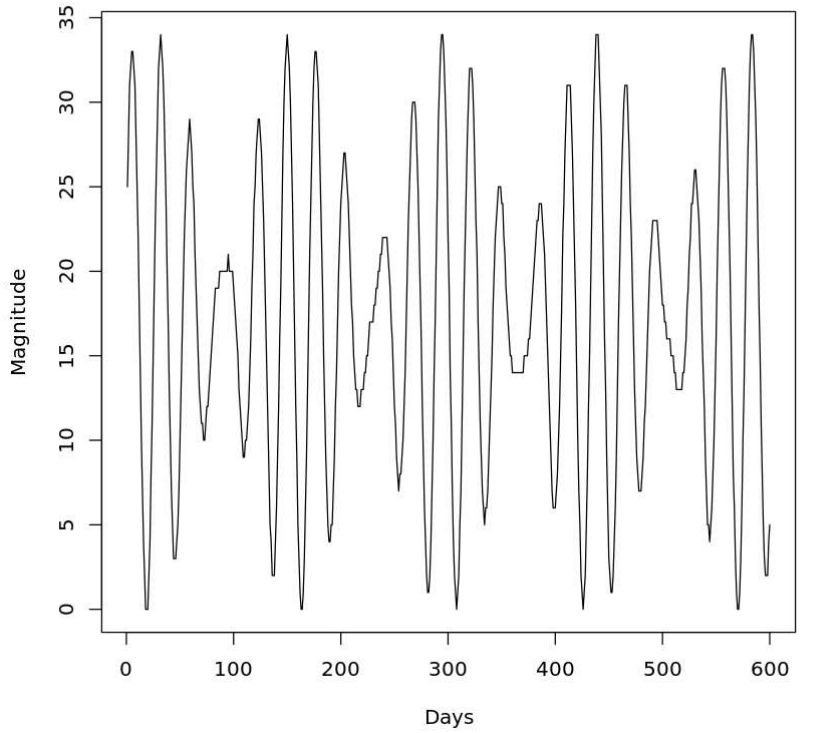
al mean land(only) deviations from average temperature of 19




```
plot(star, main='The magnitude of a star taken at midnight for 600 conse
```

Download

The magnitude of a star taken at midnight for 600 consecutive



What We've Learned

- Time series exist in variety of areas
- How to produce meaningful time plots

Objectives

Get some intuition for (weak) stationary time series

Stationary time series means --

- No systematic change in mean i.e., No trend
- No systematic change in variation
- No periodic fluctuations
- The properties of one section of a data are much like the properties of the other sections of the data

For a non-stationary time series, we will do some transformations to get stationary time series

What We've Learned

In a (weak) stationary time series, there is no

- systematic change in mean(no trend)
- systematic change in variance
- periodic variations

Objectives

- recall random variables and covariance of two random variables
- characterize time series as a realization of a stochastic process
- define autocovariance function

What We've Learned

- the definition of a stochastic processes
- how to characterize time series as realization of a stochastic process
- how to define autocovariance function of a time series

Objectives

- Recall the covariance coefficient for a bivariate data set
- Define autocovariance coefficients for a time series
- Estimate autocovariance coefficients of a time series at different lags

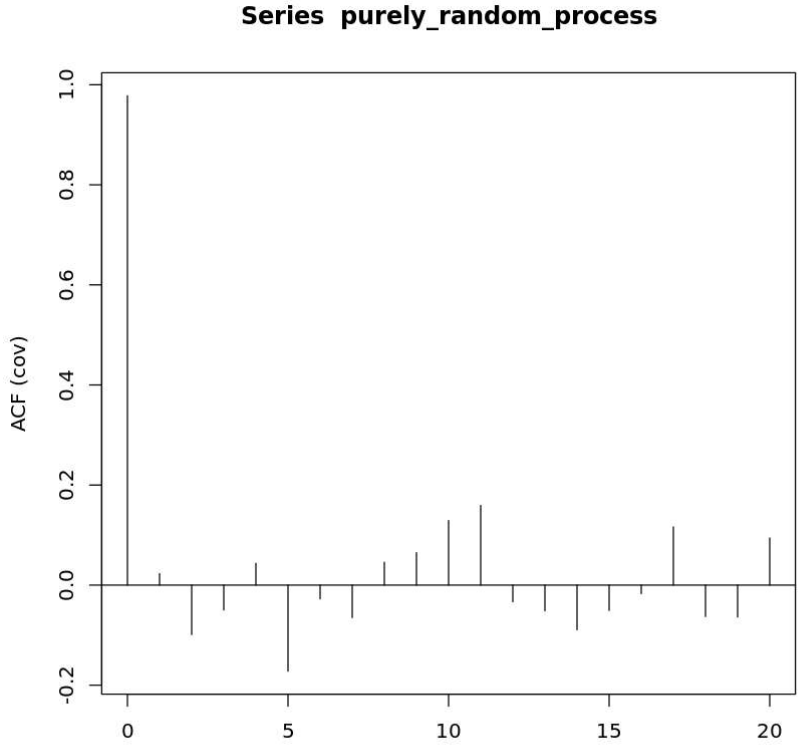
```
purely_random_process=ts(rnorm(100))
print(purely_random_process)
```

```
Time Series:
Start = 1
End = 100
Frequency = 1
  [1]  0.46922294  0.90481651 -0.76030030  0.88655981 -0.39371520 -0
  [7] -2.24136525 -0.45892182 -1.48504308 -0.87842584  1.36309336  2
 [13] -0.89257166  0.56027733  0.78511857 -2.59399965  0.15966634 -1
 [19]  0.55025082  0.01730218  2.27680264  0.07170792  1.01591739 -1
 [25]  0.44981431  0.86580419 -0.38015224 -0.20320824 -0.83955224  0
 [31]  0.93439110  0.09245820 -0.91631295 -0.52417078 -0.57043428 -0
 [37]  0.03972930  1.33457845 -1.06243850  0.61655987 -0.01710874  0
 [43]  1.58401360 -0.45968121 -1.66940883 -0.39419578 -0.55531380  1
 [49]  0.24680239 -0.80957971  0.82000824  1.18055591 -0.57072910  0
 [55]  1.68879559  0.66931446 -0.09146958  0.56545765  1.56107247 -0
 [61] -0.93340140  0.27188747  0.73410879  0.11200668  0.52196009  0
 [67] -0.11291226  1.58006963  1.42540731  1.93936951 -0.71412971 -0
 [73]  0.17012951 -0.45955285  0.44354304  0.20151191 -1.20458438  0
 [79]  1.50977099  0.99178058 -1.38408774  0.04801623 -0.60402352  1
 [85]  0.04508681 -0.73314166 -0.99297405  1.08356602 -0.21415416  0
```

```
# get autocovariance coefficient for every lag

(acf(purely_random_process, type='covariance'))
```

```
Autocovariances of series ‘purely_random_process’, by lag

     0      1      2      3      4      5      6      7
0.9780 0.0229 -0.0989 -0.0497 0.0434 -0.1719 -0.0275 -0.0649 0.045
    10     11     12     13     14     15     16     17     1
0.1290 0.1594 -0.0331 -0.0514 -0.0891 -0.0505 -0.0169 0.1161 -0.062
    20
0.0942
```

Download



What We've Learned

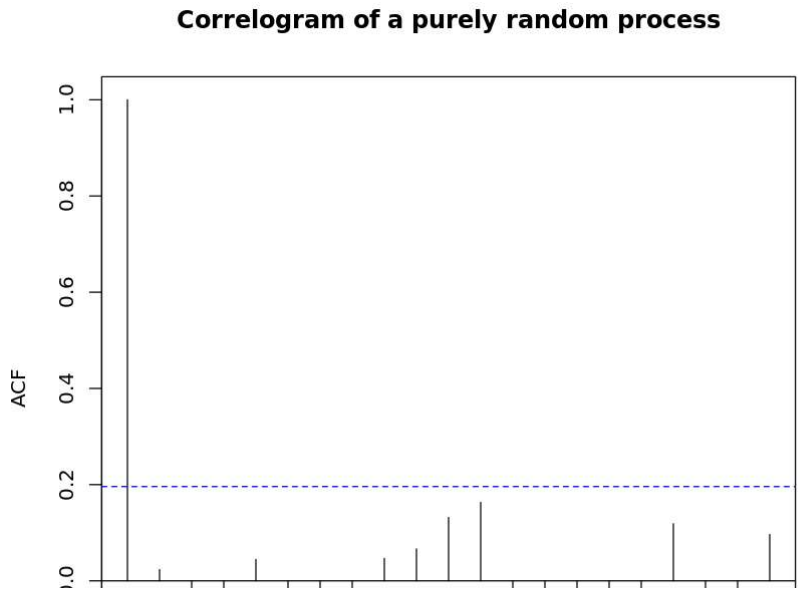
- Definition of autocovariance coefficients at different lags
- Estimate autocovariance coefficients of a time series using acf() routine

Objectives

- Define the autocorrelation function
- Obtain corellograms using acf() routine
- Estimate autocorrelation coefficients at different lags using acf() routine

```
# Correlogram - autocorrelation coefficients

acf(purely_random_process, main='Correlogram of a purely random process'
```

Download

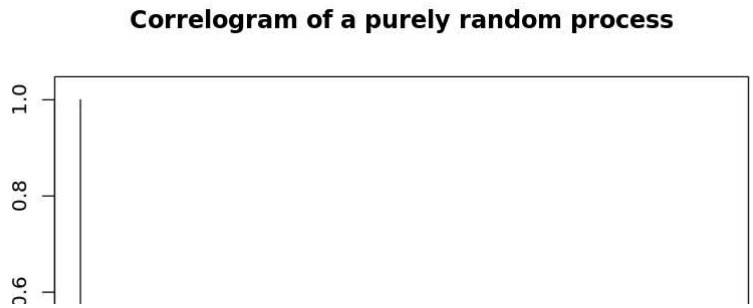


Plot shows that for a random process there is very less correlation between lag 0 and lag 1,2,3...n

```
# putting parenthesis also gives us autocorrelation coefficients

(acf(purely_random_process, main='Correlogram of a purely random process
```

```
Autocorrelations of series ‘purely_random_process’, by lag

     0      1      2      3      4      5      6      7      8      9
 1.000  0.023 -0.101 -0.051  0.044 -0.176 -0.028 -0.066  0.047  0.066
    11     12     13     14     15     16     17     18     19     20
 0.163 -0.034 -0.053 -0.091 -0.052 -0.017  0.119 -0.064 -0.065  0.096
```

Download



What We've Learned

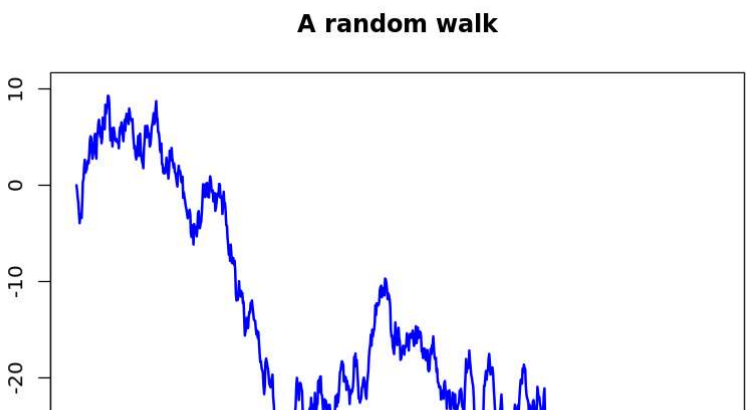
- Definition of the autocorrelation function (ACF)
- How to produce correlograms using acf() routine
- How to estimate the autocorrelation coefficients at different lags using acf() routine

Objectives

- Get Familiar with the random walk model
- Simulate a random walk in R
- Obtain the correlogram of a random walk
- See the difference operator in action

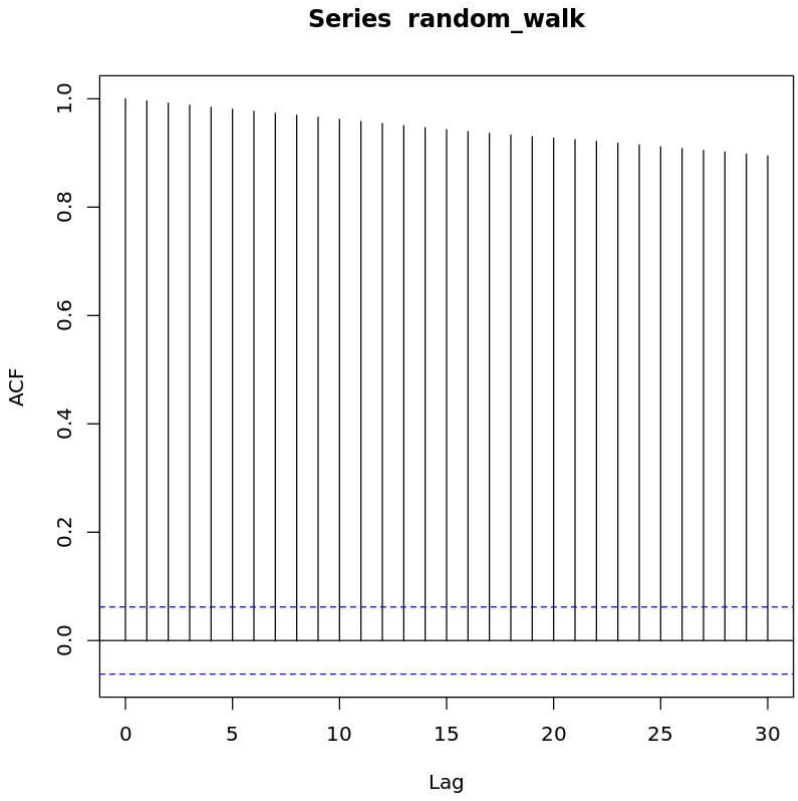
```
## Random Walk Time Series

x=NULL
x[1]=0
for(i in 2:1000){
    x[i]=x[i-1]+rnorm(1)
}
random_walk=ts(x)
plot(random_walk, main='A random walk', ylab=' ', xlab='Days', col='blue
```

Download




```
# Random walk is not a stationary time series
# so ACF plot doesnt make any sense
# still calculating ACF

acf(random_walk)
```

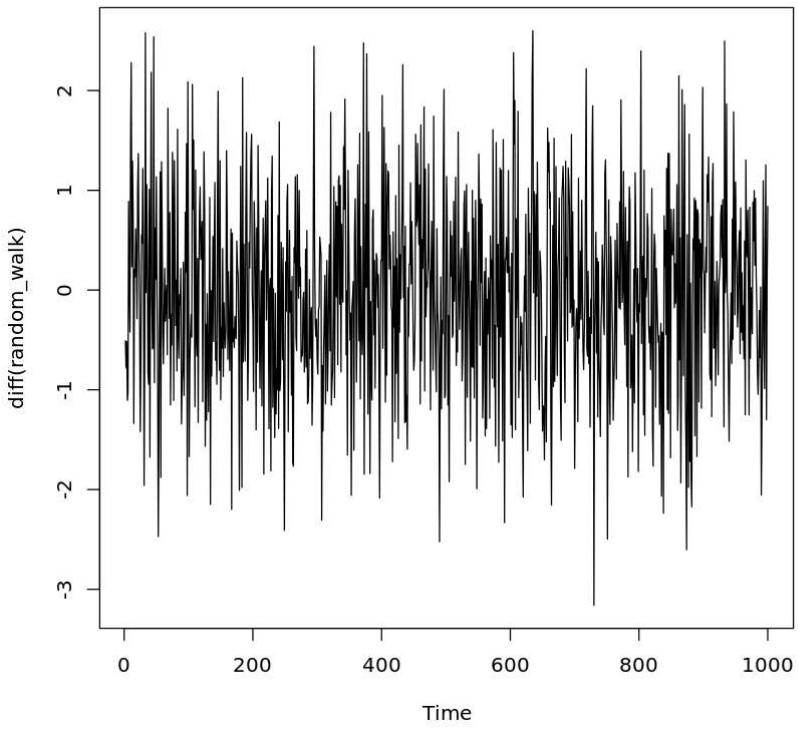
Download



Trend is not decaying below 0 which clearly shows that the time series is not stationary

```
# we will do 1 lag differencing to remove the trend on random walk
# doing this we get a purely random process which is stationary

plot(diff(random_walk))
```

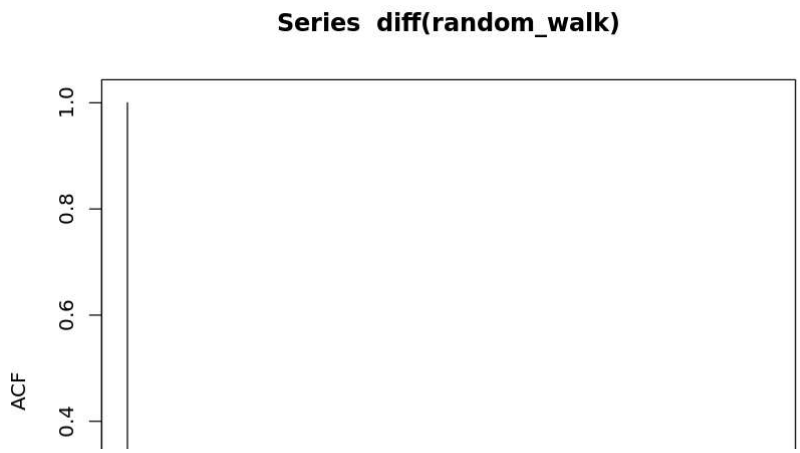
Download



The plot shows the white noise

```
# lets plot the ACF for this time series
acf(diff(random_walk))
```

Download



This plot is similar to the one we saw before which has very less autocorrelation between lag 0 and lag 1,2,3...

Difference operator

- diff() to remove the trend
- Plot and ACF differenced time series

What We've Learned

- Random Walk model
- How to simulate a random walk in R
- How to get stationary time series from a random walk using diff() operator

Objective

- Identify Moving average processes

What We've Learned

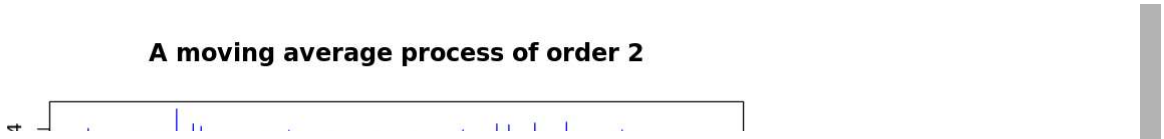
- How to identify Moving average processes MA(q) where q is the order

Objective

- Simulate a moving average process
- Interpret correlogram of a Moving average process

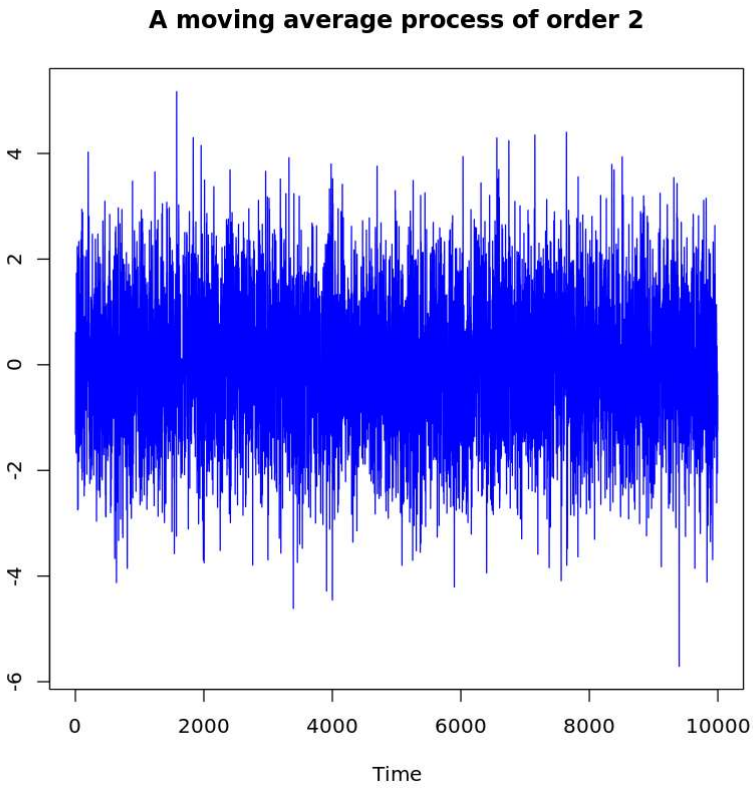
```
# MOVING AVERAGE PROCESS MA(2)

# Generate noise
noise=rnorm(10000)

# Introduce a variable
ma_2=NULL

# Loop for generating MA(2) process
for(i in 3:10000){
    ma_2[i]=noise[i] + 0.7*noise[i-1] + 0.2*noise[i-2]
}

# Shift data to left by 2 units
moving_average_process=ma_2[3:10000]

# Put time series structure on a vanilla data
moving_average_process=ts(moving_average_process)

# Partition output graphics as a multi frame of 2 rows and 1 column
par(mfrow=c(2,1))

# plot the process and plot its ACF
plot(moving_average_process, main='A moving average process of order 2',
acf(moving_average_process, main='Corellogram of a moving average proces
```

Download



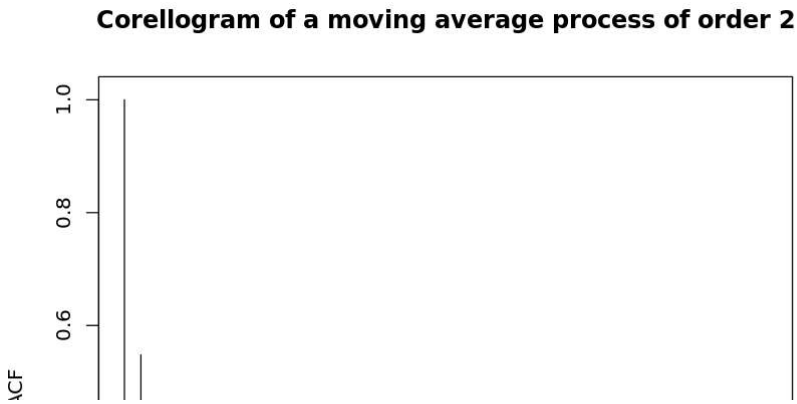

```
# MA(2) plot
plot(moving_average_process, main='A moving average process of order 2',
```

Download




```
# Correlogram of MA(2)
acf(moving_average_process, main='Corellogram of a moving average proces
```

Download



In the ACF plot above ACF value cuts off after lag 2 which shows our data is MA(2) process

What We've Learned

- How to simulate MA processes in R
- That ACF of MA(q) cuts off at lag q