Time Series

Fundamentals

Assumptions in Time Series Algorithms

- Consecutive Observations in the series are equally spaced
- Series is indexed on specific period of time. e.g. Weekly, Daily, Yearly etc.
- There aren’t any missing values

Object **ts**

- Data in the form of data frame / vector / matrix usually is not acceptable for time series functions in R
- For making the data compatible to be accepted for time series functions it need to be converted into objects like **ts** or **xts**
- We will be covering the object of class **ts**

Creating **ts** object

- Function *ts()* is used to create an object of class **ts**

Syntax : ts(data, start, end ,frequency,…)

Where

data : a vector or matrix or data frame of the observed time-series values

start : the time of the first observation

end : the time of the last observation

frequency : the number of observations per unit of time

Example of monthly **ts**

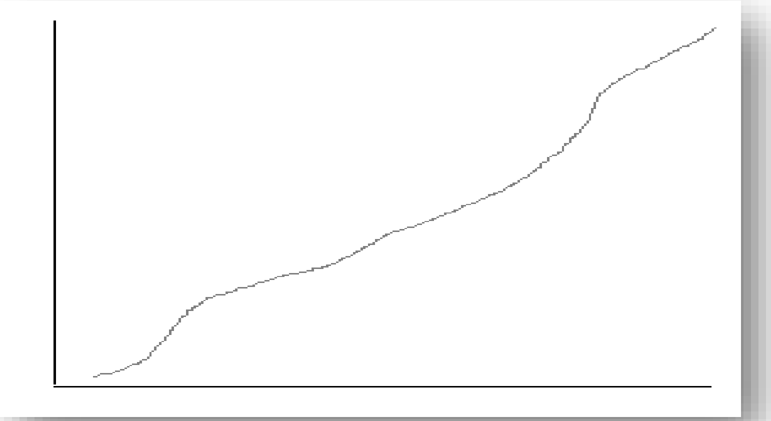
	Date	Value
1	30-04-1968	39.100
2	31-05-1968	42.000
3	30-06-1968	40.950
4	31-07-1968	38.900
5	31-08-1968	39.850
6	30-09-1968	39.700
7	31-10-1968	39.200
8	30-11-1968	39.850

```
> BUNDESBANK_ts <- ts(BUNDESBANK$Value,start=c(1968,4), frequency = 12)
> BUNDESBANK_ts
         Jan     Feb     Mar     Apr     May     Jun     Jul
1968                          39.100  42.000  40.950  38.900
1969  42.550  42.775  43.100  43.600  43.150  41.225  41.450
1970  34.980  35.000  35.300  35.850  35.500  35.510  35.290
1971  38.000  38.790  38.800  39.600  40.800  40.200  42.475
1972  46.950  48.400  48.375  49.500  59.300  64.100  68.900
1973  66.000  85.300  90.250  90.700 114.500 123.500 115.200
1974 133.250 169.500 173.000 168.500 156.500 146.750 154.000
1975 176.250 181.750 177.750 167.400 167.750 166.000 166.400
1976 128.000 132.300 129.500 128.150 125.250 123.800 112.400
```

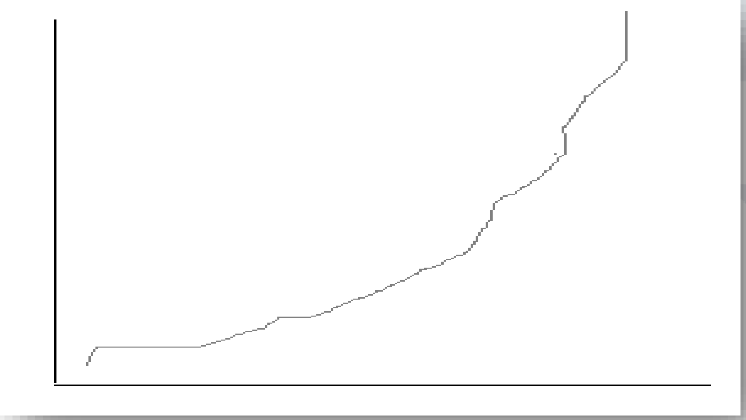
```
> start(BUNDESBANK_ts) # Start Time Point
[1] 1968    4
> end(BUNDESBANK_ts) # End Time Point
[1] 2016    4
> # Fraction of time between the observations, for monthly - 1/12, for quarterly - 1/4
> deltat(BUNDESBANK_ts)
[1] 0.08333333
```

Types of Trends

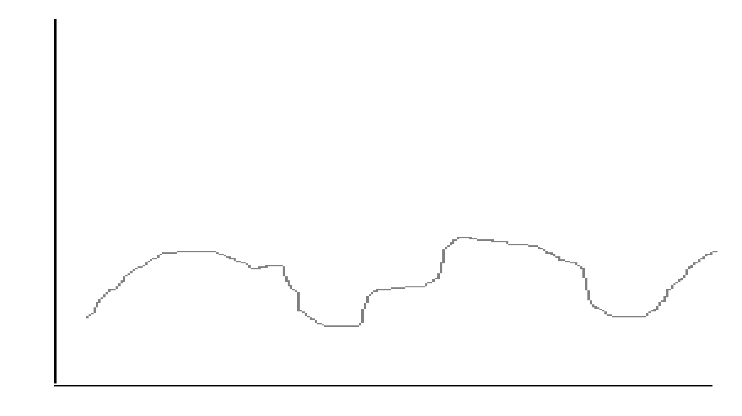
Linear



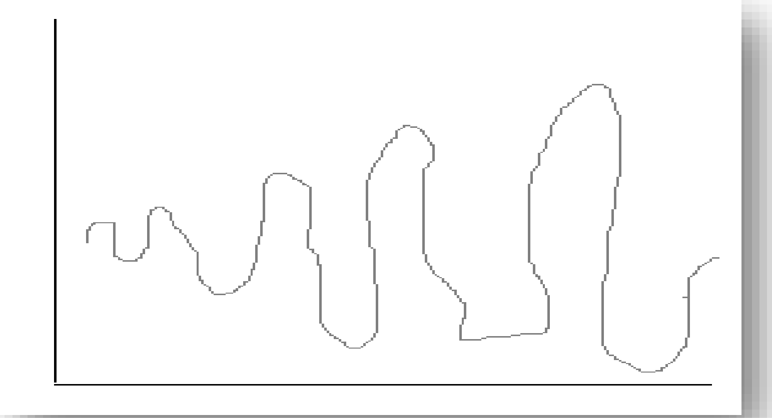
Rapid Growth



Periodic



Varying Variance



Some Tranformations

- log: The log() function can linearize the rapid growth trend. It can also stabilize the varying variance series. It is only for positive values.
- diff: The diff() function can remove the linear trends. It can also remove periodic trends.

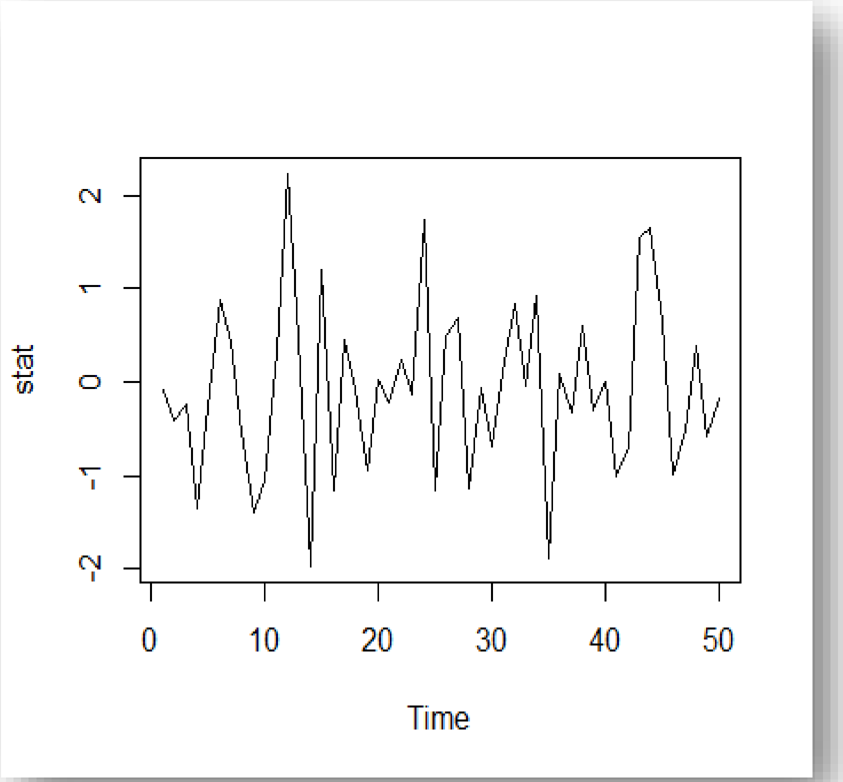
Sane's STATS Academy of Statistics

Stationary Process

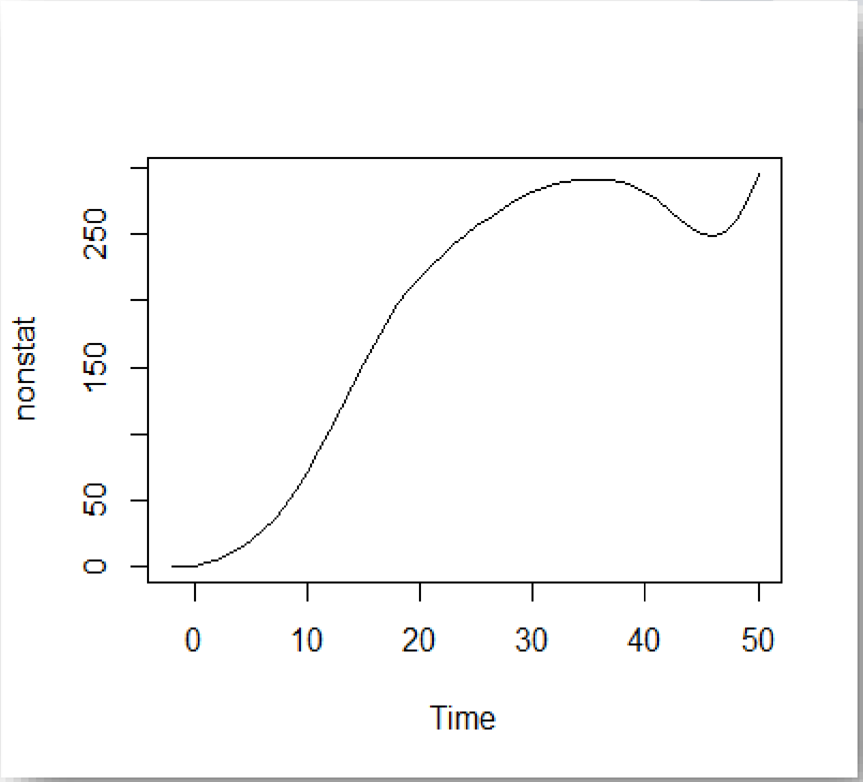
- Stationary process is that stochastic (probabilistic) process whose joint probability distribution does not change when shifted in time.
- In our context of time series, it is that time series whose mean and variance do not change over time.
- White Noise Model is the simplest example of Stationary series.
- For weak stationarity, covariance of y_t and y_s is constant for all $|t-s|=h$, for all h. e.g. $Cov(y_3, y_7) = Cov(y_{22}, y_{26})$

Stationary and Non-Stationary

Stationary



Non-Stationary



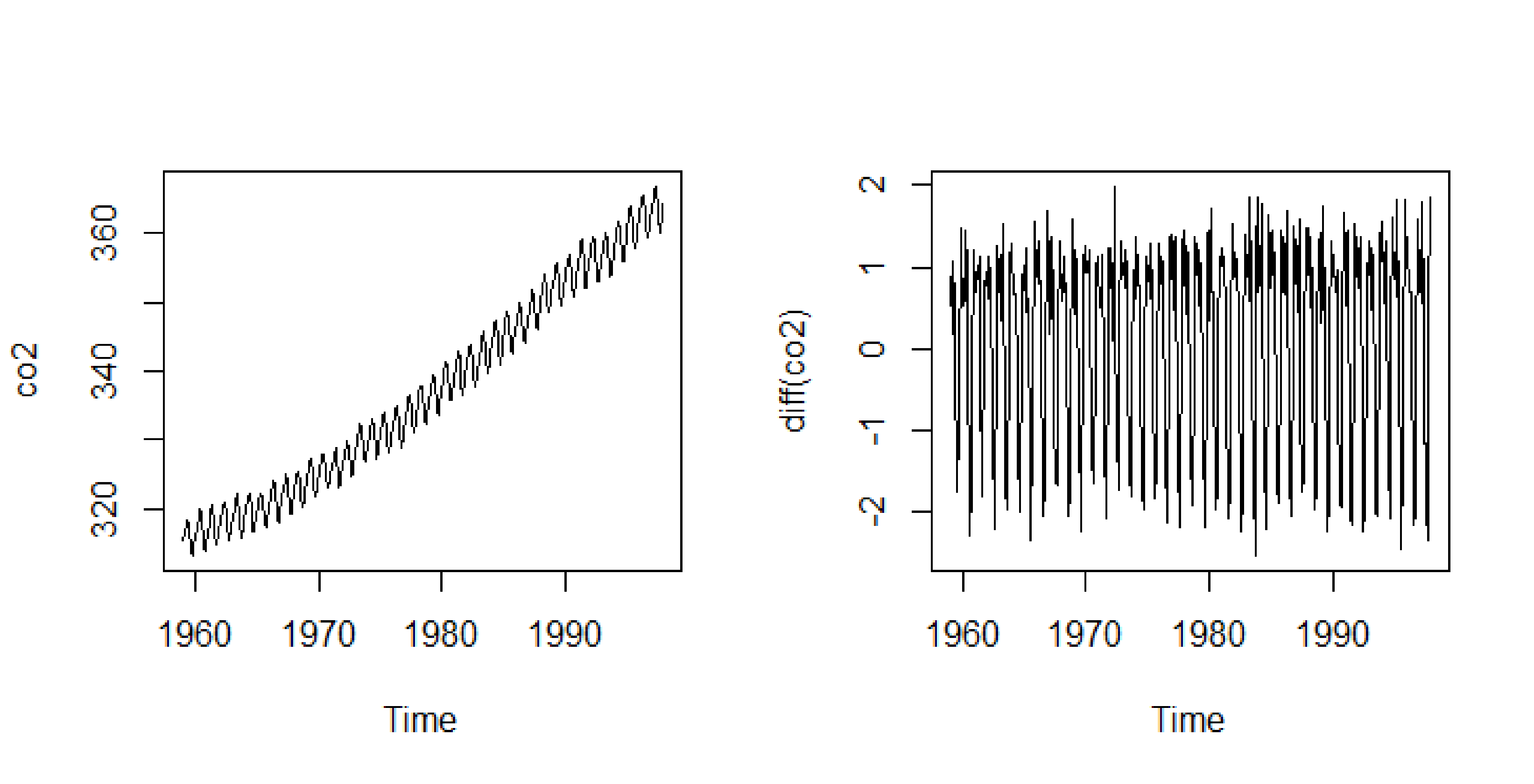
White Noise Model (WN Model)

- WN Model is a simple example of stationary process
- A weak White Noise has
  - A fixed constant mean
  - A fixed constant variance
  - No correlation of any time point value with any time point value

Random Walk (RW) Model

- RW Model is a simple example of non-stationary time series
- A random walk series has
  - No specific mean and variance
  - Strong dependence over time
- Changes or increments in RW series are white noise
- Random Walk Recursion: Today's value = Yesterday's Value + Noise
- In other words, $y_t = y_{t-1} + \in_t$, where $\in_t$ is white noise with mean zero
- RW Model has only one parameter i.e. variance of the white noise σ_ϵ^2

Example of RW Model



Random Walk with Drift

- Random Walk Recursion:
  $$Today's\,Value = Constant + Yesterday's\,Value + Noise$$
- In other words, $y_t = c + y_{t-1} + \in_t$, where $\in_t$ is white noise with mean zero
- This has two parameters, drift constant c and σ_ϵ^2

Model Simulations in R

- We can simulate the ARIMA type of models in R with the function arima.sim()

Syntax : arima.sim(model, n, …)

Where

model : List of components of ARIMA. It can be specified with list(order=c(p,d,q)), where p is autoregressive order, d is order of differencing, q is moving average order

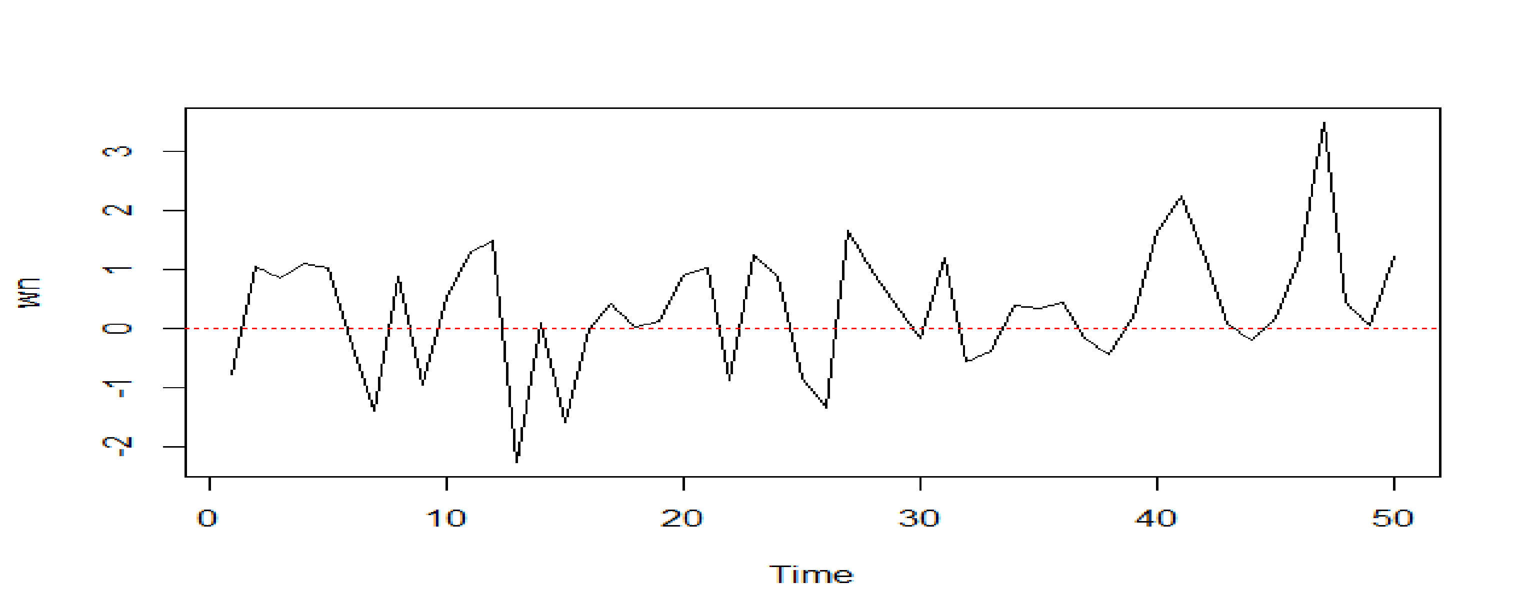
n : length of the series to be generated

Simulating WN and RW Models

- For simulating WN, we specify c(0,0,0) in the function call of arima.sim()
- For simulating RW, we specify c(0,1,0) in the function call of arima.sim()

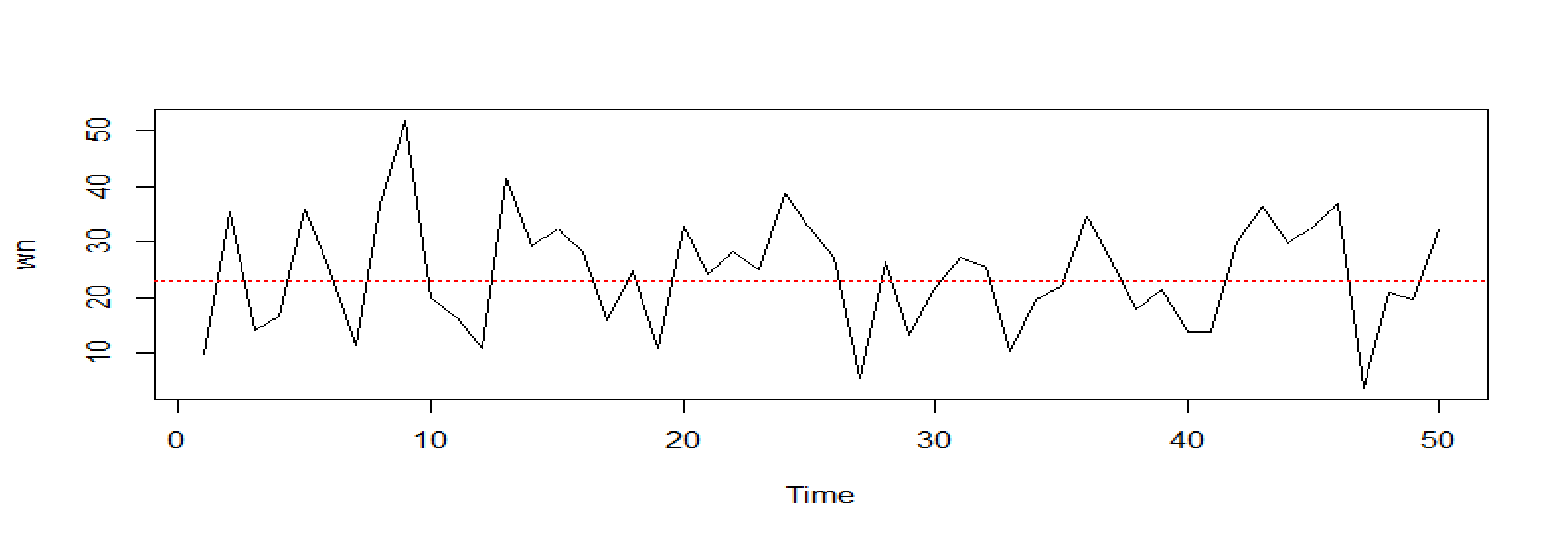
WN Simulation with Standard Normal Distribution

```
> wn <- arima.sim(model=list(order=c(0,0,0)), n = 50)
> ts.plot(wn) # Mean=0 and SD=1 by default
> abline(h=0, lty=3, col="red")
> mean(wn)
[1] 0.3554053
> sd(wn)
[1] 1.03506
```



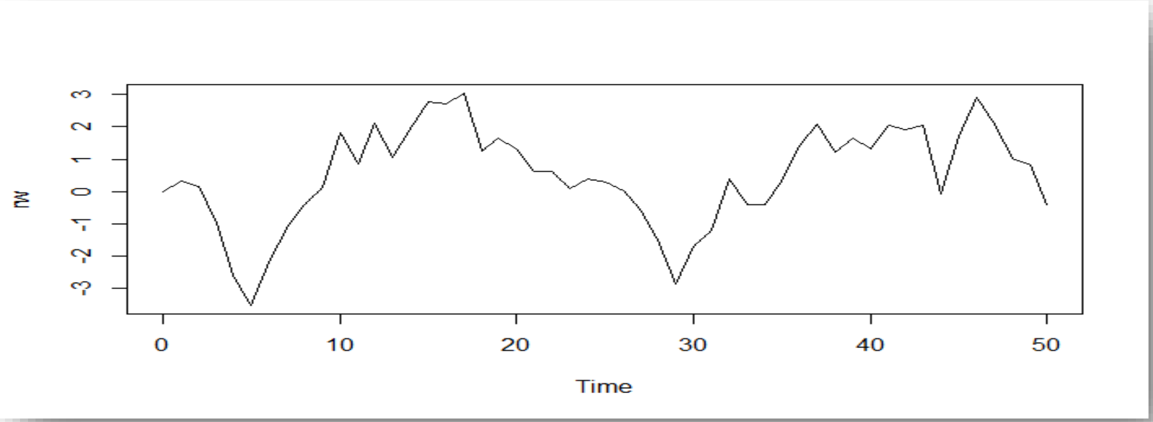
WN Simulation with Specified Mean & SD

```
> wn <- arima.sim(model = list(order=c(0,0,0)), n = 50, mean=23, sd=10)
> ts.plot(wn)
> abline(h=23, lty=3, col="red")
> mean(wn)
[1] 24.35173
> sd(wn)
[1] 10.15362
```

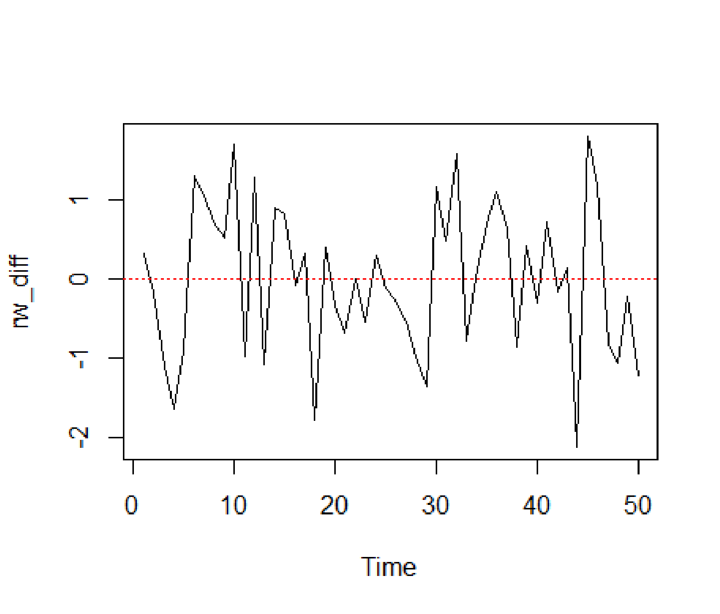


Simulating Random Walk without drift

```
> rw <- arima.sim(model=list(order=c(0,1,0)), n=50)
> ts.plot(rw)
```

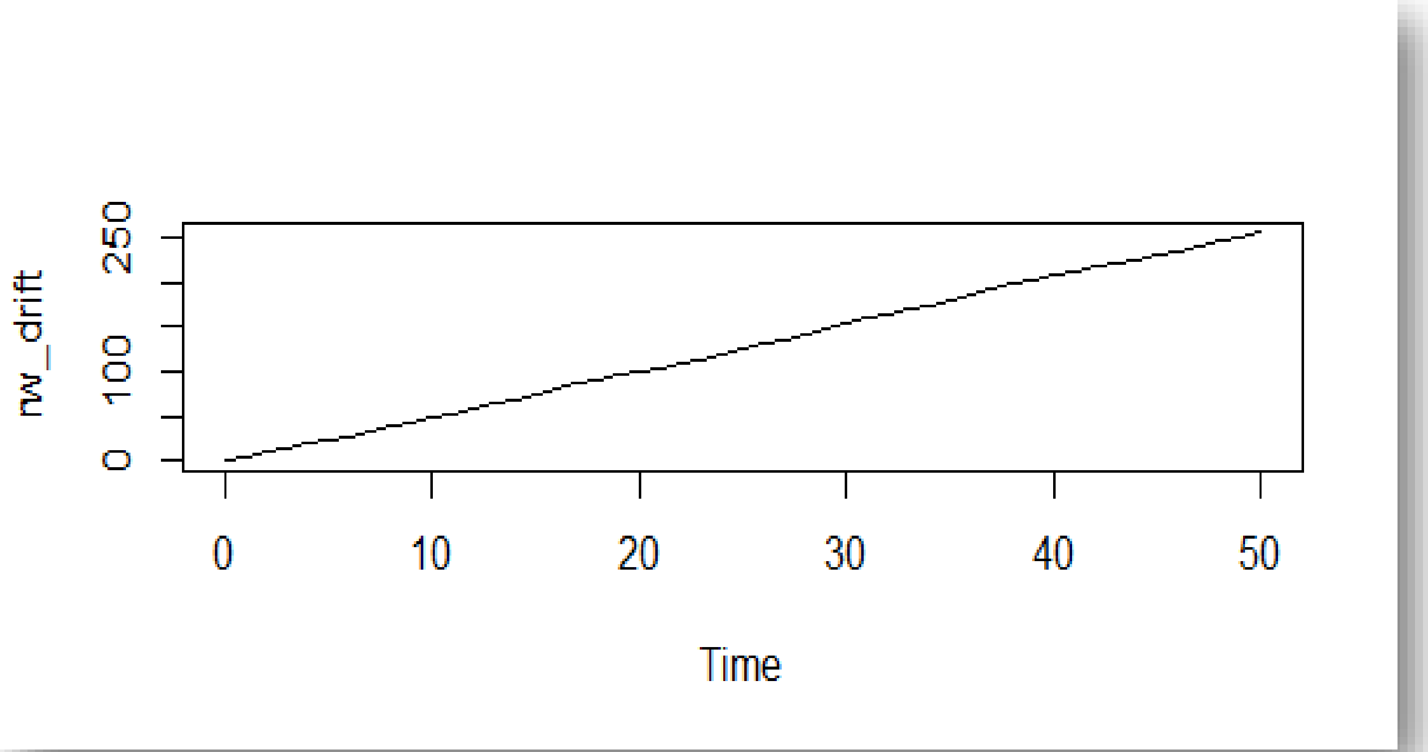



```
> rw_diff <- diff(rw)
> ts.plot(rw_diff)
> abline(h=0, lty=3, col="red")
> mean(rw_diff)
[1] -0.008369115
> sd(rw_diff)
[1] 0.9612737
```



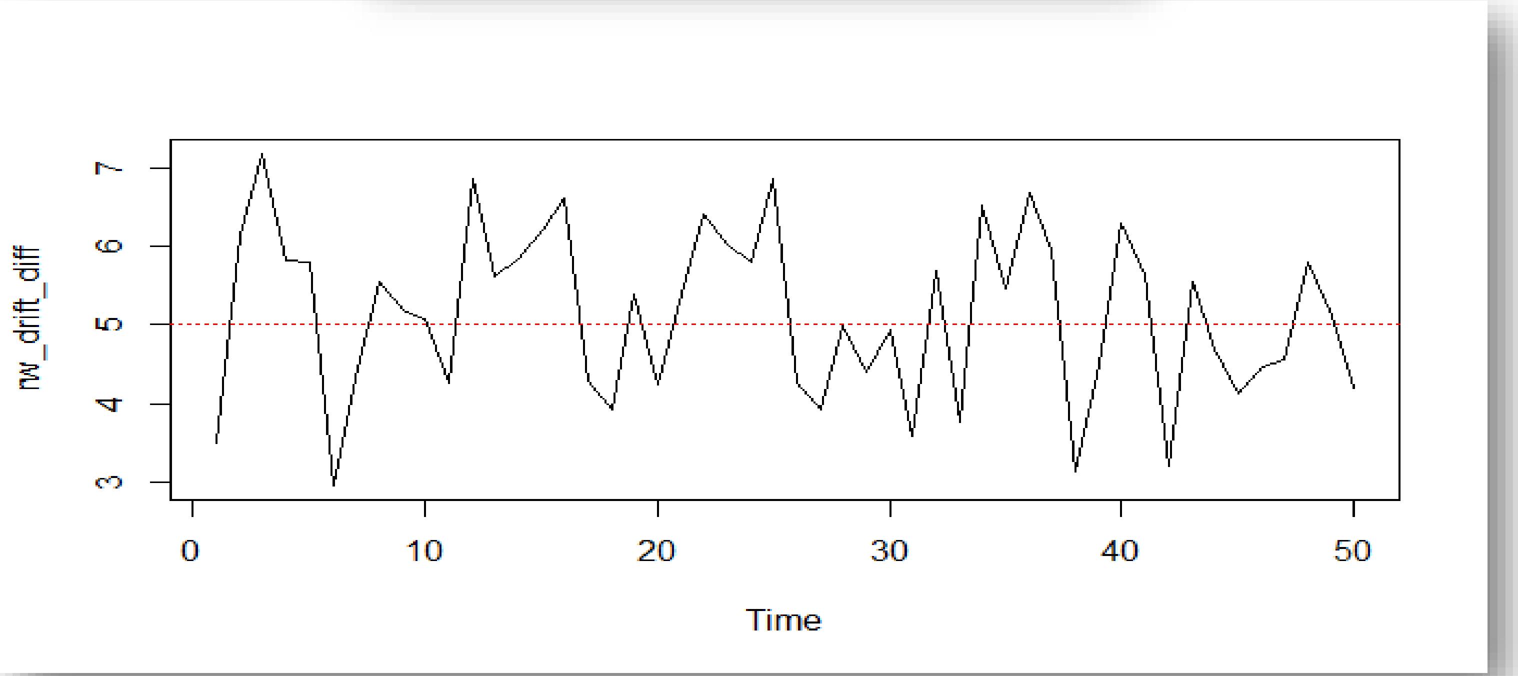
Simulating Random Walk with drift

```
> rw_drift <- arima.sim(model = list(order = c(0, 1, 0)), n = 50, mean = 5)
> ts.plot(rw_drift)
```



Simulating Random Walk with drift

```
> rw_drift_diff <- diff(rw_drift)
> ts.plot(rw_drift_diff)
> abline(h=5, lty=3, col="red")
> mean(rw_drift_diff)
[1] 5.135947
> sd(rw_drift_diff)
[1] 1.081396
```



Autocorrelation

What is Autocorrelation?

- Autocorrelation is correlation between the elements of a series and others from the same series separated from them by a given interval.
- Lag 1 Autocorrelation: Correlation of today’s value with yesterday’s value
- Lag 2 Autocorrelation: Correlation between today’s and day before yesterday’s values
- Lag k Autocorrelation: Correlation between Day 1 with Day k values

Autocorrelation in R

- Autocorrelation can be found with function acf() in R

Syntax : acf(x, lag.max, type, …)

Where

x : a univariate or multivariate numeric time series object or a numeric vector or matrix, or an "acf" object

lag.max : maximum lag at which to calculate the acf. Default is 10*log10(N/m) where N is the number of observations and m the number of series.

type : acf to calculate, can be correlation, covariance or partial. Default is correlation.

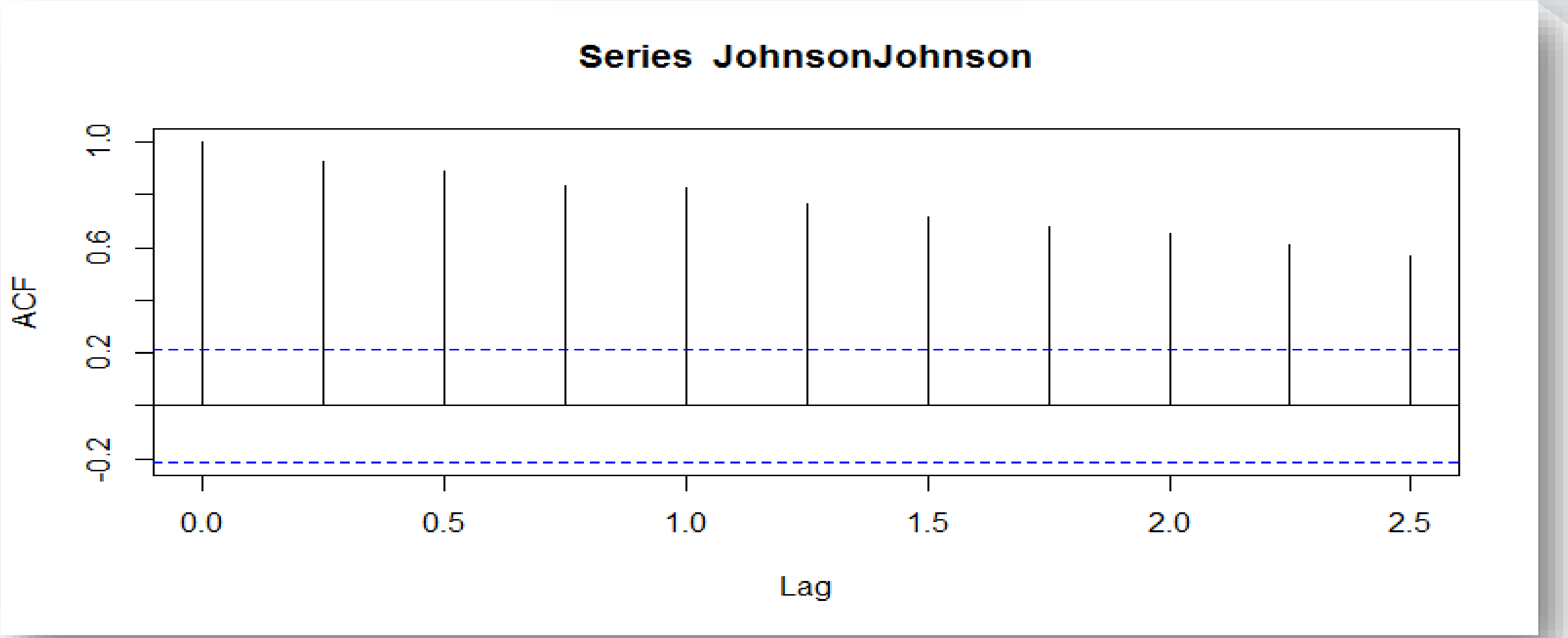
Calculating acf

```
> acf(JohnsonJohnson,10, plot = F)

Autocorrelations of series ‘JohnsonJohnson’, by lag

 0.00  0.25  0.50  0.75  1.00  1.25  1.50  1.75  2.00  2.25  2.50
1.000 0.925 0.888 0.833 0.824 0.764 0.718 0.675 0.654 0.608 0.564
```

- Output shows quarterly autocorrelations. Plot is rendered FALSE so that the function doesn’t produce graph. Here 10 is for maximum lags to produce.
- We observe that, the correlation goes on decreasing with the increase in the lag. This is not the case with every time series.

Plotting acf

```
acf(JohnsonJohnson,10)
```



- We observe here that as the lag goes on increasing, the correlation goes on decreasing

Autoregressive Models

AR Process

Autoregressive Model

- In this model, we consider that today's observation is regressed on yesterday's observation or any of the previous day's observation.
- Model:

$$Today's\ Value = Constant + Slope * Yesterday's\ Value + Noise$$

- R uses mean centered version of this model as

$$(Today's\ Value - Mean) = Slope * (Yesterday's\ Value - Mean) + Noise$$

- By notations, $y_t - \mu = \phi(y_{t-1} - \mu) + \epsilon_t$, where ϵ_t is a white noise with mean 0 with variance σ_ϵ^2 and ϕ and μ are the slope and mean respectively

AR Process

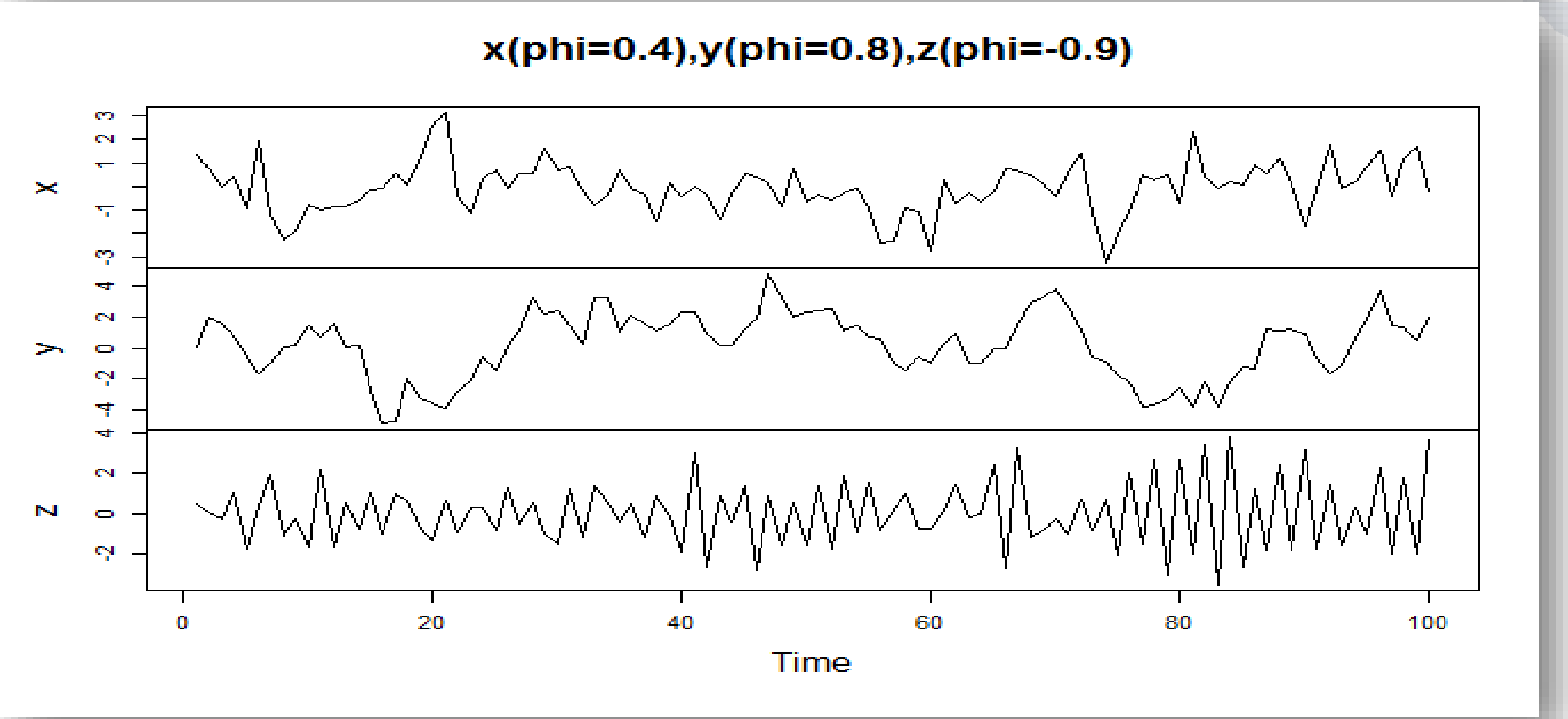
$$y_t - \mu = \phi(y_{t-1} - \mu) + \epsilon_t$$

- If slope $\phi = 0$ then $y_t = \mu + \epsilon_t$ and y_t will be white noise with mean μ and variance σ_ϵ^2
- If slope $\phi \neq 0$ then the process of $\{y_t\}$ is autocorrelated
- Large value of Ø implies greater dependency of current values with previous values
- Negative value of Ø implies oscillatory time series
- If $\mu = 0$ and slope $\phi = 1$, then $y_t = y_{t-1} + \epsilon_t$, which is a random walk process

Simulating AR process

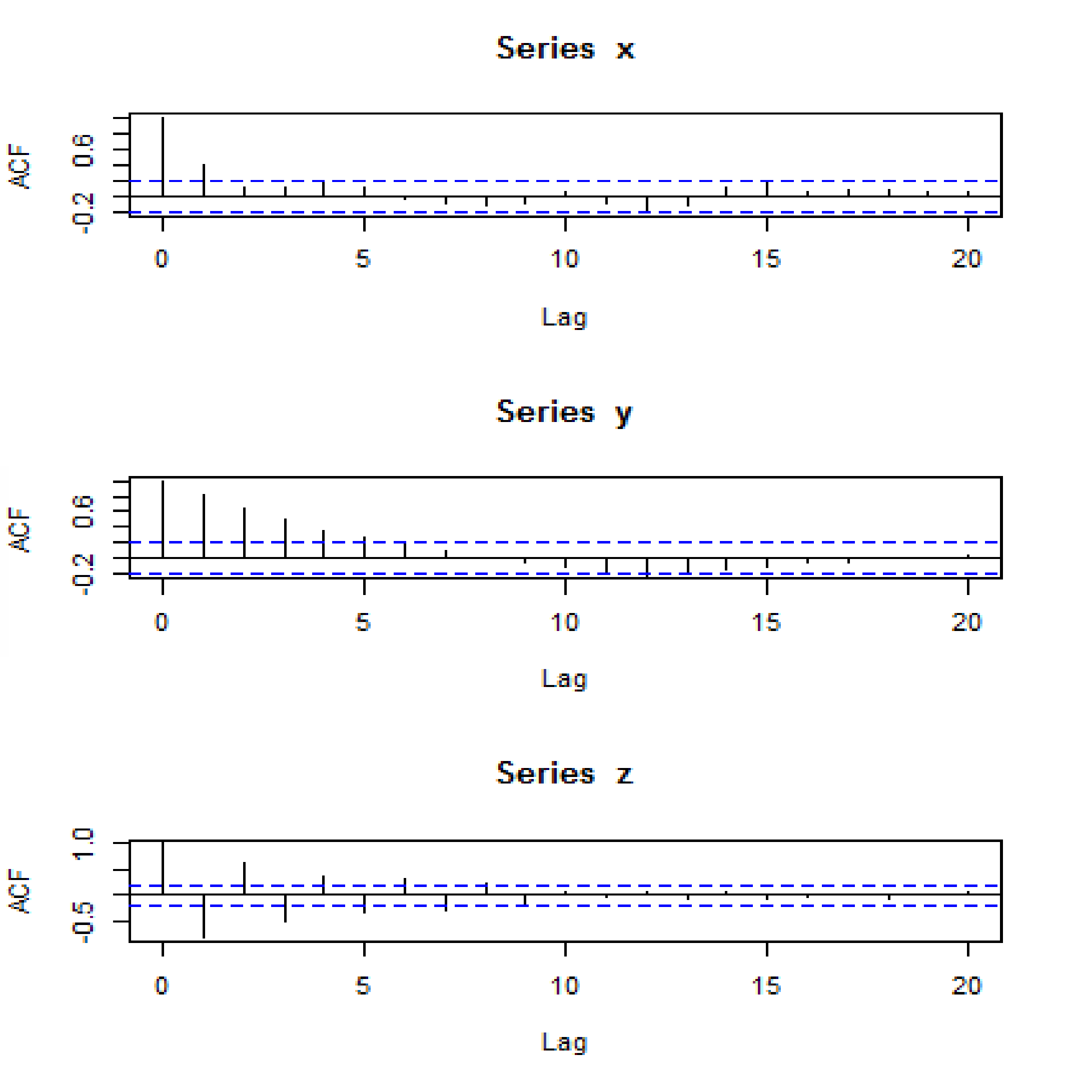
- AR process can be simulated by specifying list(ar=ϕ) in the option model of the function arima.sim()

```
x <- arima.sim(model = list(ar=0.4), n = 100)
y <- arima.sim(model = list(ar=0.8), n = 100)
z <- arima.sim(model = list(ar=-0.9), n = 100)
plot.ts(cbind(x, y, z), main="x(phi=0.4),y(phi=0.8),z(phi=-0.9)")
```



ACF

```
par(mfcol=c(3,1))
acf(x)
acf(y)
acf(z)
```



Fitting AR Models

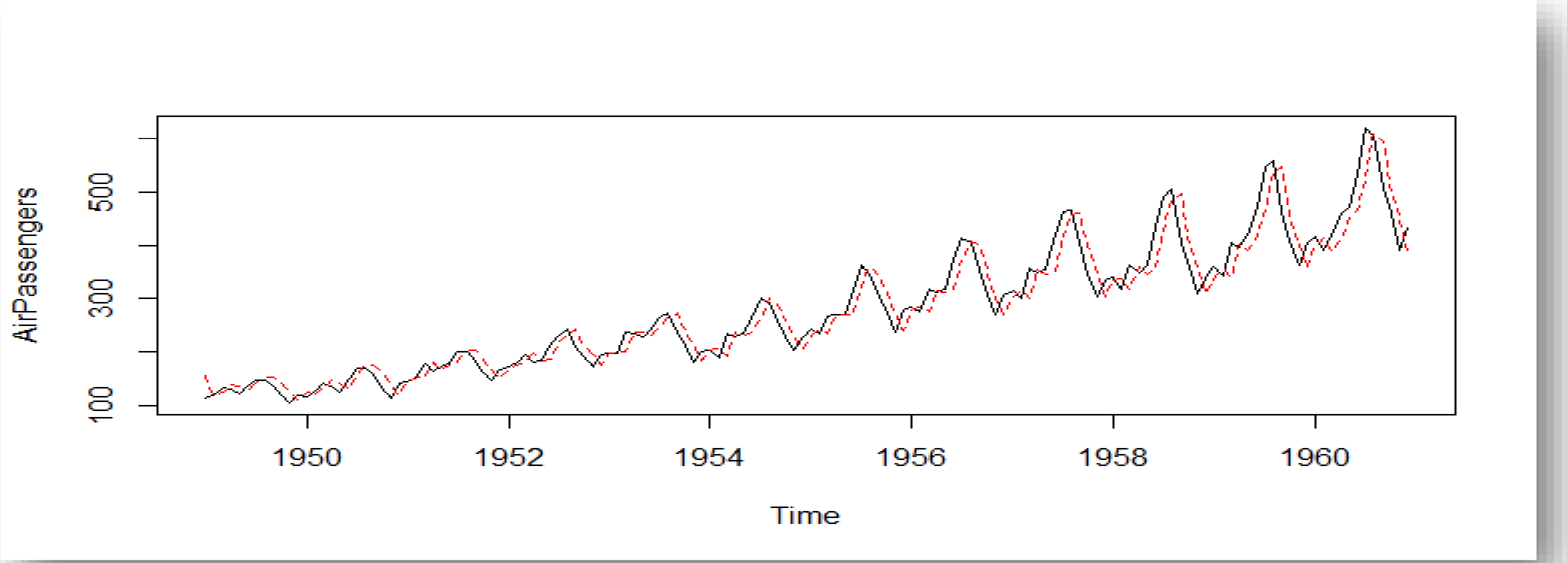
- AR models can be fitted with the following function call:

```
arima(x, order = c(1,0,0))
```

- Here 1 in c(1,0,0) is the function call stands for order of AR
- For calculating residuals we can use the function residuals()

Fitting AR Model

```
> AR <- arima(AirPassengers, order = c(1,0,0))
> print(AR)

Call:
arima(x = AirPassengers, order = c(1, 0, 0))

Coefficients:
         ar1  intercept
      0.9646   278.4649
s.e.  0.0214    67.1141

sigma^2 estimated as 1119:  log likelihood = -711.09,  aic = 1428.18
> ts.plot(AirPassengers)
> AR_fitted <- AirPassengers - residuals(AR)
> points(AR_fitted, type = "l", col = 2, lty = 2)
```



Forecasting

```
> predict(AR)
$pred
          Jan
1961 426.5698

$se
          Jan
1961 33.44577

> predict(AR,n.ahead = 6)
$pred
          Jan      Feb      Mar      Apr      May      Jun
1961 426.5698 421.3316 416.2787 411.4045 406.7027 402.1672

$se
          Jan      Feb      Mar      Apr      May      Jun
1961 33.44577 46.47055 55.92922 63.47710 69.77093 75.15550
```

Simple Moving Average Model

MA Process

Simple Moving Average Model

- Simple MA model:
$$Today's\ \text{Value} = \text{Mean} + \text{Noise} + \text{Slope} * (\text{Yesterday's Noise})$$
- In mathematical notations,
$$y_t = \mu + \epsilon_t + \theta\epsilon_{t-1}$$

Where

μ: Mean of the series

θ: Slope

ϵ_t: Error or Noise at time t which has mean 0 and some variance σ_ϵ^2

- At $\theta = 0$, the model will be a white noise with mean μ and variance σ_ϵ^2

Simple Moving Average Model

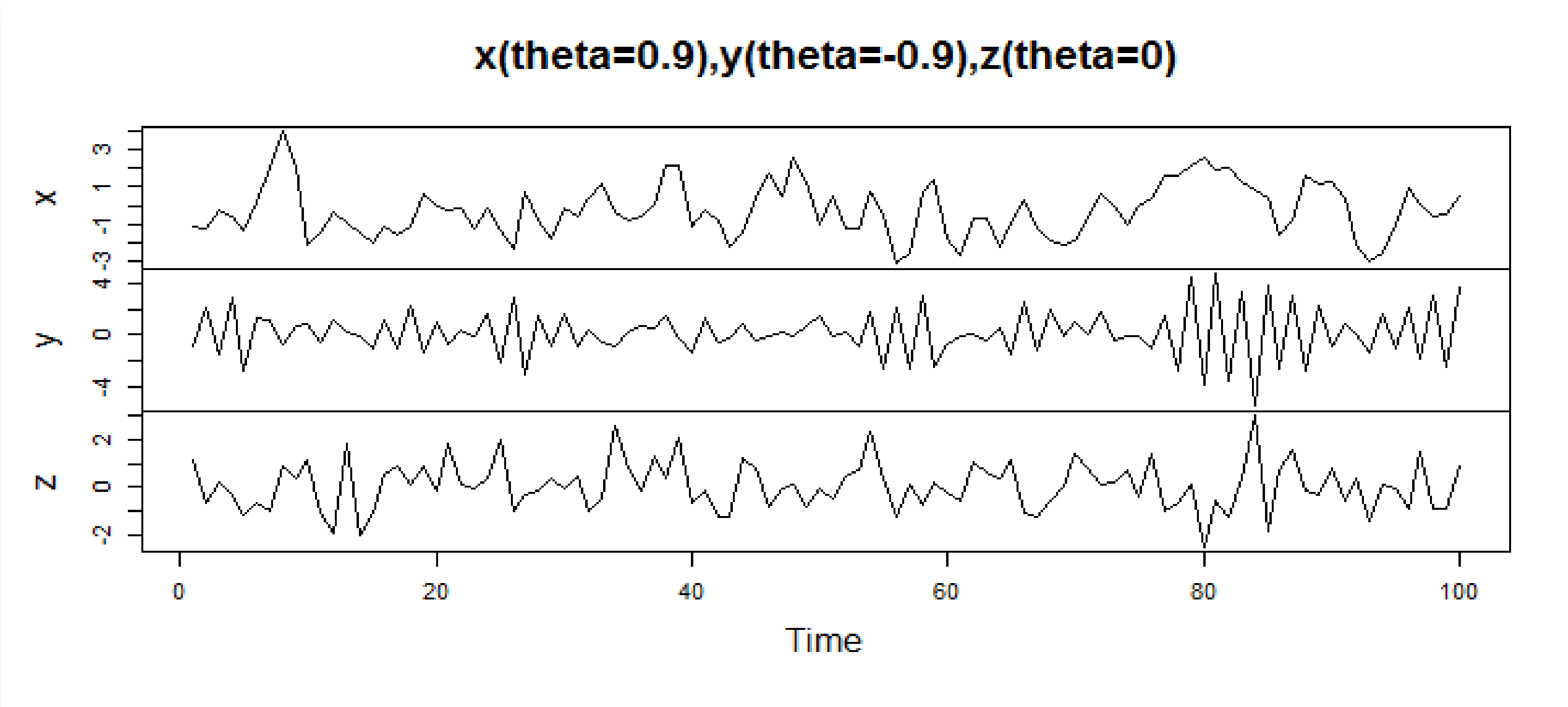
$$y_t = \mu + \epsilon_t + \theta\epsilon_{t-1}$$

- If θ is non-zero then y_t depends on both ϵ_t and ϵ_{t-1} and the process is auto correlated
- Larger values of θ imply greater autocorrelation
- Negative values of θ imply oscillatory time series

Simulating MA process

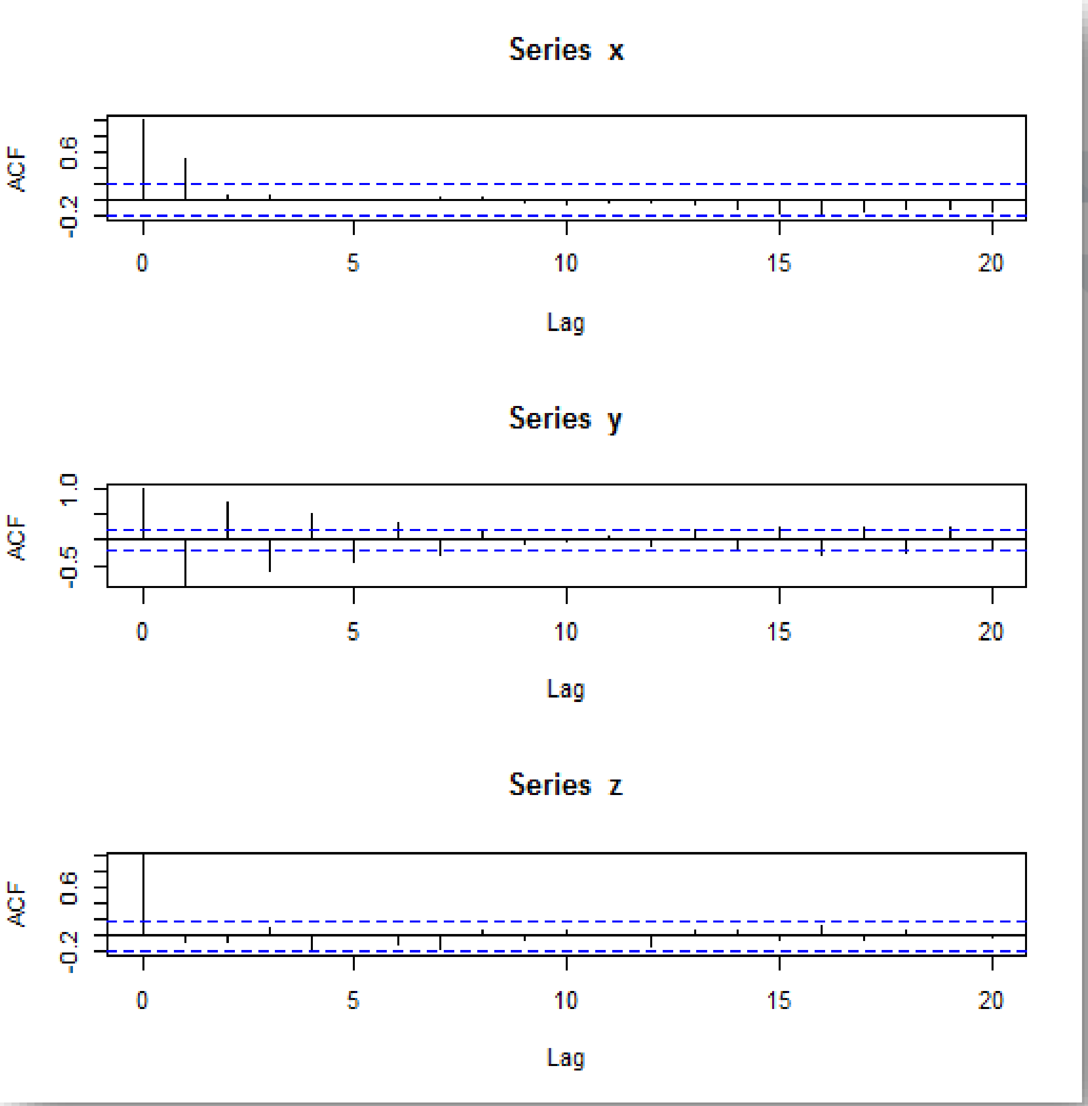
- MA process can be simulated by specifying list(MA=θ) in the option model of the function arima.sim()

```
x <- arima.sim(model = list(ma=0.9), n = 100)
y <- arima.sim(model = list(ar=-0.9), n = 100)
z <- arima.sim(model = list(ar=0.01), n = 100)
```



ACF

```
par(mfcol=c(3,1))
acf(x)
acf(y)
acf(z)
```



Fitting MA Models

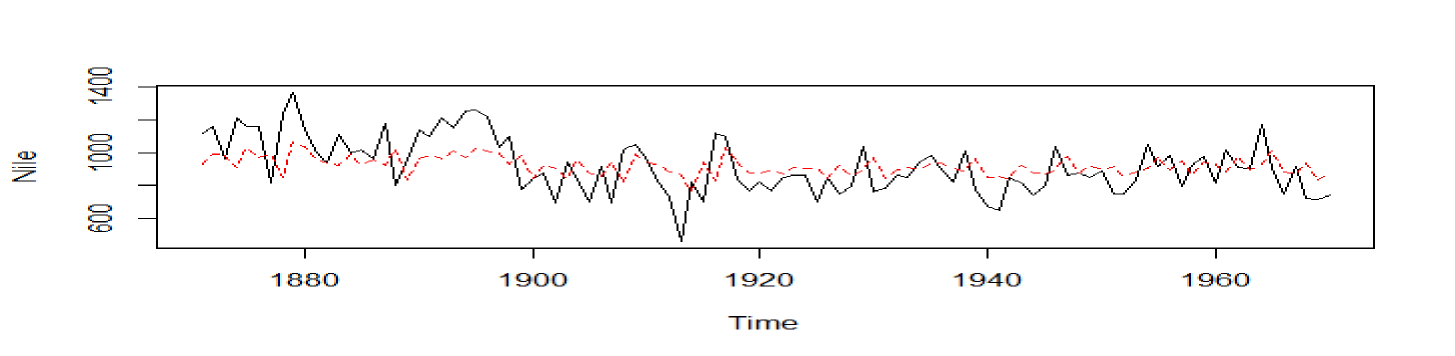
- MA models can be fitted with the following function call:

```
arima(x, order = c(0,0,1))
```

- Here 1 in c(0,0,1) is the function call stands for order of MA
- For calculating residuals we can use the function residuals()

Fitting a MA Model

```
> ts.plot(Nile)
> MA_fit <- Nile - residuals(MA)
> points(MA_fit, type = "l", col = 2, lty = 2)
>
> MA <- arima(Nile, order = c(0,0,1))
> print(MA)

Call:
arima(x = Nile, order = c(0, 0, 1))

Coefficients:
         ma1  intercept
      0.3783   919.2433
s.e.  0.0791    20.9685

sigma^2 estimated as 23272:  log likelihood = -644.72,  aic = 1295.44
> ts.plot(Nile)
> MA_fit <- Nile - residuals(MA)
> points(MA_fit, type = "l", col = 2, lty = 2)
```



Forecasting

```
> predict(MA)
$pred
Time Series:
Start = 1971
End = 1971
Frequency = 1
[1] 868.8747

$se
Time Series:
Start = 1971
End = 1971
Frequency = 1
[1] 152.5508

> predict(MA,n.ahead = 6)
$pred
Time Series:
Start = 1971
End = 1976
Frequency = 1
[1] 868.8747 919.2433 919.2433 919.2433 919.2433 919.2433

$se
Time Series:
Start = 1971
End = 1976
Frequency = 1
[1] 152.5508 163.1006 163.1006 163.1006 163.1006 163.1006
```