Analysis of Time Series Data Using R

Preliminaries

Always remember to load the packages that we need to use for analyzing time series in R. I've listed them below:

```
library(tidyverse)
library(tidyquant)
library(gridExtra)
library(tibbletime)
library(forecast)
library(itsmr)
library(tsibble)
library(fpp2)
knitr::opts_chunk$set(comment=NA,tidy=FALSE)

#library(future) Not needed yet
#library(doFuture) Not needed yet
#library(rbenchmark) Not needed yet
```

Developing intuition on simulated data

- Can use the `arima.sim` function to generate data according to various ARMA models
- `arima.sim` requires two arguments (there are others, but defaults are OK for our purposes): a model specification and a number of points in the series to generate
- Specify models by using a named `list`
- Ex.

```
true_ar_coef=c(0.6,0.2)
true_ma_coef=c(0.4)
my_ARMA_2_1_model = list(ar=true_ar_coef, ma=true_ma_coef)
my_ARMA_2_1_model
```

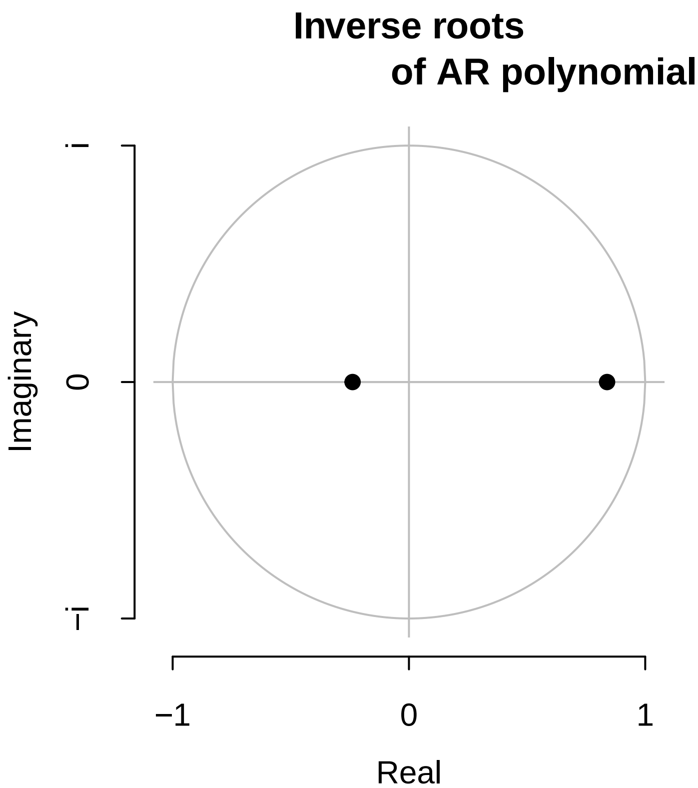
```
$ar
[1] 0.6 0.2

$ma
[1] 0.4
```

- The above code specifies an ARMA(2,1) model where $\phi = (\phi_1, \phi_2) = (0.6, 0.2)$ and $\theta = (\theta_1) = 0.4$. If you want only an AR or MA model, you can either leave out that component of the vector or set it to NULL.
- We can find (and plot the inverse of) the roots of the AR and MA polynomials using the code below. Note that we plot the *inverse* of the roots, as in most cases we will have causal and/or invertible series, so this keeps the plots nice as the roots will be inside of the circle instead of potentially far outside of it.

```r
ar_roots<-polyroot(c(1,-my_ARMA_2_1_model$ar))
ar_roots
```

```
[1]  1.192582-0i -4.192582+0i
```

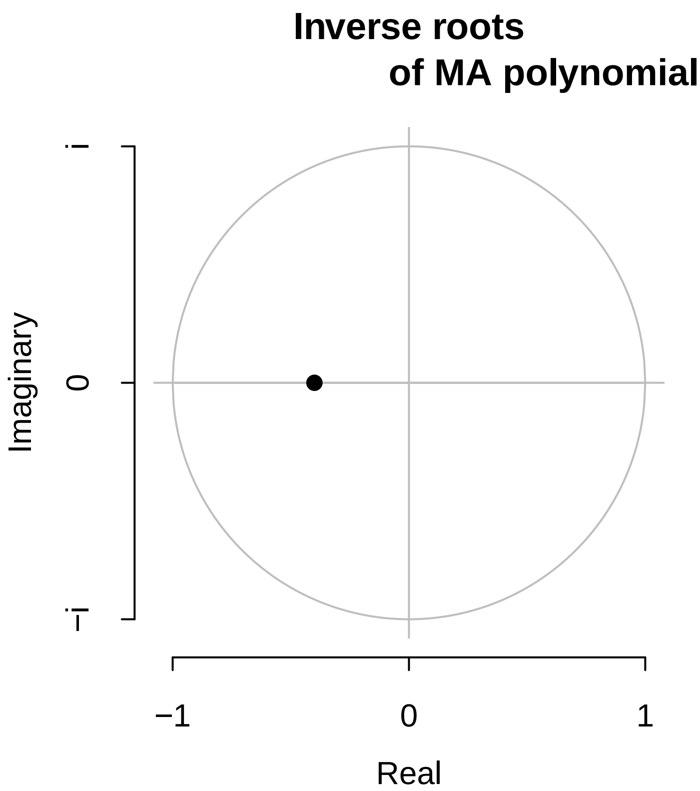
```r
forecast:::plot.armaroots(structure(list(roots=ar_roots),
                          class = "armaroots"),xlab="Real", ylab="Imaginary",main="Inverse roots
                          of AR polynomial")
```




```r
ma_roots<-polyroot(c(1,my_ARMA_2_1_model$ma))
ma_roots
```

```
[1] -2.5+0i
```

```r
forecast:::plot.armaroots(structure(list(roots=ma_roots),
                          class = "armaroots"),xlab="Real", ylab="Imaginary",main="Inverse roots
                          of MA polynomial")
```



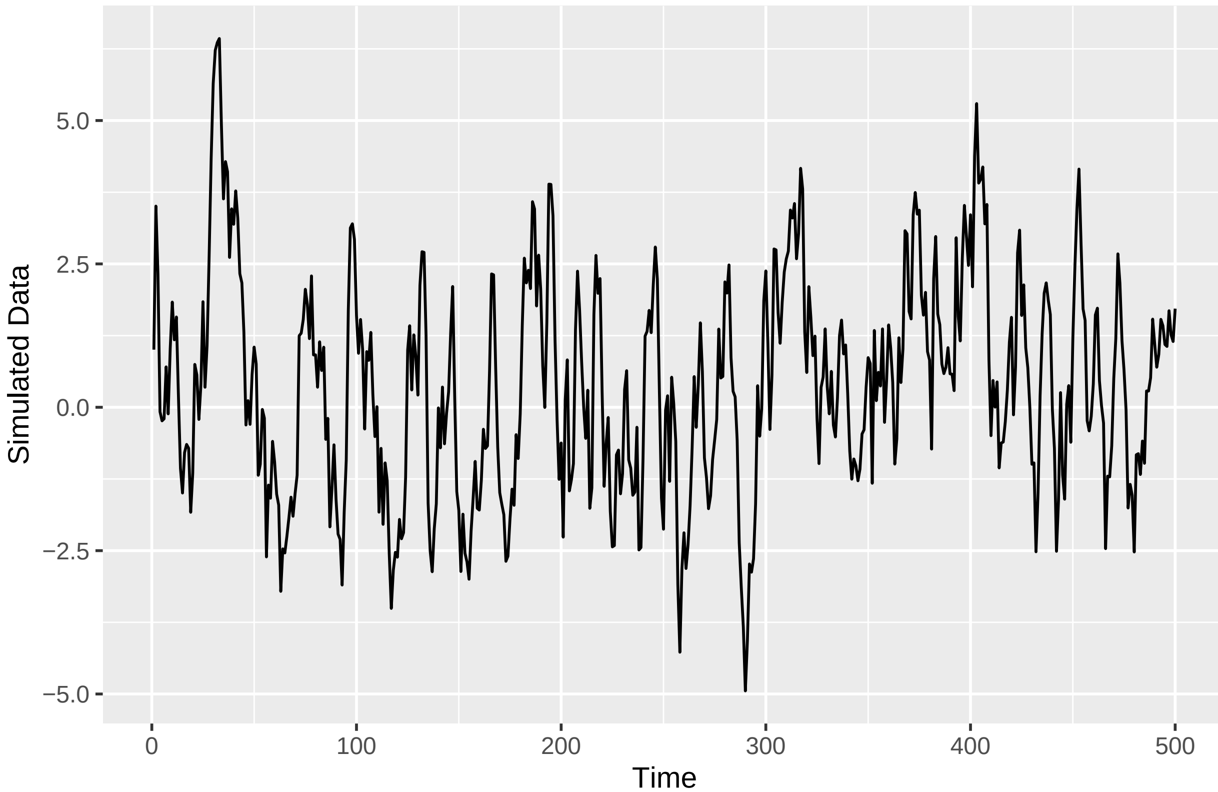
Simulating the data

- With these cofficients, we can now generate an ARMA(2,1) model using the `arima.sim` function:

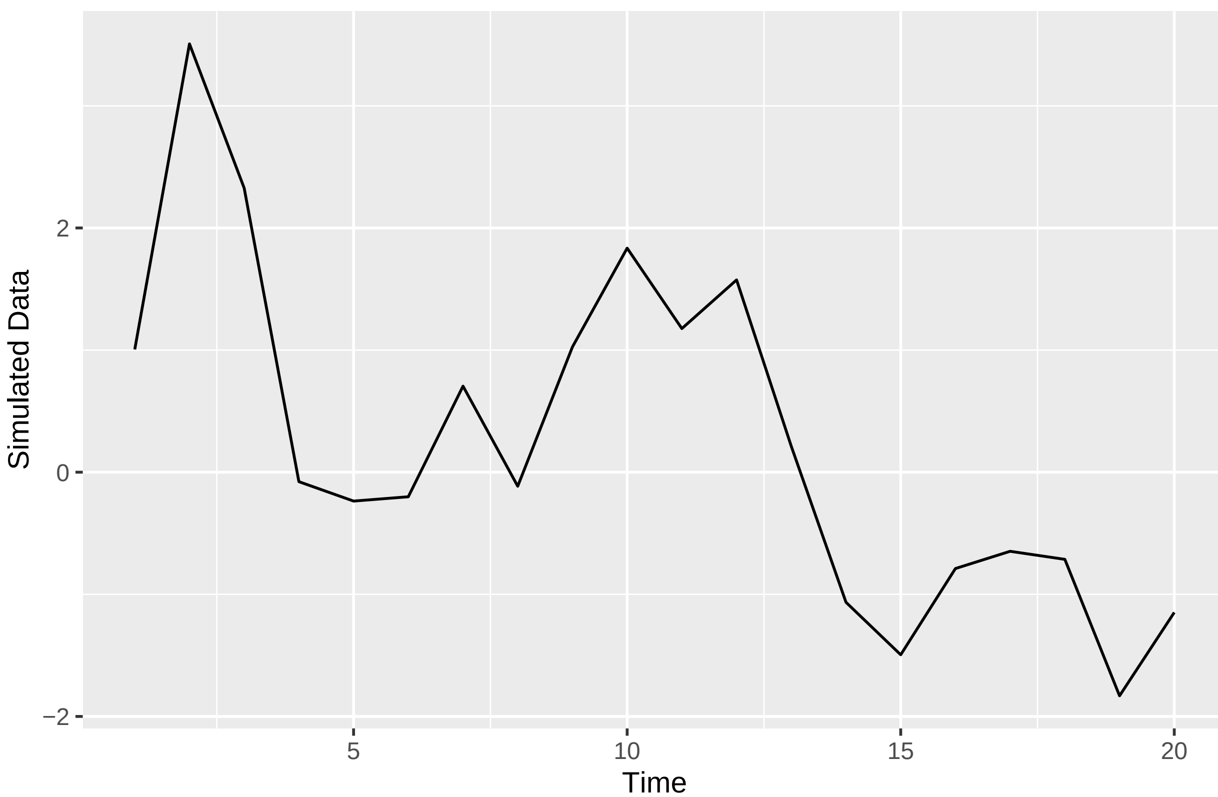
```
my_arma_2_1_data = arima.sim(my_ARMA_2_1_model,n=500)
glimpse(my_arma_2_1_data)
```

```
 Time-Series [1:500] from 1 to 500: 1.0052 3.5077 2.3267 -0.0776 -0.2365 ...
```

```
autoplot(my_arma_2_1_data) + ylab("Simulated Data")
```




```
autoplot(window(my_arma_2_1_data,end=20)) + ylab("Simulated Data")
```

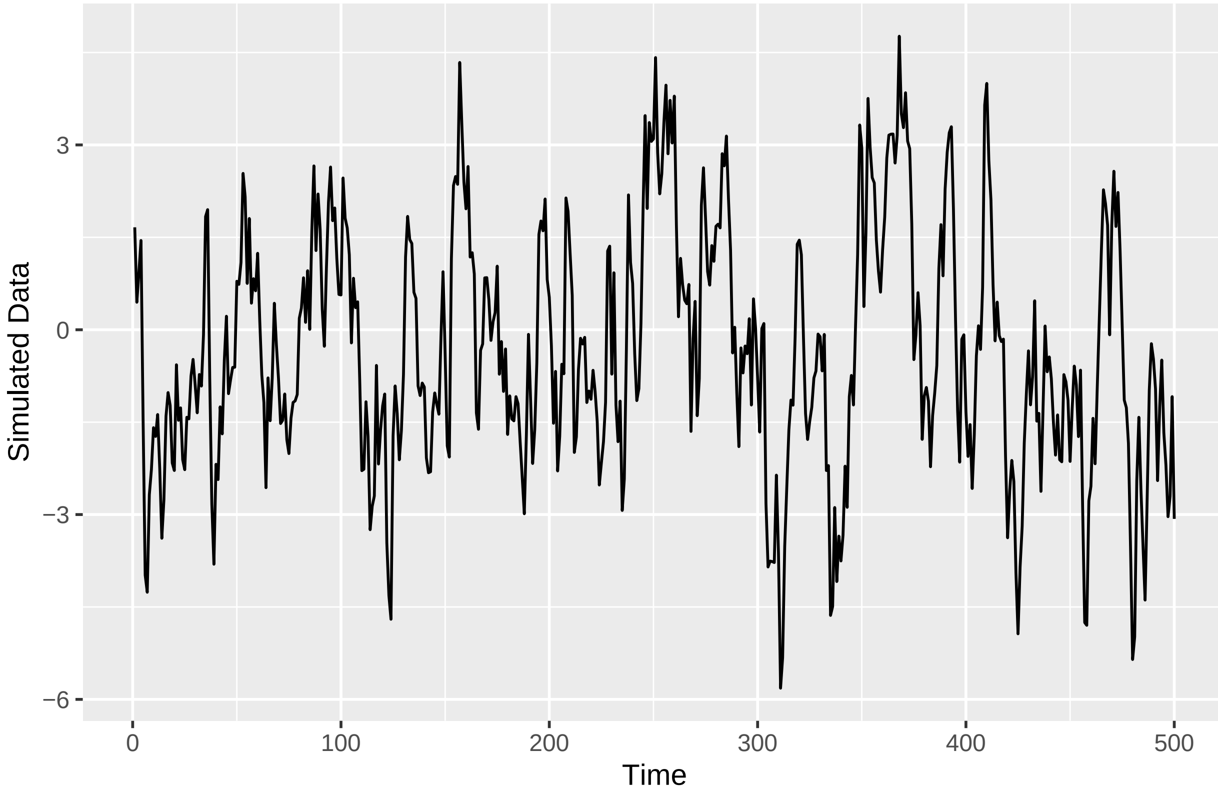


- Note if we want to be able to replicate our simulations (i.e. obtain the same dataset each time we run our code), we need to set a random seed (different seeds or running two sims without resetting the seed leads to different data):

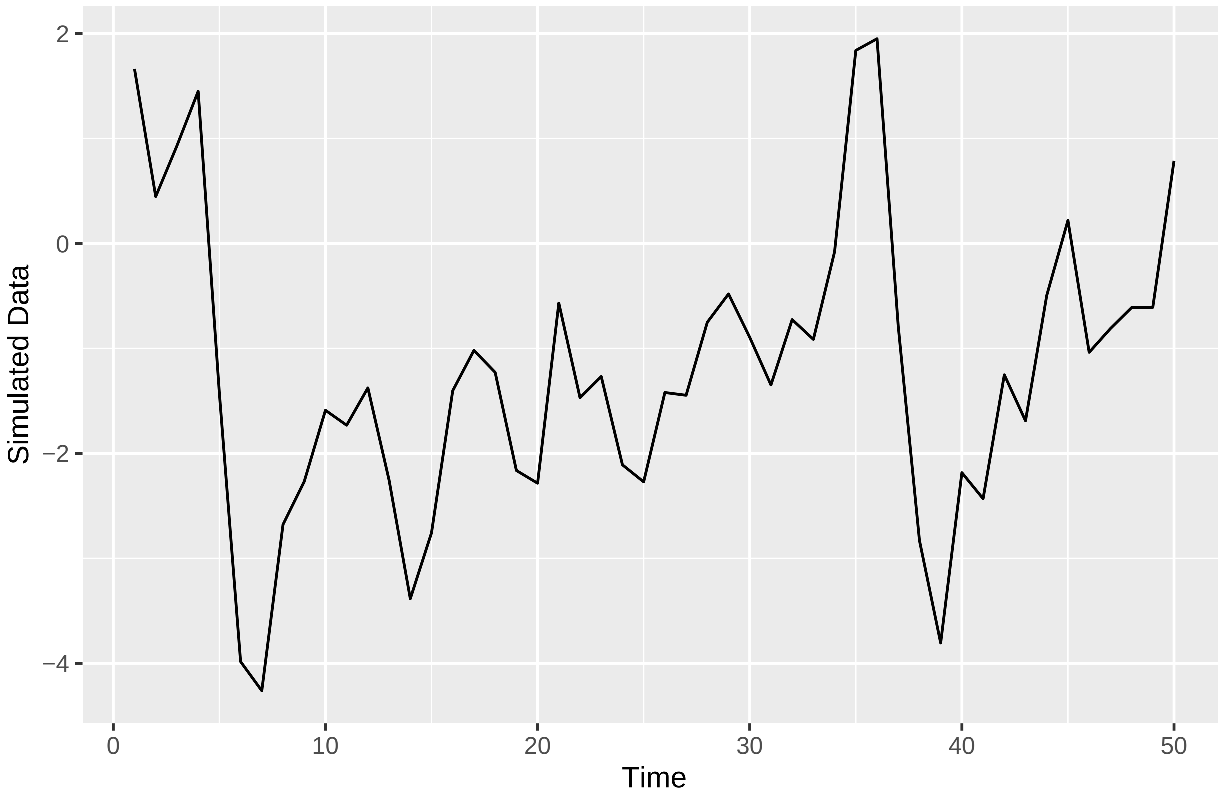
```
set.seed(19750606)
my_arma_2_1_data = arima.sim(my_ARMA_2_1_model,n=500)
glimpse(my_arma_2_1_data)
```

```
 Time-Series [1:500] from 1 to 500: 1.663 0.446 0.93 1.449 -1.426 ...
```

```
autoplot(my_arma_2_1_data) + ylab("Simulated Data")
```



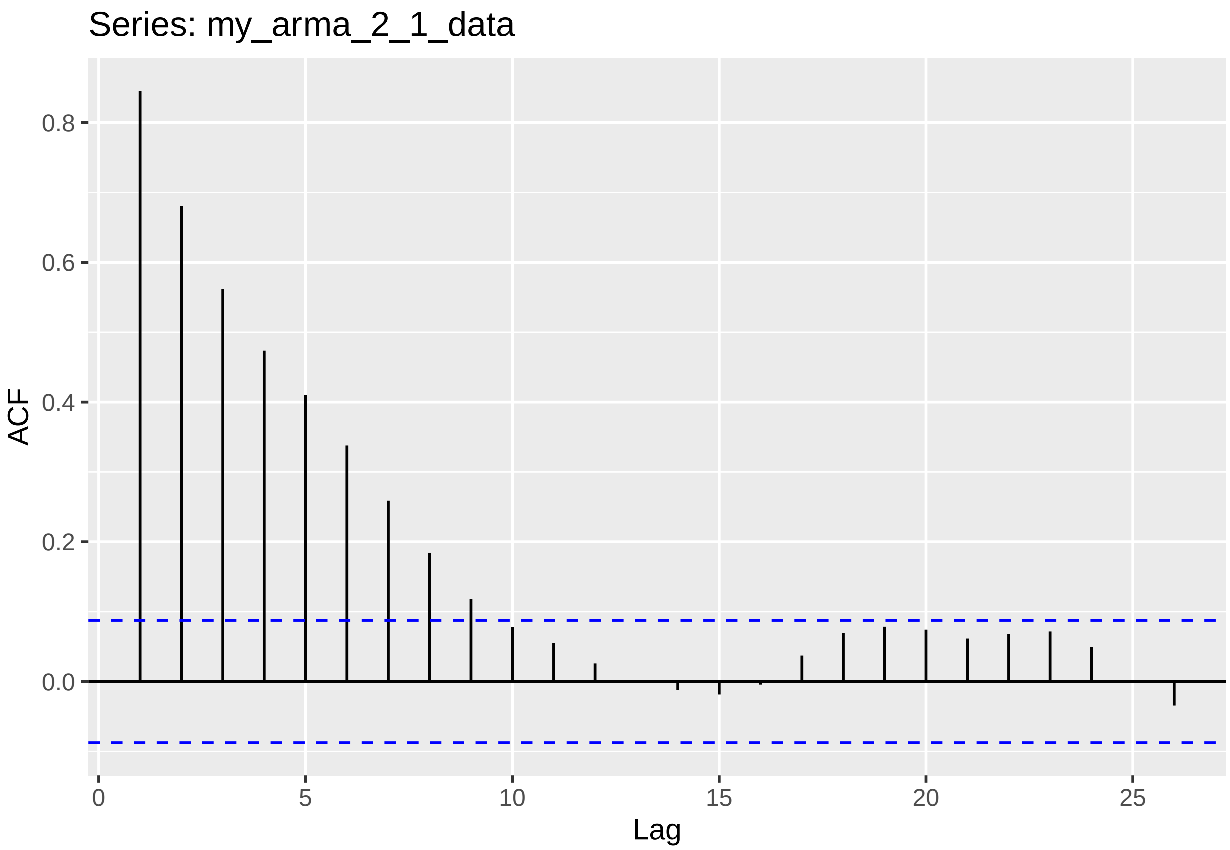

```
autoplot(window(my_arma_2_1_data,end=50)) + ylab("Simulated Data")
```



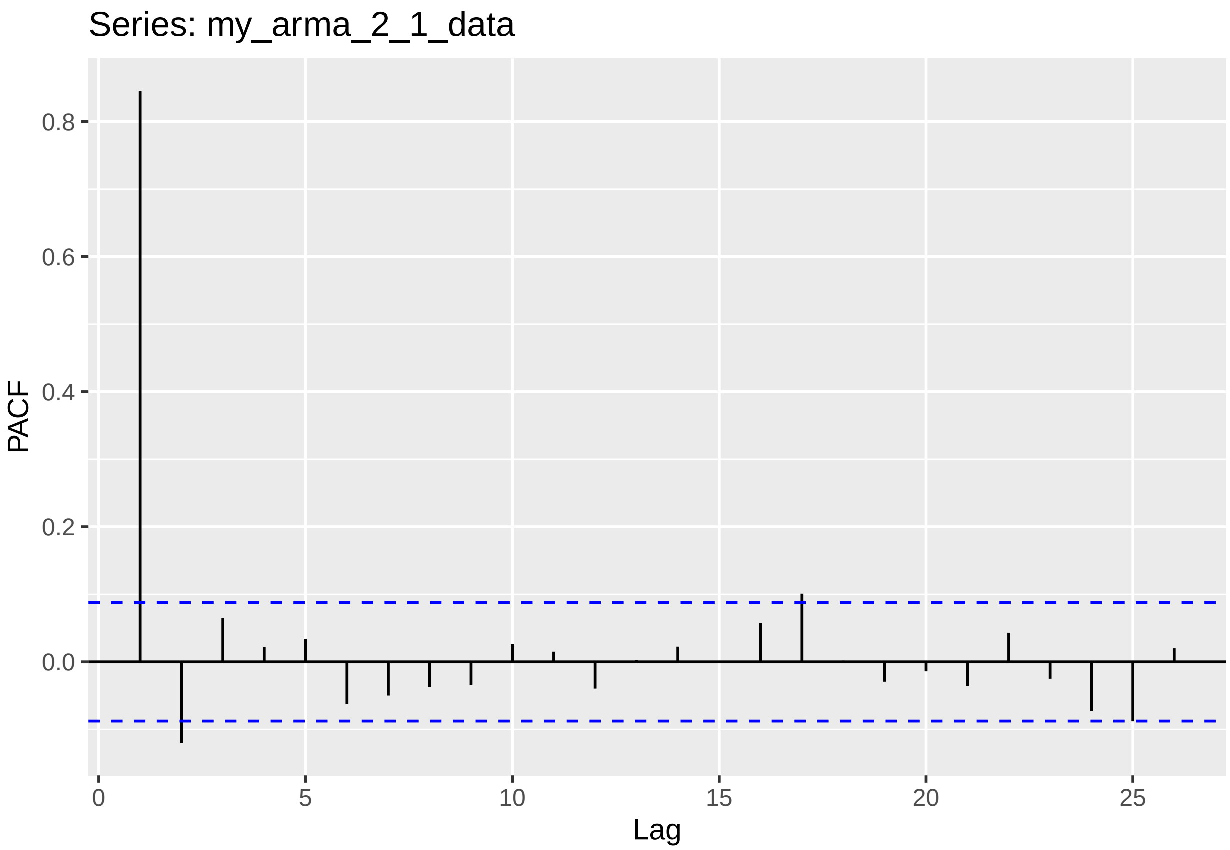
Sample autocorrelation

- Recall we can plot the autocorrelation for various lags using the `ggAcf` function:

```
ggAcf(my_arma_2_1_data)
```




```
ggPacf(my_arma_2_1_data)
```



- Can compute theoretical ACF and PACF values using the `ARMAacf` function and them to the plot:

```
ARMAacf(ar=true_ar_coef,ma=true_ma_coef,lag.max=10)
```

```
        0         1         2         3         4         5         6         7
1.0000000 0.8693182 0.7215909 0.6068182 0.5084091 0.4264091 0.3575273 0.2997982
        8         9        10
0.2513844 0.2107903 0.1767510
```

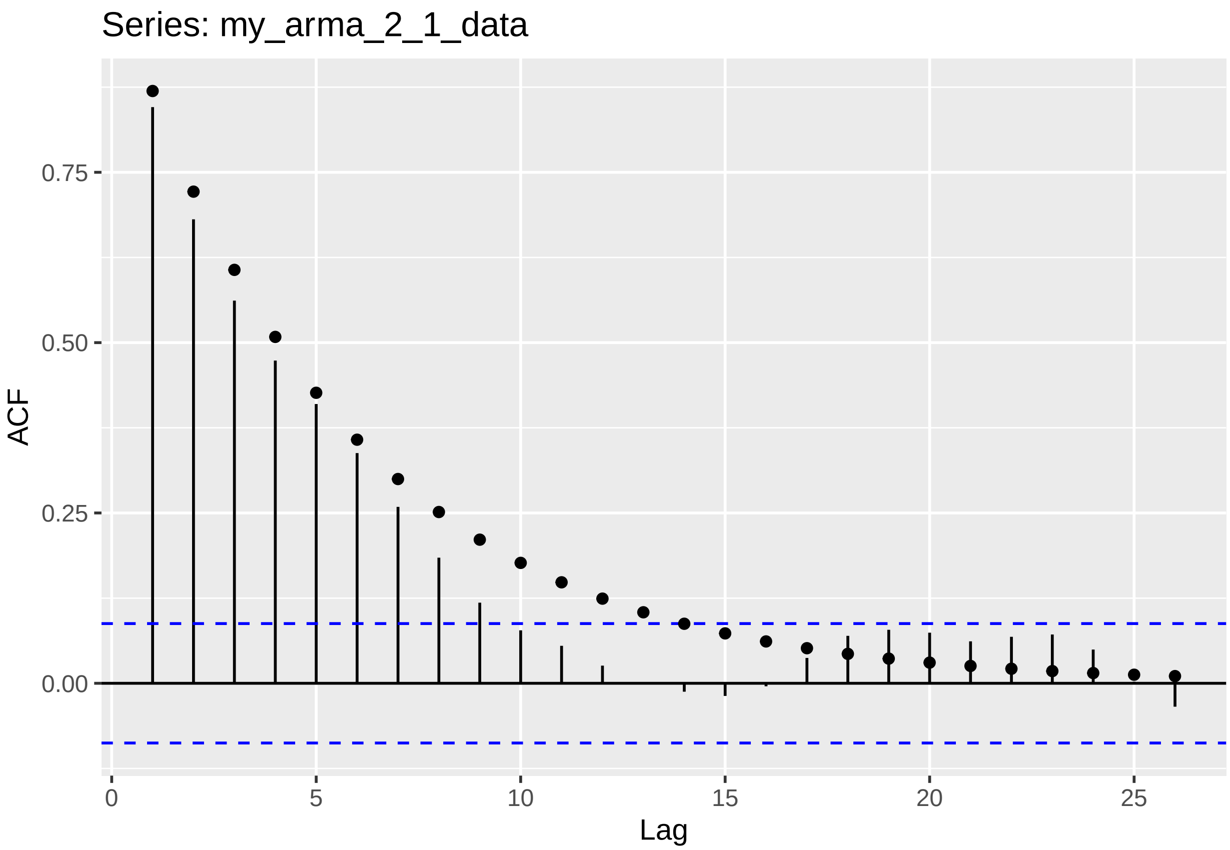
```
ARMAacf(ar=true_ar_coef,ma=true_ma_coef,lag.max=25) %>% head(.)
```

```
        0         1         2         3         4         5
1.0000000 0.8693182 0.7215909 0.6068182 0.5084091 0.4264091
```

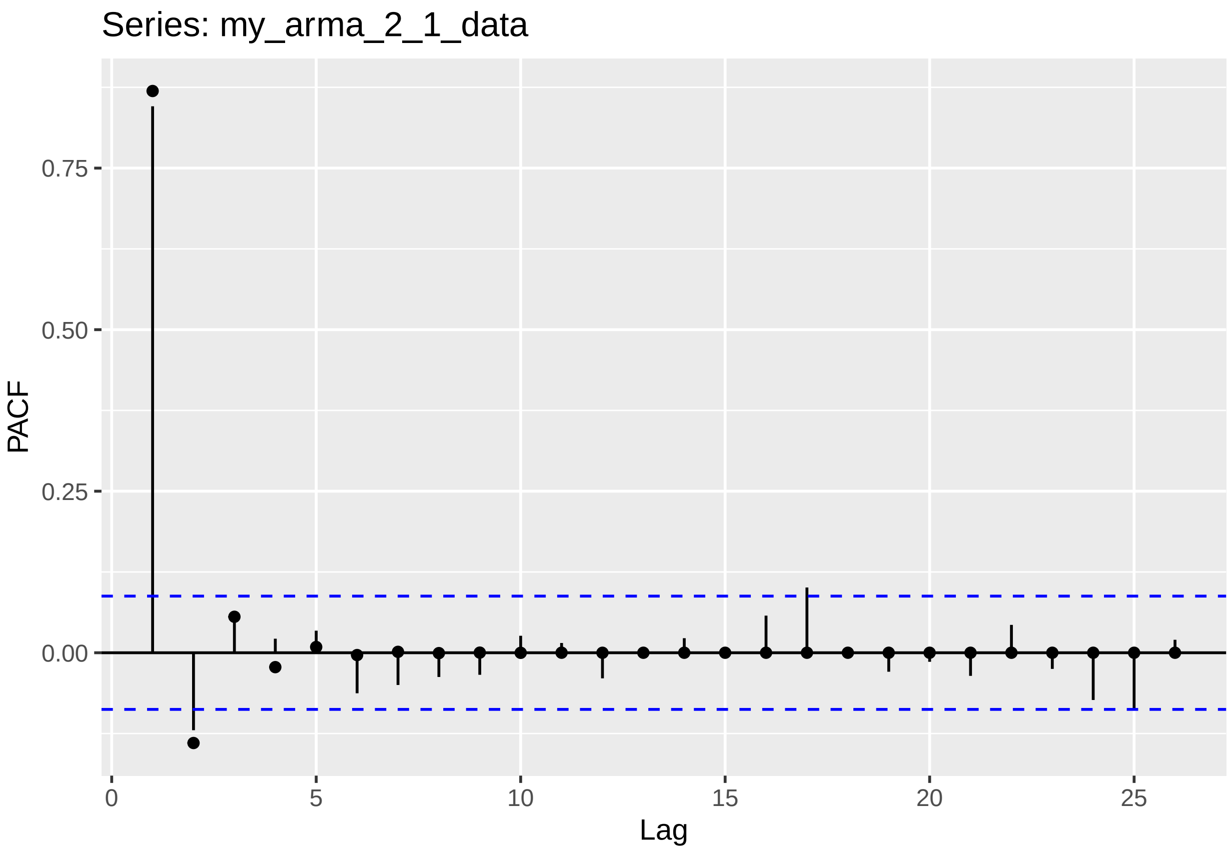
```
ARMAacf(ar=true_ar_coef,ma=true_ma_coef,lag.max=25,pacf=TRUE) %>% head(.)
```

```
[1]  0.869318182 -0.139685476  0.055668203 -0.022254154  0.008900822
[6] -0.003560275
```

```
ggAcf(my_arma_2_1_data) +
  geom_point(aes(y=ARMAacf(ar=true_ar_coef,ma=true_ma_coef,lag.max=26)[-1]))
```




```
ggPacf(my_arma_2_1_data) +
  geom_point(aes(y=ARMAacf(ar=true_ar_coef,ma=true_ma_coef,
                          lag.max=26,pacf=TRUE)))
```



Functions for estimating coefficients

AR estimation

- First let's generate some true AR(4) data:

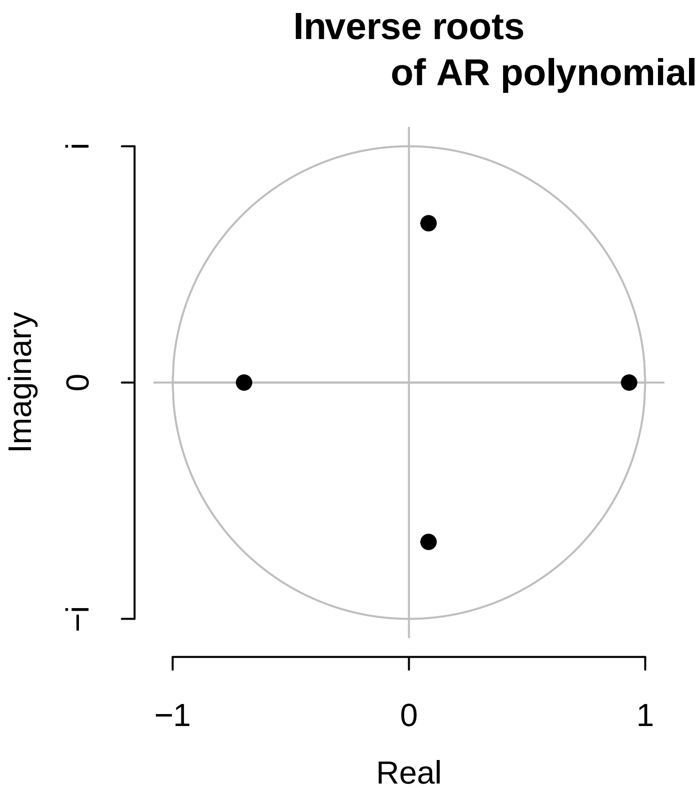
```
true_ar_coef_2=c(0.4,0.15,0,0.3)
my_AR4_model = list(ar=true_ar_coef_2,ma=NULL)
```

- Check the roots to make sure it is stationary

```
ar_roots<-polyroot(c(1,-my_AR4_model$ar))
ar_roots
```

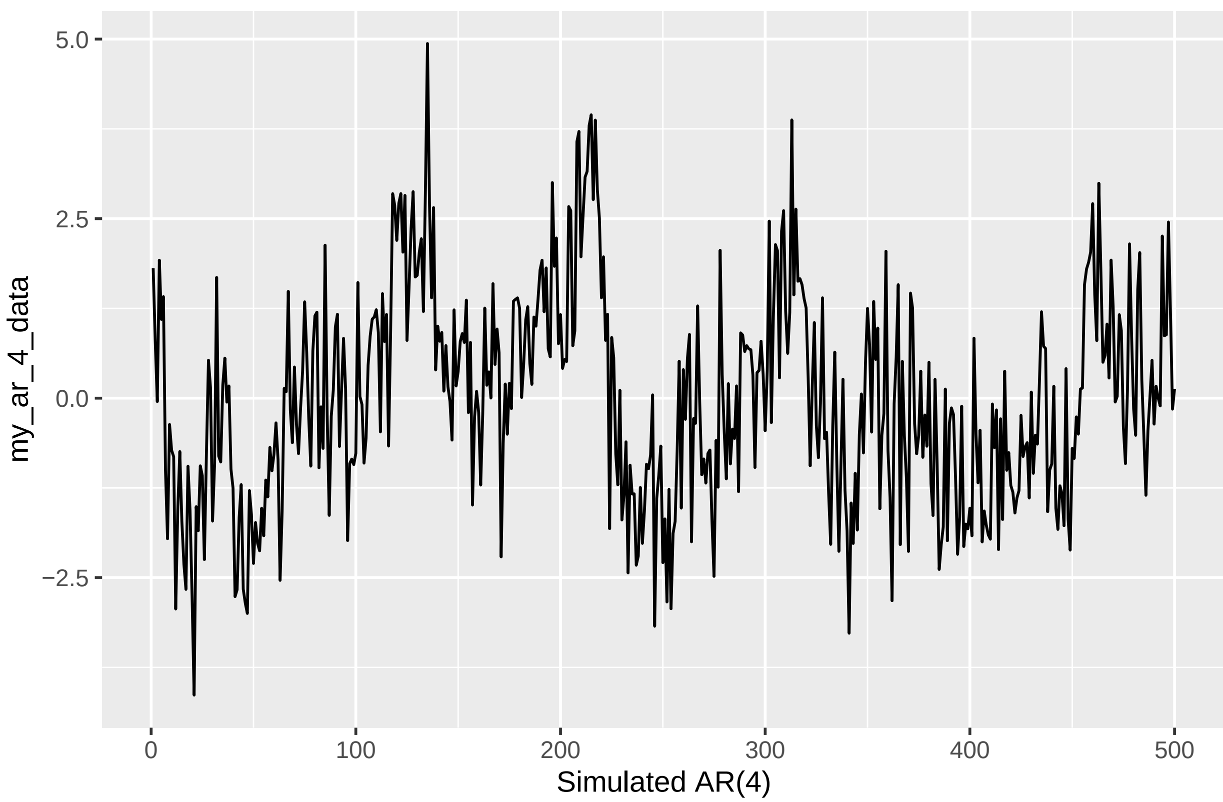
```
[1]  1.073196-0.000000i -1.433037+0.000000i  0.179920-1.461179i
[4]  0.179920+1.461179i
```

```
forecast:::plot.armaroots(structure(list(roots=ar_roots),
                          class = "armaroots"),xlab="Real", ylab="Imaginary",main="Inverse roots
                          of AR polynomial")
```

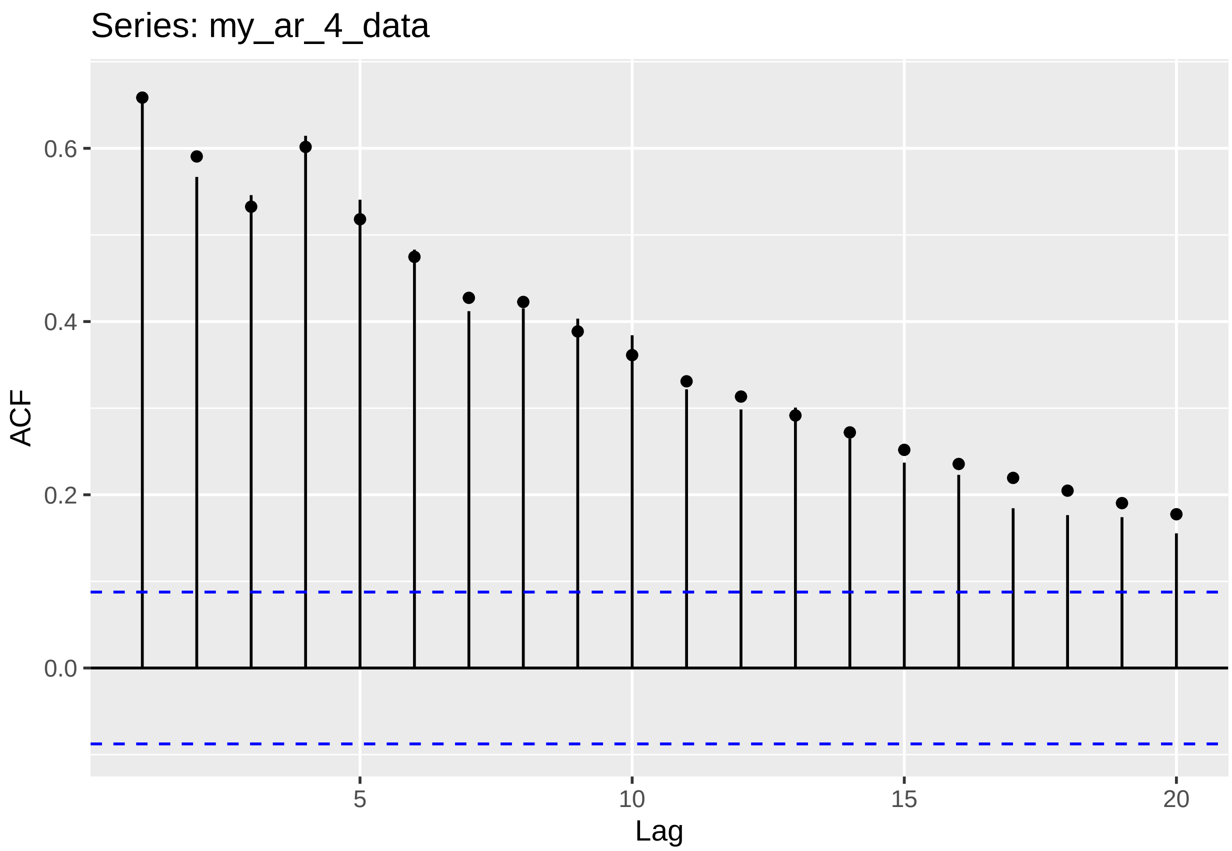


- Now generate some data and compare sample ACF and PACF to truth

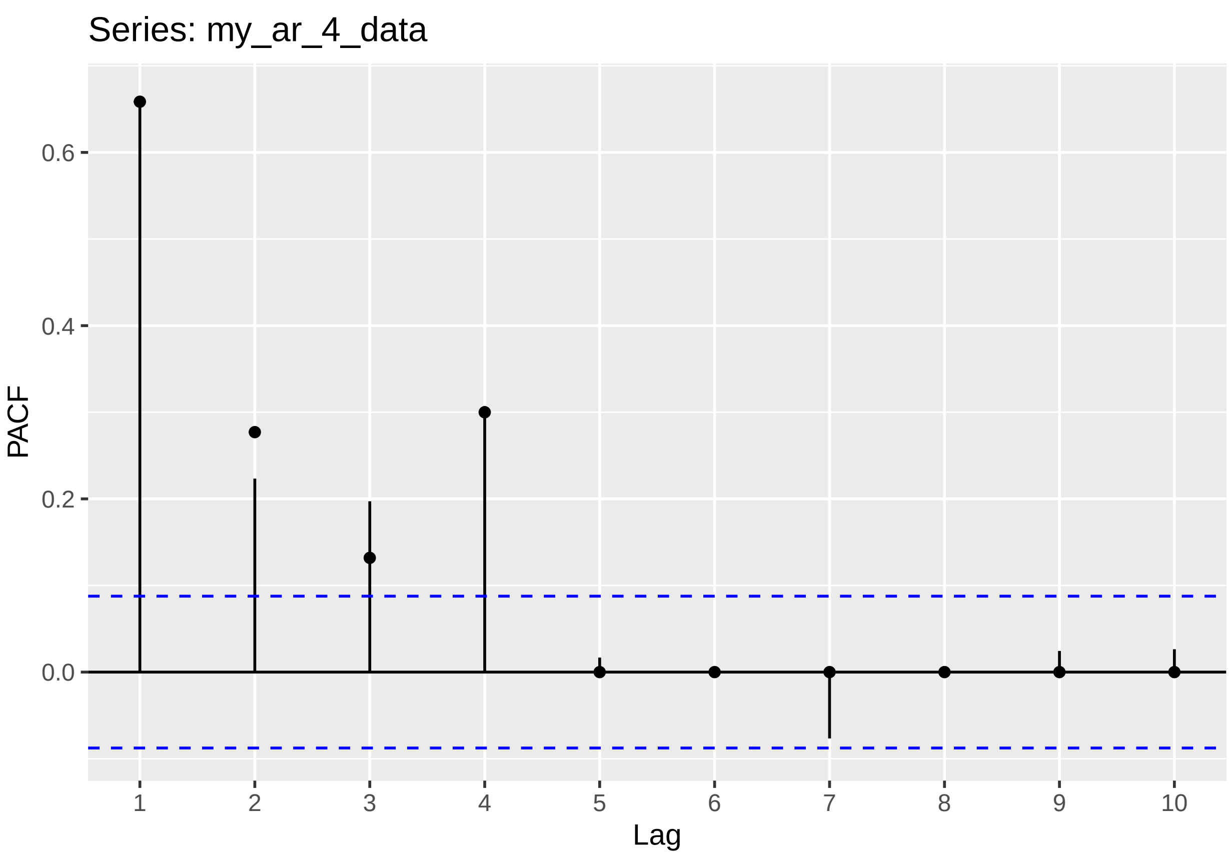
```
my_ar_4_data = arima.sim(my_AR4_model,n=500)
autoplot(my_ar_4_data) + xlab("Simulated AR(4)")
```



```r
ggAcf(my_ar_4_data,lag.max=20) +
  geom_point(aes(y=c(ARMAacf(ar=true_ar_coef_2,ma=0,lag.max=20)[-1])))
```




```r
ggPacf(my_ar_4_data,lag.max=10) +
  geom_point(aes(y=c(ARMAacf(ar=true_ar_coef_2,pacf=TRUE),rep(0,6))))
```



Estimation via Yule-Walker, Burg, and MLE

```
my_ar_mod_4_yw<-ar(my_ar_4_data,method="yw",aic=FALSE,order.max=10)
my_ar_mod_4_burg<-ar(my_ar_4_data,method="burg",aic=FALSE,order.max=10)
my_ar_mod_4_<-ar(my_ar_4_data,method="burg",aic=FALSE,order.max=10)

my_ar_map<-map_df(
  c("yw","burg","mle"),
  function(x){
    myfit=ar(my_ar_4_data,method=x,aic=FALSE,order.max=10)
    tibble(method=x,coef_ind=1:10,truth=c(true_ar_coef_2,rep(0,6)),
           coef=myfit[["ar"]],se=sqrt(diag(myfit[["asy.var.coef"]])))}
  )

my_ar_map %>% filter(method=="yw")
```

```
# A tibble: 10 x 5
   method coef_ind truth    coef     se
   <chr>     <int> <dbl>   <dbl>  <dbl>
 1 yw            1  0.4   0.406  0.0452
 2 yw            2  0.15  0.0847 0.0488
 3 yw            3  0     0.0745 0.0489
 4 yw            4  0.3   0.302  0.0489
 5 yw            5  0     0.0127 0.0508
 6 yw            6  0     0.0275 0.0508
```

```
 7 yw             7  0     -0.0805 0.0489
 8 yw             8  0     -0.0122 0.0489
 9 yw             9  0      0.0137 0.0488
10 yw            10  0      0.0264 0.0452
```

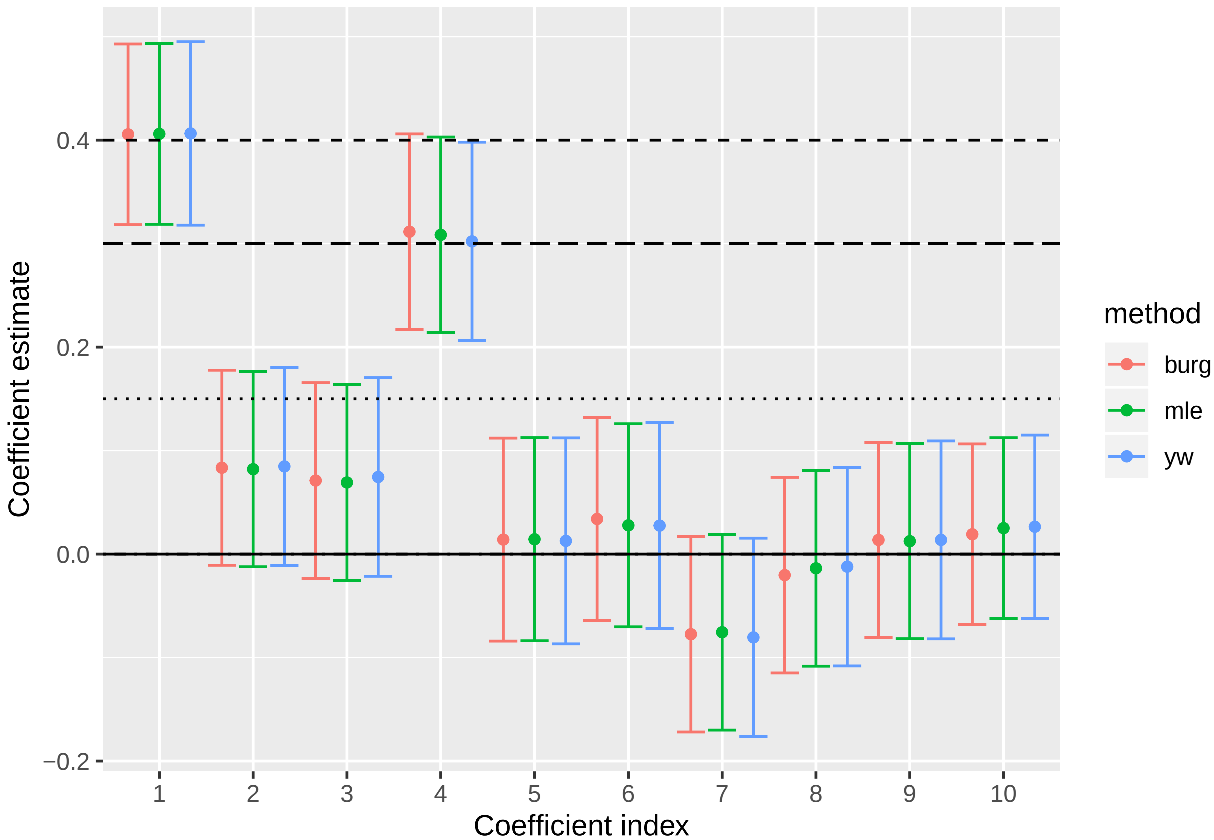
```
my_ar_map %>% filter(method=="burg")
```

```
# A tibble: 10 x 5
   method coef_ind truth    coef     se
   <chr>     <int> <dbl>   <dbl>  <dbl>
 1 burg          1  0.4   0.406  0.0445
 2 burg          2  0.15  0.0834 0.0481
 3 burg          3  0     0.0710 0.0482
 4 burg          4  0.3   0.312  0.0482
 5 burg          5  0     0.0140 0.0500
 6 burg          6  0     0.0339 0.0500
 7 burg          7  0    -0.0774 0.0482
 8 burg          8  0    -0.0203 0.0482
 9 burg          9  0     0.0137 0.0481
10 burg         10  0     0.0191 0.0445
```

```
my_ar_map %>% filter(method=="mle")
```

```
# A tibble: 10 x 5
   method coef_ind truth    coef     se
   <chr>     <int> <dbl>   <dbl>  <dbl>
 1 mle           1  0.4   0.406  0.0446
 2 mle           2  0.15  0.0820 0.0481
 3 mle           3  0     0.0692 0.0482
 4 mle           4  0.3   0.308  0.0482
 5 mle           5  0     0.0143 0.0501
 6 mle           6  0     0.0278 0.0501
 7 mle           7  0    -0.0755 0.0482
 8 mle           8  0    -0.0138 0.0482
 9 mle           9  0     0.0125 0.0481
10 mle          10  0     0.0251 0.0446
```

```
ggplot(my_ar_map,aes(x=factor(coef_ind),y=coef,colour=method,ymin=coef-1.96*se, ymax=coef+1.96*se)) +
  geom_errorbar(position=position_dodge(width=1)) + geom_point(position=position_dodge(width=1)) +
  xlab("Coefficient index") + ylab("Coefficient estimate") +
  geom_hline(yintercept=c(0,true_ar_coef_2),linetype=1:5)
```

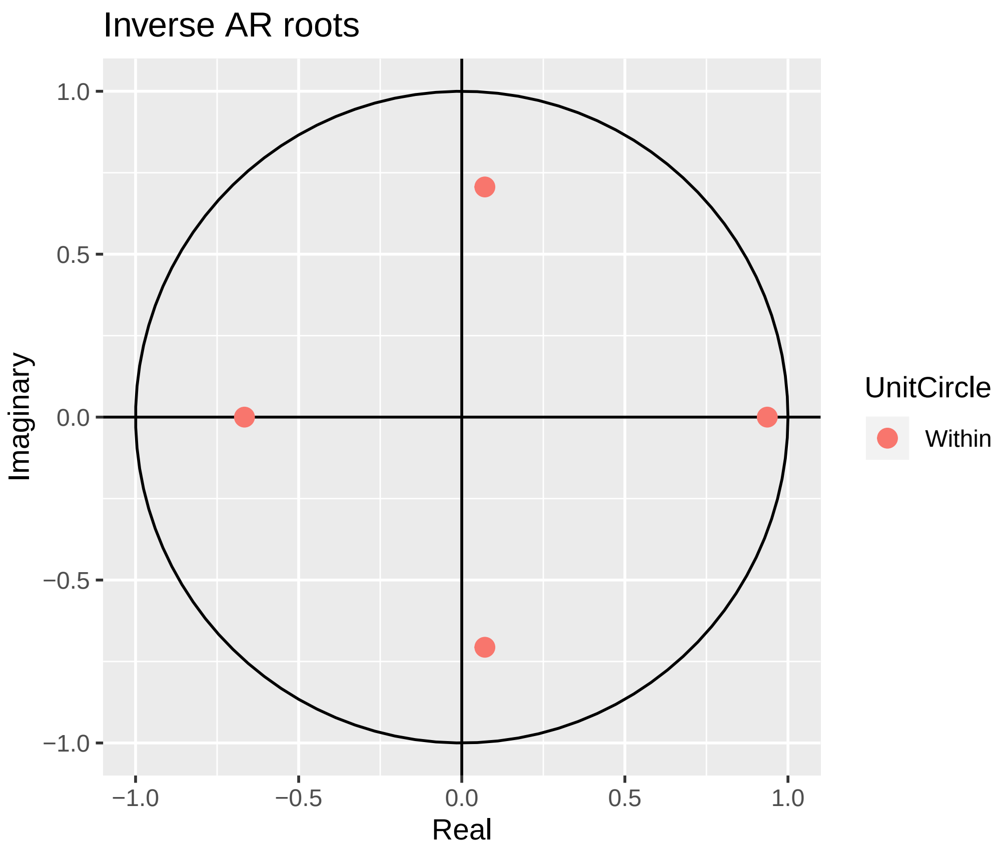


Alternate method for estimating AR models with diagnostics

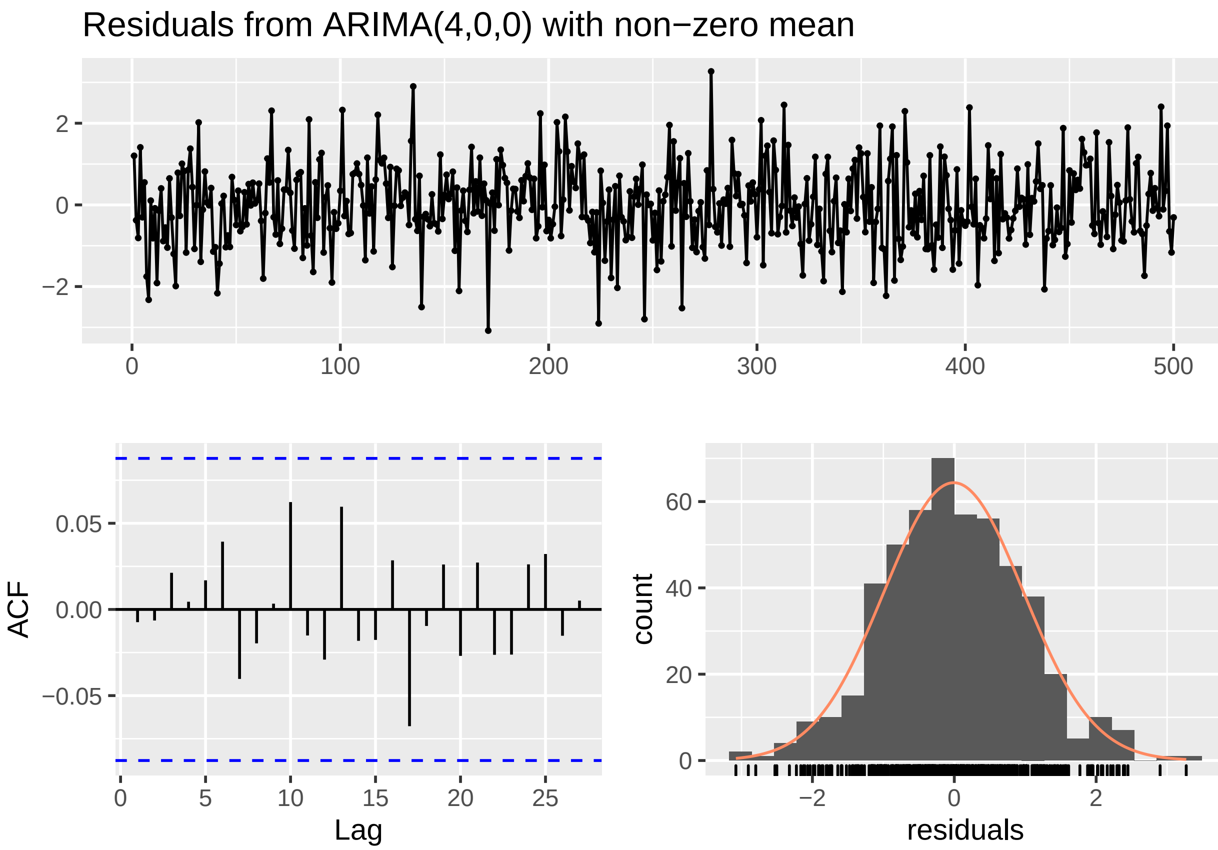
```
arima_mod_ar4<-Arima(my_ar_4_data,order=c(4,0,0))
summary(arima_mod_ar4)
```

```
Series: my_ar_4_data
ARIMA(4,0,0) with non-zero mean

Coefficients:
         ar1     ar2     ar3     ar4    mean
      0.4120  0.0817  0.0479  0.3144  0.0370
s.e.  0.0424  0.0462  0.0463  0.0426  0.2976

sigma^2 estimated as 0.9757:  log likelihood=-701.42
AIC=1414.84   AICc=1415.01   BIC=1440.13

Training set error measures:
                       ME      RMSE       MAE       MPE     MAPE      MASE
Training set -0.008055167 0.9828311 0.7737122 -19.79253 205.9686 0.8244087
                     ACF1
Training set -0.007352168
```

```r
autoplot(arima_mod_ar4)
```




```r
checkresiduals(arima_mod_ar4)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(4,0,0) with non-zero mean
Q* = 4.2365, df = 5, p-value = 0.5159

Model df: 5.   Total lags used: 10
```

Estimating MA model via Innovations algorithm

- First simulate some MA data:

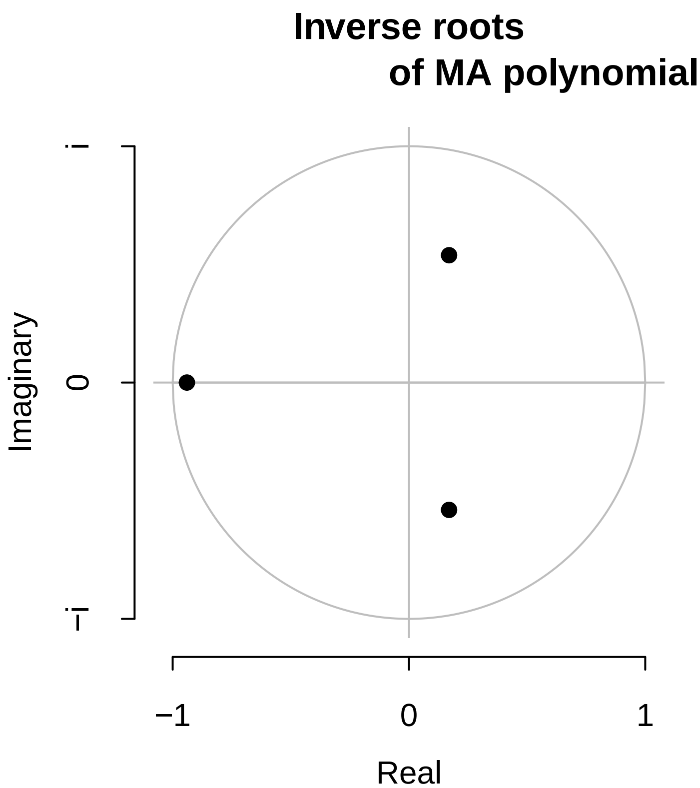
```
true_ma_coef_2=c(0.6, 0, 0.3)
my_MA3_model = list(ma=true_ma_coef_2,ar=NULL)
```

- Check the roots to make sure it is invertible

```
ma_roots<-polyroot(c(1,true_ma_coef_2))
ma_roots
```

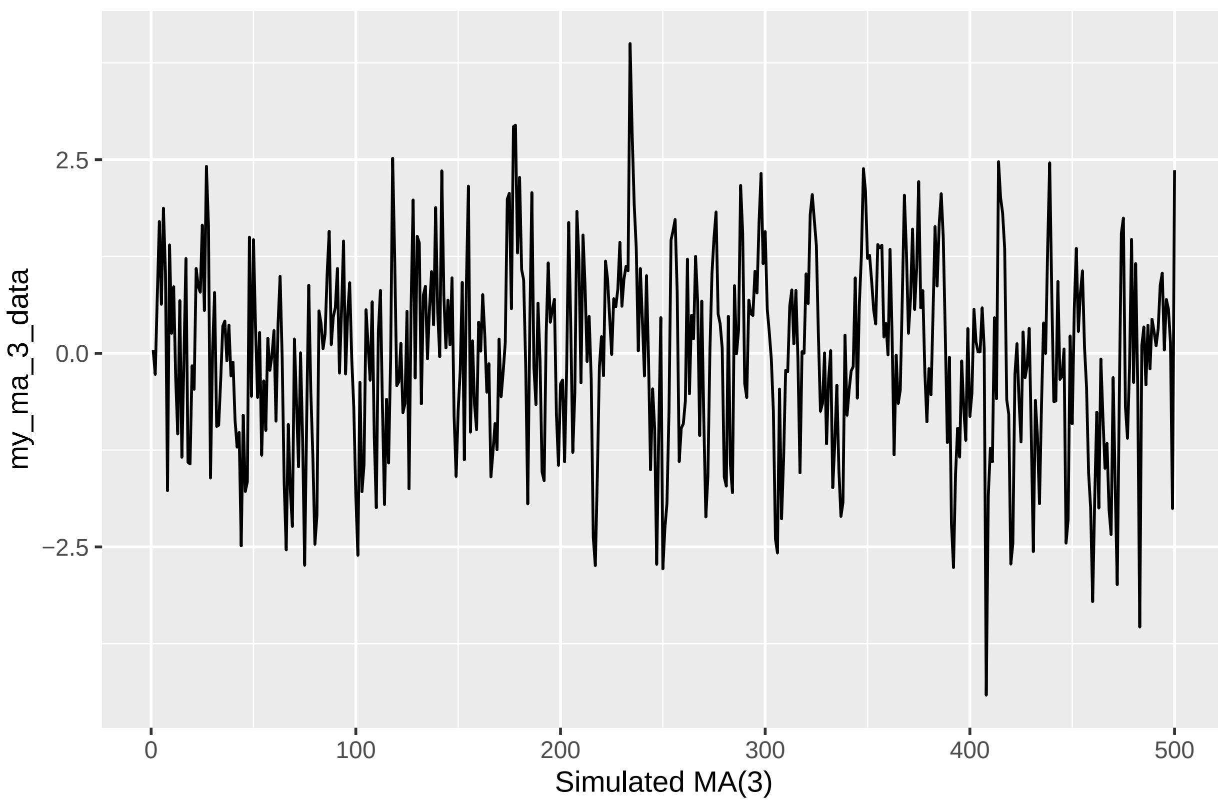
```
[1]  0.532073+1.687988i -1.064145+0.000000i  0.532073-1.687988i
```

```
forecast:::plot.armaroots(structure(list(roots=ma_roots),
                          class = "armaroots"),xlab="Real", ylab="Imaginary",main="Inverse roots
                          of MA polynomial")
```

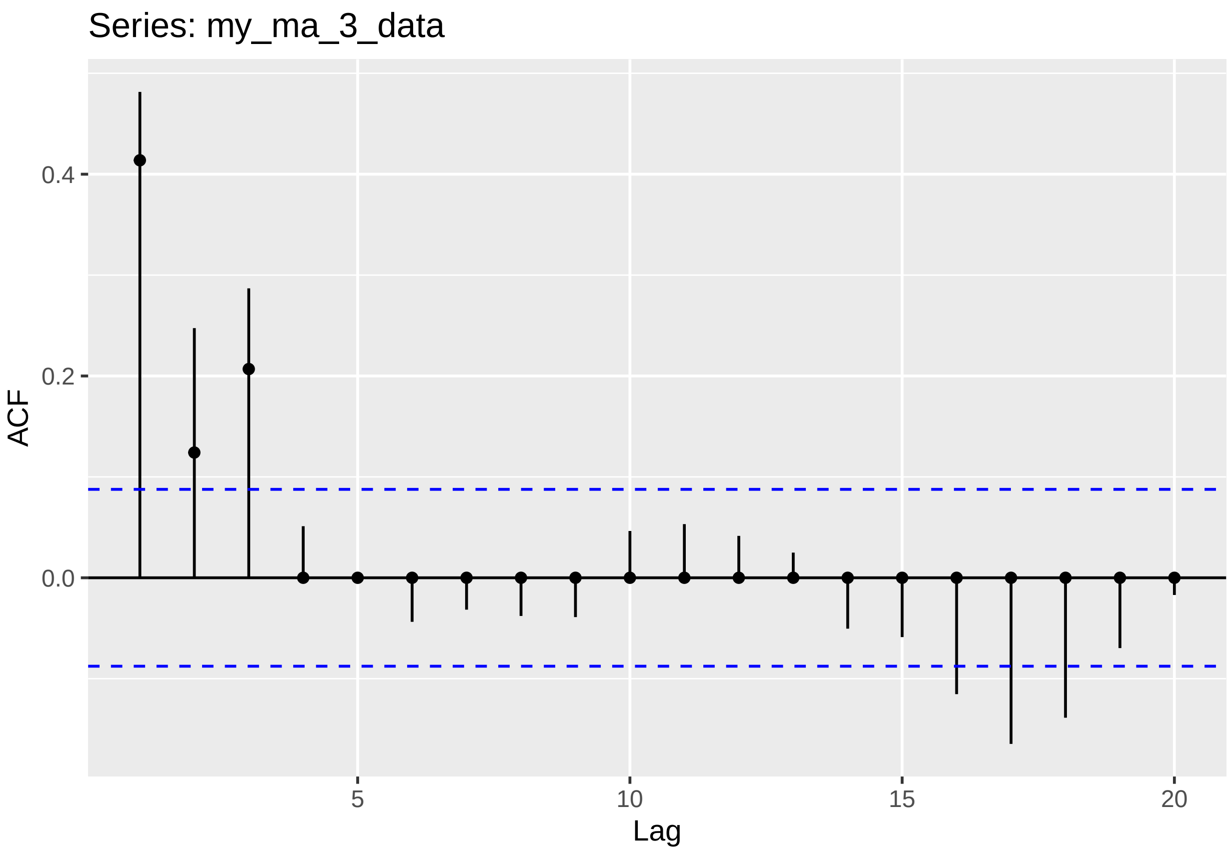


- Now generate some data and compare sample ACF and PACF to truth

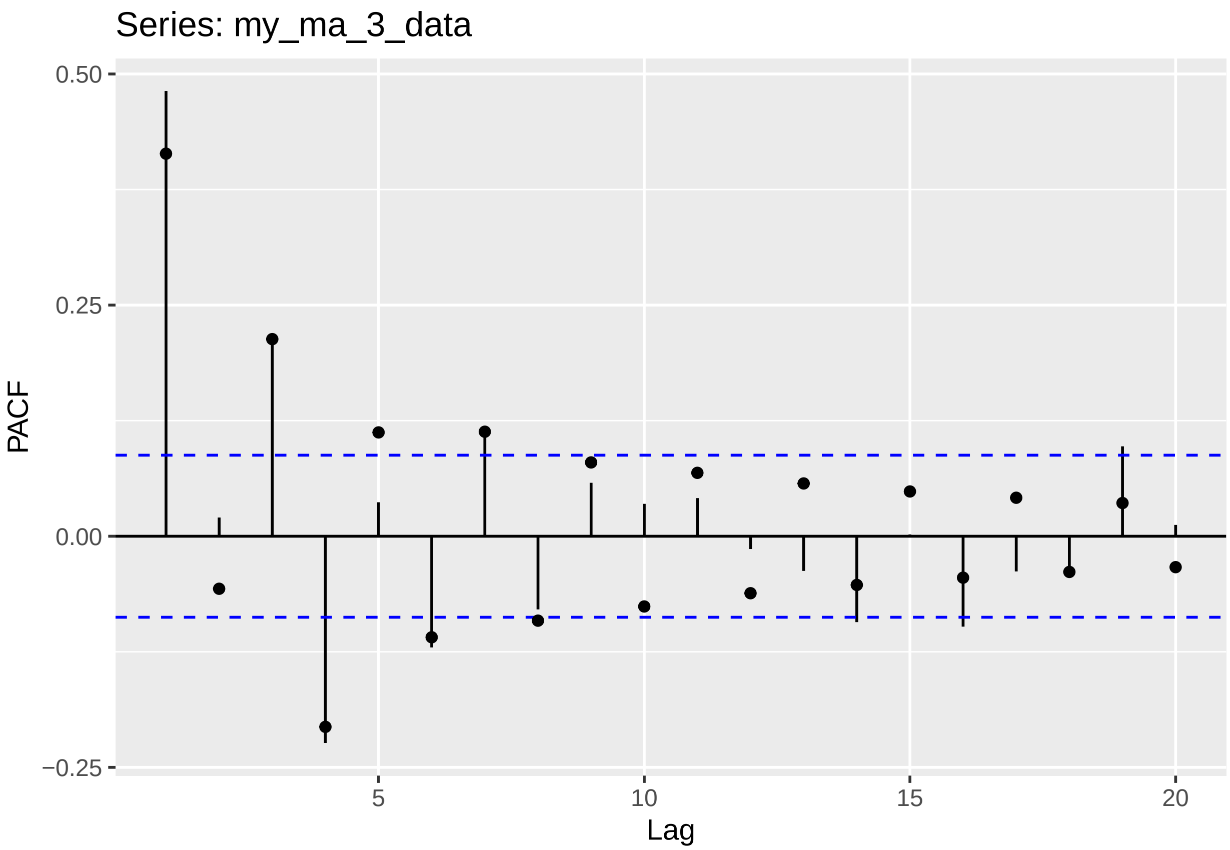
```
my_ma_3_data = arima.sim(my_MA3_model,n=500)
autoplot(my_ma_3_data) + xlab("Simulated MA(3)")
```



```
ggAcf(my_ma_3_data,lag.max=20) +
  geom_point(aes(y=c(ARMAacf(ma=true_ma_coef_2,ar=0,lag.max=20)[-1])))
```




```
ggPacf(my_ma_3_data,lag.max=20) +
  geom_point(aes(y=c(ARMAacf(ma=true_ma_coef_2,ar=0,pacf=TRUE,lag.max=20))))
```

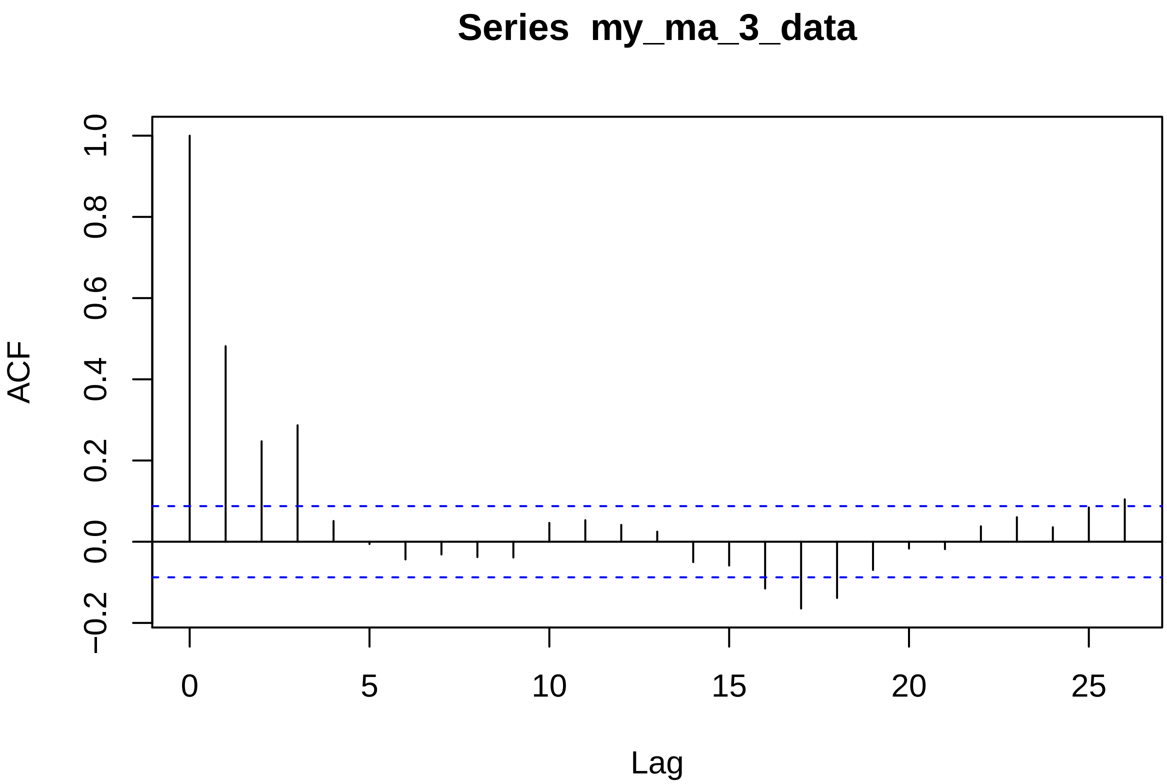


- Now estimate MA coefficients from innovations algorithm

```
ia(my_ma_3_data,q=4,m=20)
```

```
$phi
[1] 0

$theta
[1] 0.5455685 0.1240171 0.3871978 0.0831271

$sigma2
[1] 0.9976035

$aicc
[1] 1428.693

$se.phi
[1] 0

$se.theta
[1] 0.04472136 0.05094399 0.05124500 0.05409154
```

```
acf_vals<-acf(my_ma_3_data)
```




```
acf_vals
```

```
Autocorrelations of series 'my_ma_3_data', by lag

     0      1      2      3      4      5      6      7      8      9     10 
 1.000  0.482  0.247  0.287  0.051 -0.006 -0.044 -0.032 -0.038 -0.039  0.046 
    11     12     13     14     15     16     17     18     19     20     21 
 0.053  0.042  0.025 -0.050 -0.059 -0.115 -0.165 -0.139 -0.070 -0.017 -0.018 
    22     23     24     25     26 
 0.038  0.061  0.036  0.085  0.105 
```