Moving-blocks booststrap

ShanZhong

April 29, 2019

Introduction to Bootstrap

The normal bootstrap methods:

resample the data(individual observations) with replacement. The statistic of interest is computed from the resample, require a i.i.d sample.

Resampling residuals:

Fit a model to the data to get residuals and fitted value. Resample the residuals, add back to the fitted value to get a pseudo data(artificially generated data). Refit the model and then compute the satsitic of interest. Do not require the data to be i.i.d, but require residual to be i.i.d.

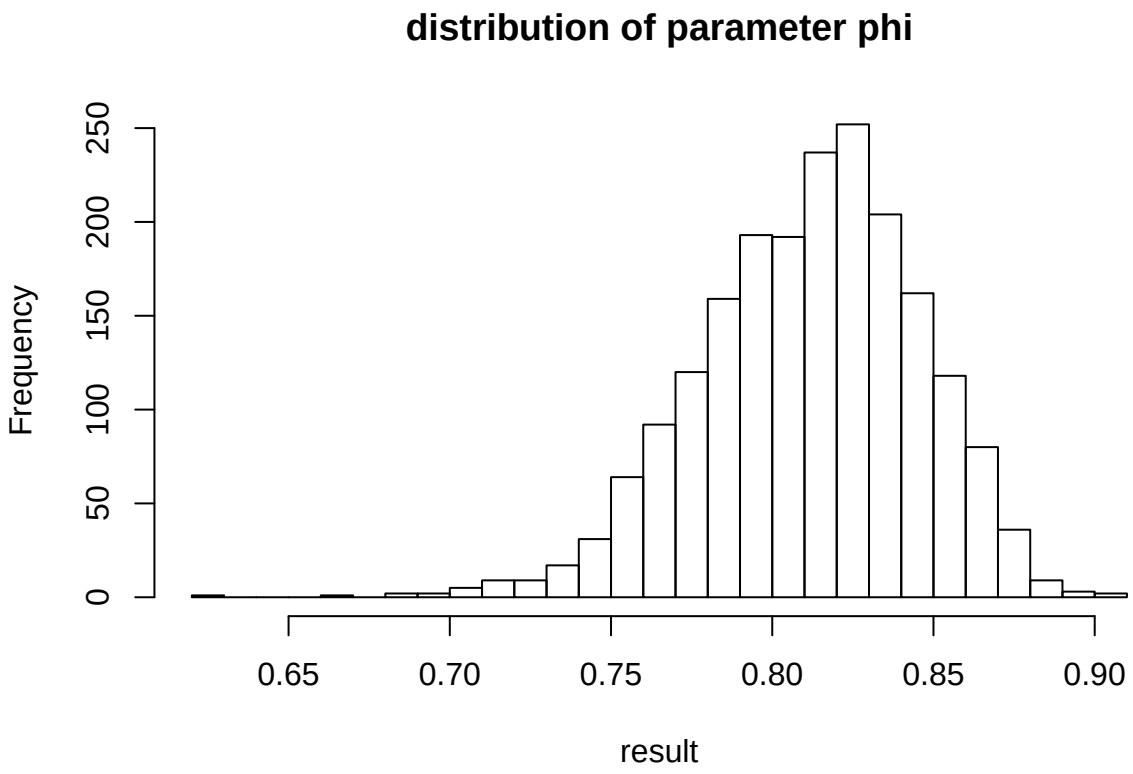
The idea is to estimate the model, and then use the residuals that are, by construction, close to being independent. Bootstrap these residuals and "back out" the observations using your estimated parameters.

Application of the residual based bootstrap methods is straightforward if the error distribution is specified to be an ARMA(p,q) process with known p and q

Here is an example using residual bootstrap to estimate for ar(1) parameter with true phi = 0.8

```
# specify parameters:
phi <- c(.8);
n <- 300; sigma <- 2
B<-2000
```

```
X <- arima.sim(n=n,list(ar=phi,sd=sigma))
fit<-arima(X,order=c(1,0,0))
result<-numeric()
for(iter in 1:B){
  pseudo.residual<-sample(fit$residuals,length(fit$residuals),replace = TRUE)
  Y<-numeric()
  Y[1]<-X[1]
  for(i in 2:length(X)){
    Y[i]<-Y[i-1]*fit$coef[1]+pseudo.residual[i]
  }
  result[iter]<-arima(Y,order=c(1,0,0))$coef[1]
}

hist(result,breaks=30,main="distribution of parameter phi")
```




```
mean(result)
```

```
## [1] 0.81146
```

```
fit$coef[1]
```

```
##      ar1
## 0.823387
```

However, if the structure of serial correlation is not tractable or is misspecified, the residual based methods will give inconsistent estimates

Block Boostrap:

resample blocks, do not require residual to be i.i.d.

The block bootstrap is used when the data, or the errors in a model, are correlated. In this case, a simple case or residual resampling will fail, as it is not able to replicate the correlation in the data.

The block bootstrap tries to replicate the correlation by resampling instead blocks of data. The block bootstrap has been used mainly with data correlated in time to maintaining the time series dependency structure within a pseudo-sample, but can also be used with data correlated in space, or among groups (so-called cluster data).

Here we introduce 3 types of block bootstrap:

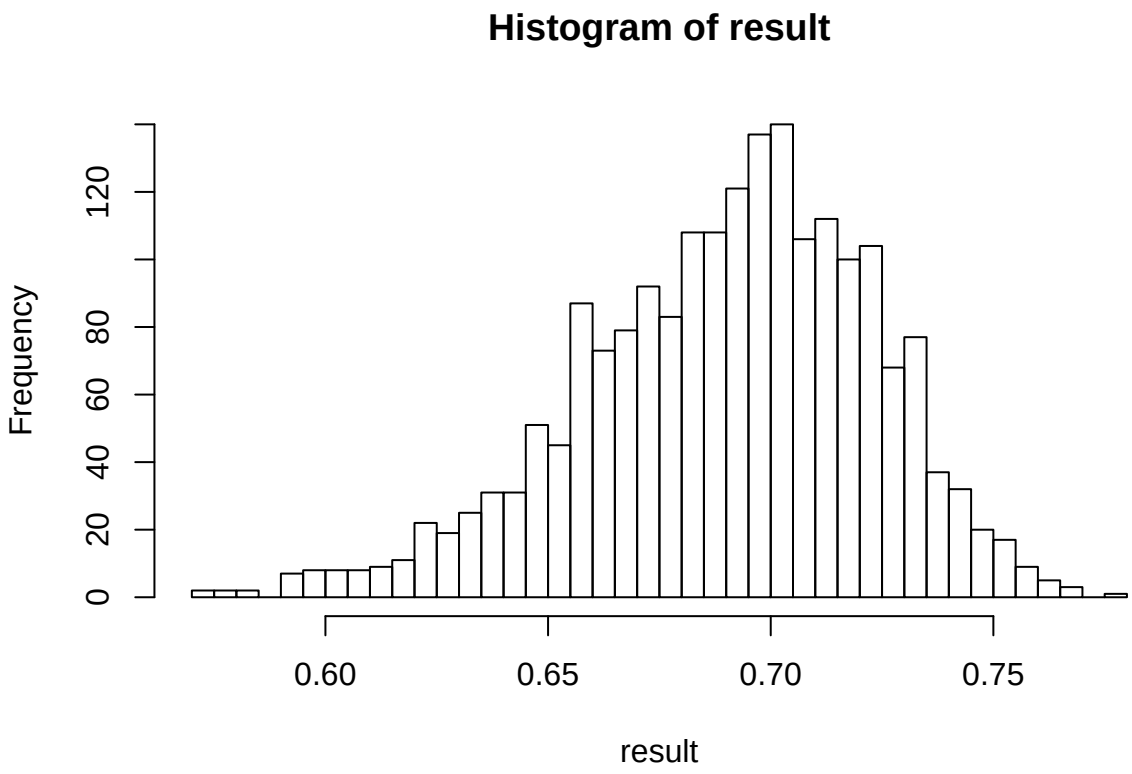
Non-overlapping block:the variable of interest is split into non-overlapping blocks.

Moving-blocks bootstrap: The moving block bootstrap (MBB) is applicable to stationary time series data. Sample whole blocks and concatenate them, in contrast to a single observation at a time.

MBB bootstrap example for a ar(1) model

Using block length = 25

```
# Here is an example for a ar(1) model with true phi = 0.8

# specify parameters:
phi <- c(.8);
n <- 300; sigma <- 2

X <- arima.sim(n=n,list(ar=phi,sd=sigma))
fit<-arima(X,order=c(1,0,0))
l<-25;B<-2000
result<-numeric(B)
  n<-length(X)
  # generate blocks
  X.blocks <- matrix(NA,n-l+1,l)
  for( j in 1:(n-l+1)){
    X.blocks[j,] <- X[j:(j+l-1)]
  }

  # run simulation
  for( iter in 1:B){
    X.star.mat <- X.blocks[sample(1:(n-l+1),floor(n/l) + 1,replace=TRUE),]
    X.star <- as.vector(t(X.star.mat))[1:n] # truncate to length n
    result[iter]<-arima(X.star,order=c(1,0,0))$coef[1]
  }

  hist(result,breaks =40)
```




```
  mean(result)
```

```
## [1] 0.6900144
```

```
  fit$coef[1]
```

```
##       ar1
## 0.7315008
```

Professor Karl have exmplained most concept of a moving blocks bootstrap

Servel points in summary:

The mean of the moving block bootstrap is biased

The MBB estimator of the variance of sqrt(n)*(X.bar) is also biased

MBB works with dependent data, however, the bootstrapped observations will not be stationary anymore by construction.

The moving bootstrap involves resampling possibly overlapping blocks. The mixed block bootstrep (MBB) does not force one to select a model and the only parameter required is the block length l.

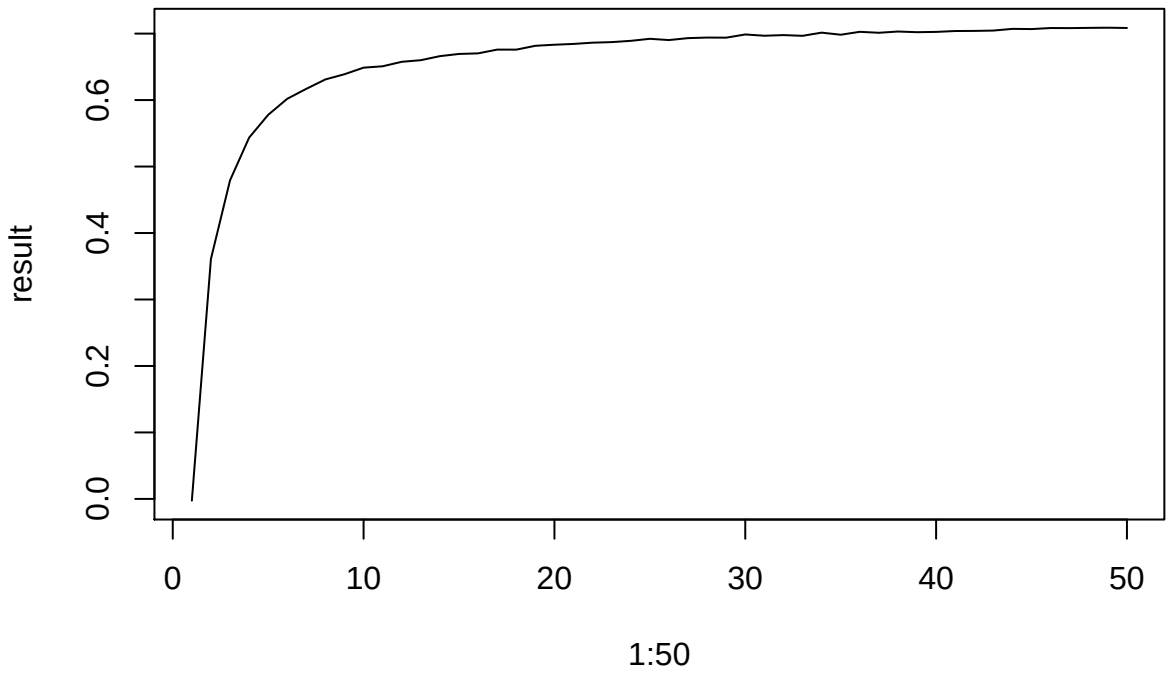
the relation of block length l and MBB parameter

```
B<-1000
value<-numeric(B)
```

```
result<-numeric(50)
# generate blocks

for(l in 1:50){
  X.blocks <- matrix(NA,n-l+1,l)
  for( j in 1:(n-l+1)){
    X.blocks[j,] <- X[j:(j+l-1)]
  }

  # run simulation
  for( iter in 1:B){
    X.star.mat <- X.blocks[sample(1:(n-l+1),floor(n/l) + 1,replace=TRUE),]
    X.star <- as.vector(t(X.star.mat))[1:n] # truncate to length n
    value[iter]<-arima(X.star,order=c(1,0,0))$coef[1]
  }

  result[l]<-mean(value)

}

fit$coef[1]
```

```
##       ar1
## 0.7315008
```

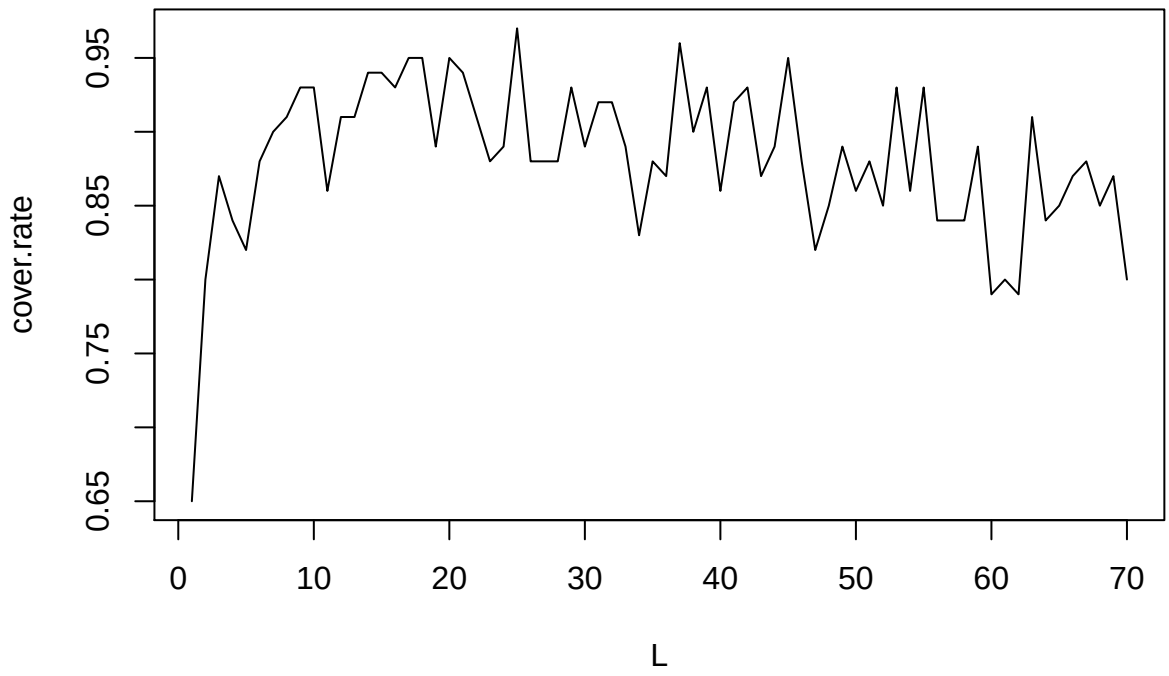
```
plot(1:50,result,type="l")
```



Here is the cover rate for different l, using a arma(2,1) model with n = 300

```
rm(list=ls())

library("data.table");library("ggplot2");library("forecast")

source("../lib/functions.R")

dat<-data.table(read.csv("../data/GS10.csv"))
```

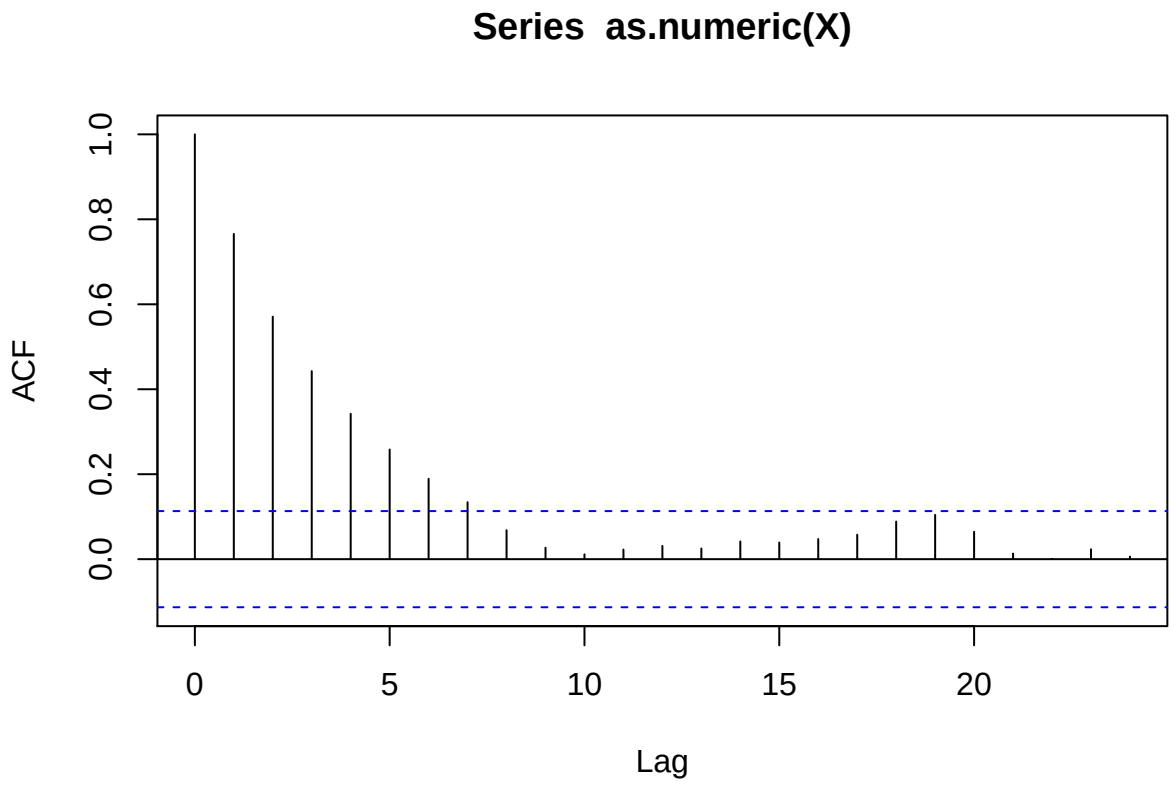
```
# specify parameters:
phi <- c(.3,.2);theta <- c(.4)
n <- 300; sigma <- 2

# l <- floor(sqrt(n))
# optimal l
L<-1:70

S <- 100 ; B <- 1000

# use the bootstrap to build a (1-alpha)*100% CI for the mean
# and to estimate Var(T_n) on S randomly generated datasets
lo.ci <- up.ci <- var.T.hat.analytic <- var.T.hat.MC <- numeric()
covered <- logical();cover.rate<-numeric();result<-c(NA,NA,NA)

for(l in L){
  for(s in 1:S){
    # generate data; compute sample mean.
    X <- arima.sim(n=n,list(ar=phi,ma=theta,sd=sigma))
    result<-get.mean.var.bootstrap(l,X,B)

    #get result
    var.T.hat.analytic<-result[1]
    var.T.hat.MC<-result[2]
    covered[s] <- result[3]
  }
  cover.rate[l]<-mean(covered)
}

library("ggplot2")

plot(L,cover.rate,type="l")
```



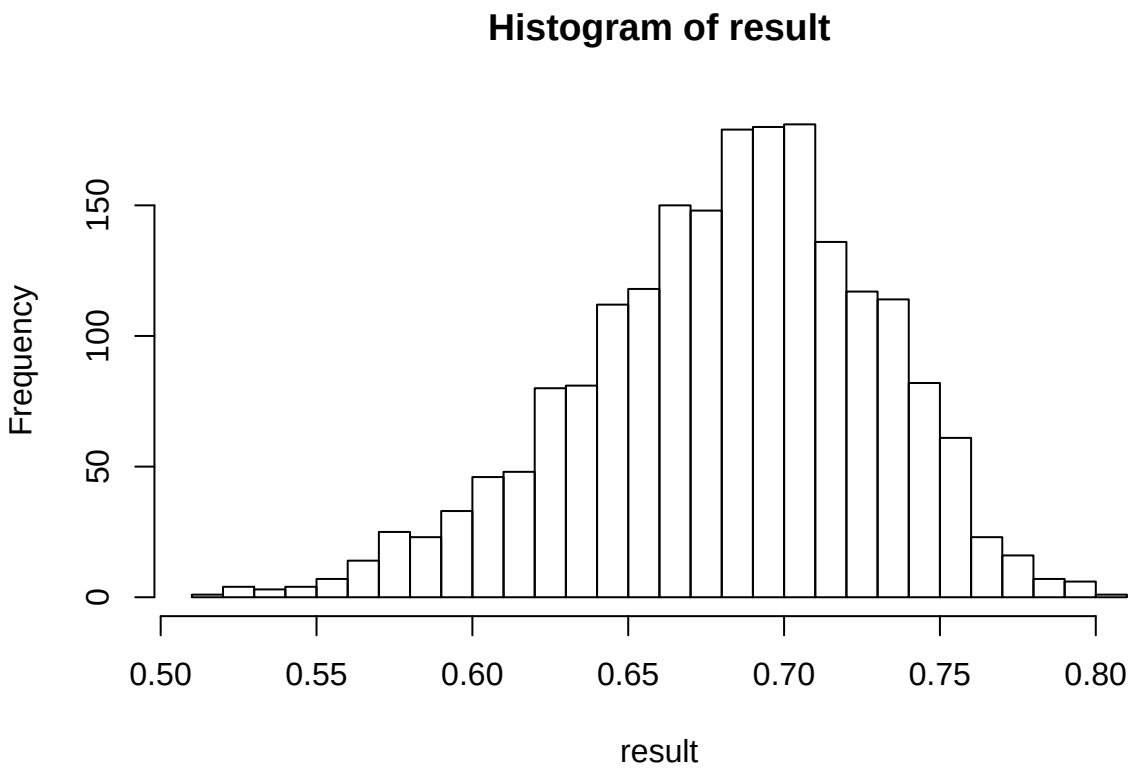
Stationary bootstrap:

Block size is a random variable in itself coming from a geometric distribution with some expected block size

```
B<-2000

# specify parameters:
phi <- c(.8);
n <- 300; sigma <- 2

X <- arima.sim(n=n,list(ar=phi,sd=sigma))
fit<-arima(X,order=c(1,0,0))

acf(as.numeric(X)) # have a look to determine appropriate block size
```




```
expectedl<-10
gprob=1/expectedl

X.star<-numeric(n)
result<-numeric(B)

for (iter in 1:B){
  # randomly choose a location
  loc <- round(runif(1,min = 1, max = n)) # loc for location
  for (i in 1:n){
    # In probability gprob, we take next observation, otherwise we start a new block
    if (runif(1)>gprob){loc = loc+1} else {loc = round(runif(1,min = 1, max = n))}
    if (loc>n){loc <- loc-n }# wrap the serie as a circule
    X.star[i] = X[loc]
  }
  result[iter]<-arima(X.star,order=c(1,0,0))$coef[1]
}

hist(result,breaks=30)
```




```
mean(result)
```

```
## [1] 0.6814871
```

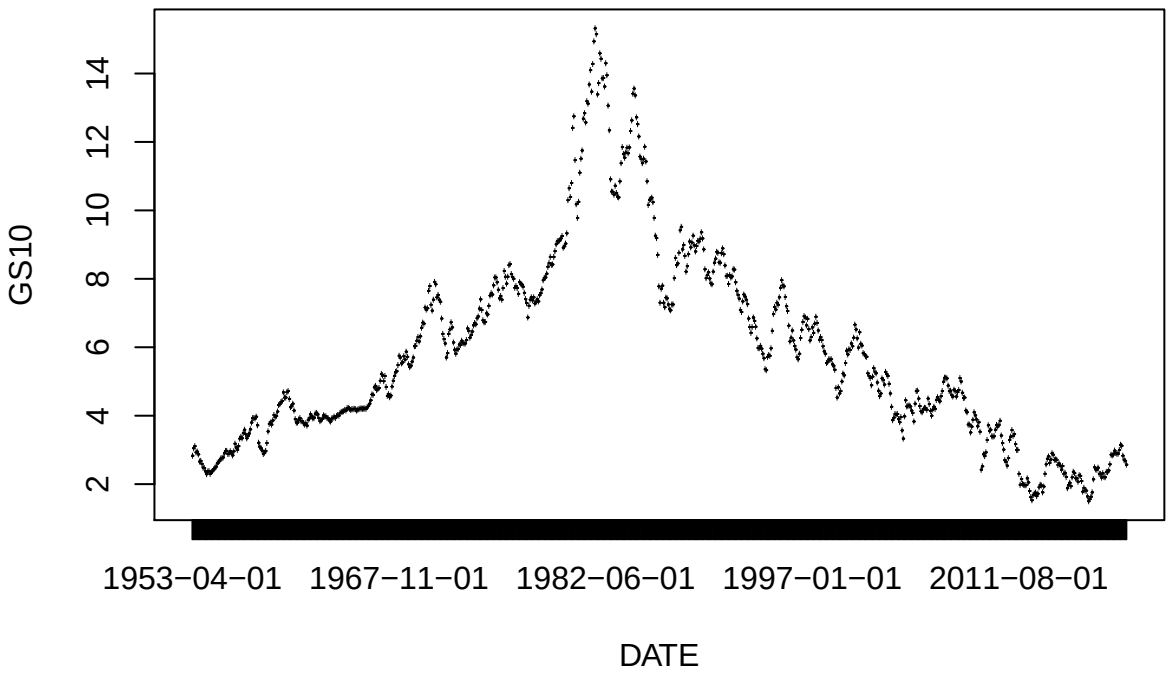
```
fit$coef[1]
```

```
##       ar1
## 0.7699157
```

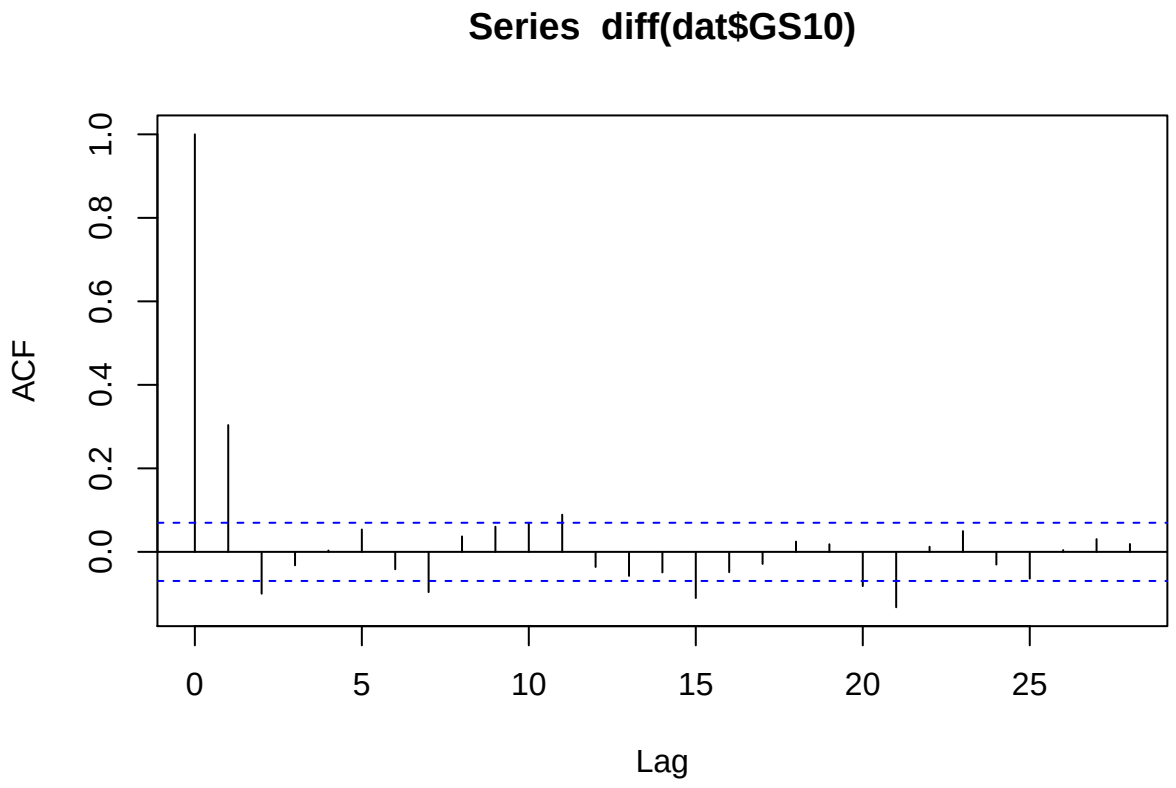
Some real data exaple

let's look at the interest rate data using 10-Year Treasury Constant Maturity Rate

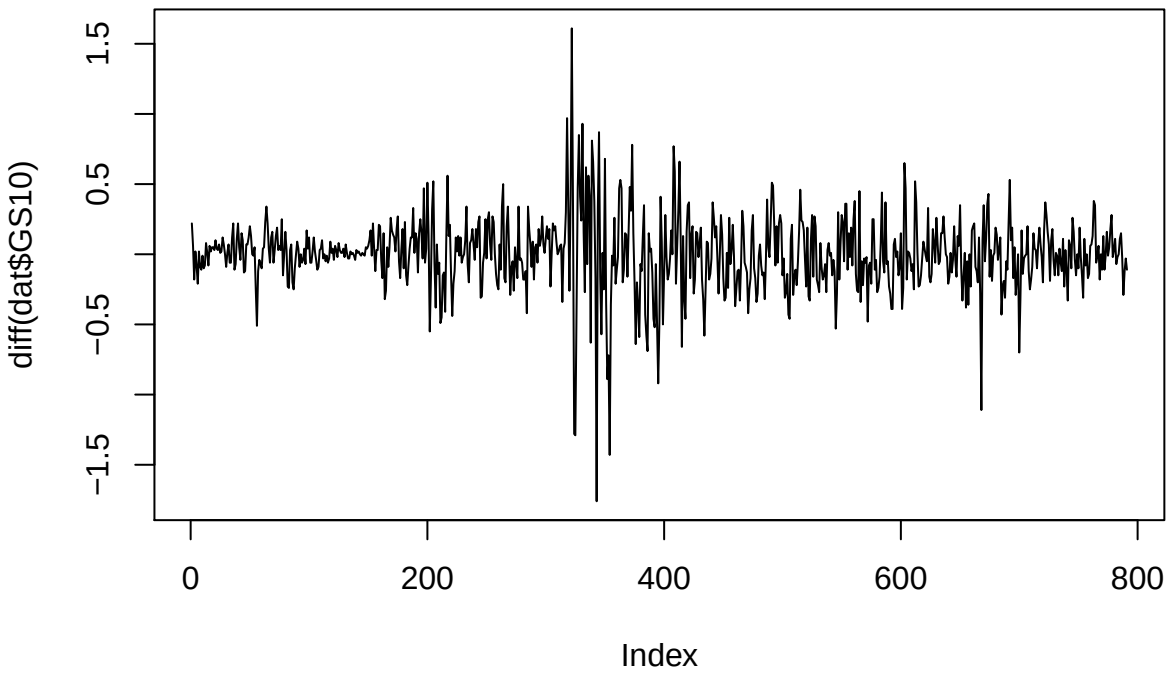
```
plot(dat)
```



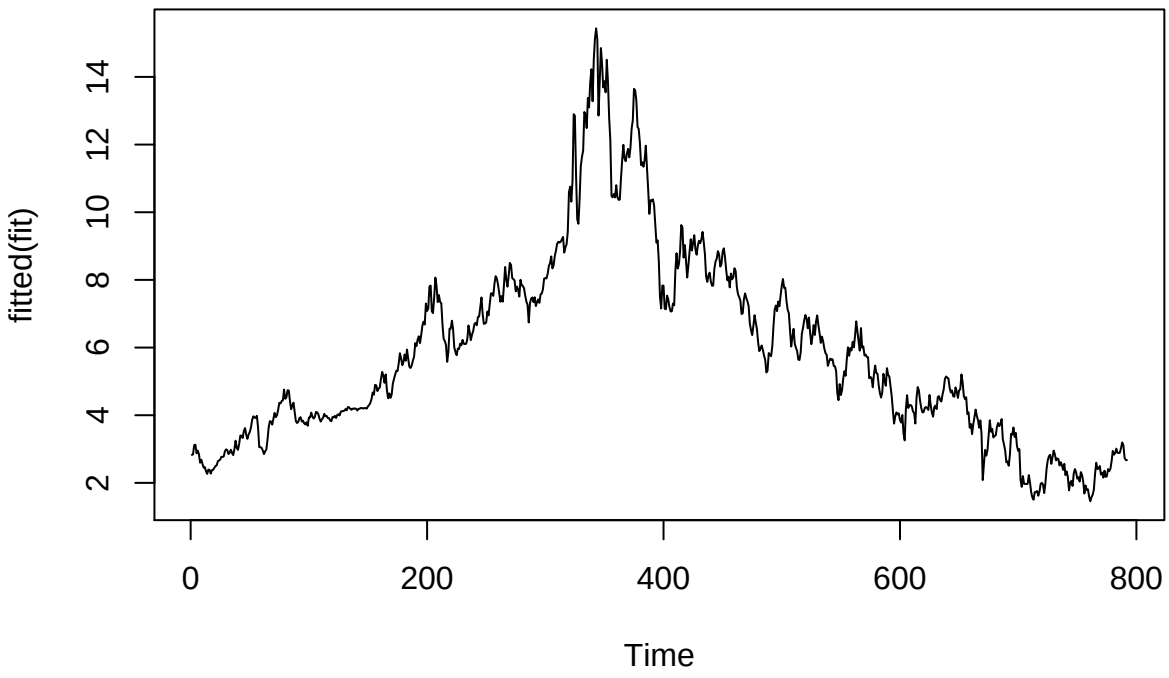

```
acf(diff(dat$GS10))
```



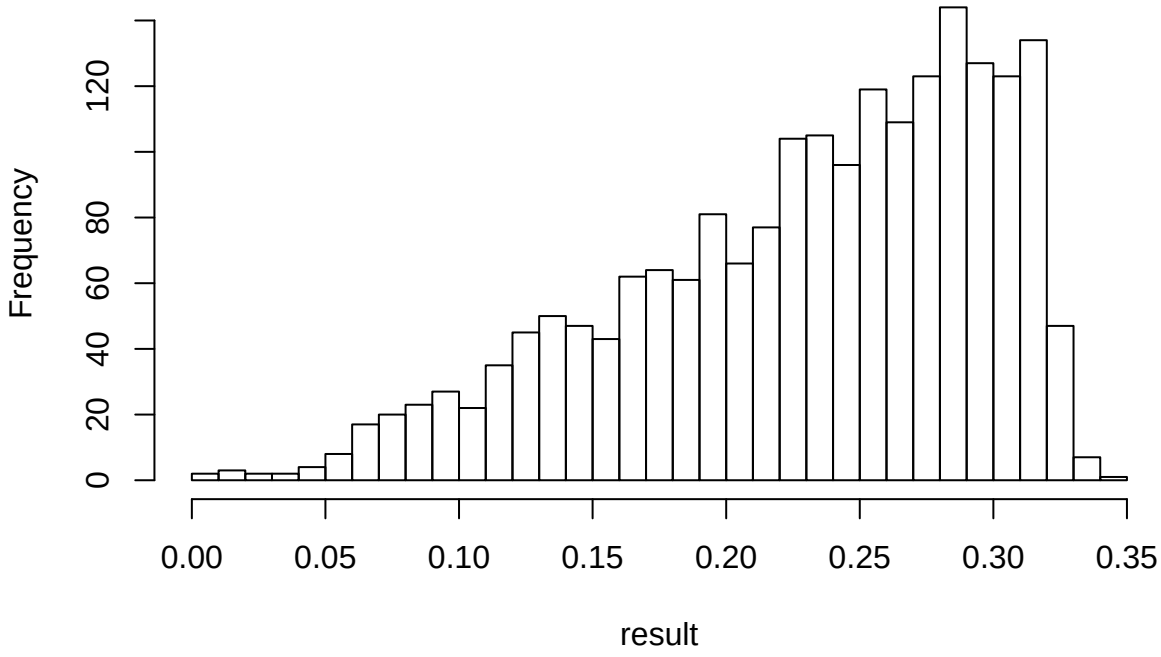

```
plot(diff(dat$GS10),type="l")
```




```
#Fit with a arima(1,1,0) model

fit<-arima(dat$GS10,order=c(1,1,0))

plot(fitted(fit))
```




```
X<-dat$GS10

# Moving block use l = 350
l<-350;B<-2000

result<-numeric(B)
n<-length(X)
# generate blocks
X.blocks <- matrix(NA,n-l+1,l)
for( j in 1:(n-l+1)){
  X.blocks[j,] <- X[j:(j+l-1)]
}

# run simulation
for( iter in 1:B){
  X.star.mat <- X.blocks[sample(1:(n-l+1),floor(n/l) + 1,replace=TRUE),]
  X.star <- as.vector(t(X.star.mat))[1:n] # truncate to length n
  result[iter]<-arima(X.star,order=c(1,1,0))$coef[1]
}

hist(result,breaks =40)
```




```
mean(result)
```

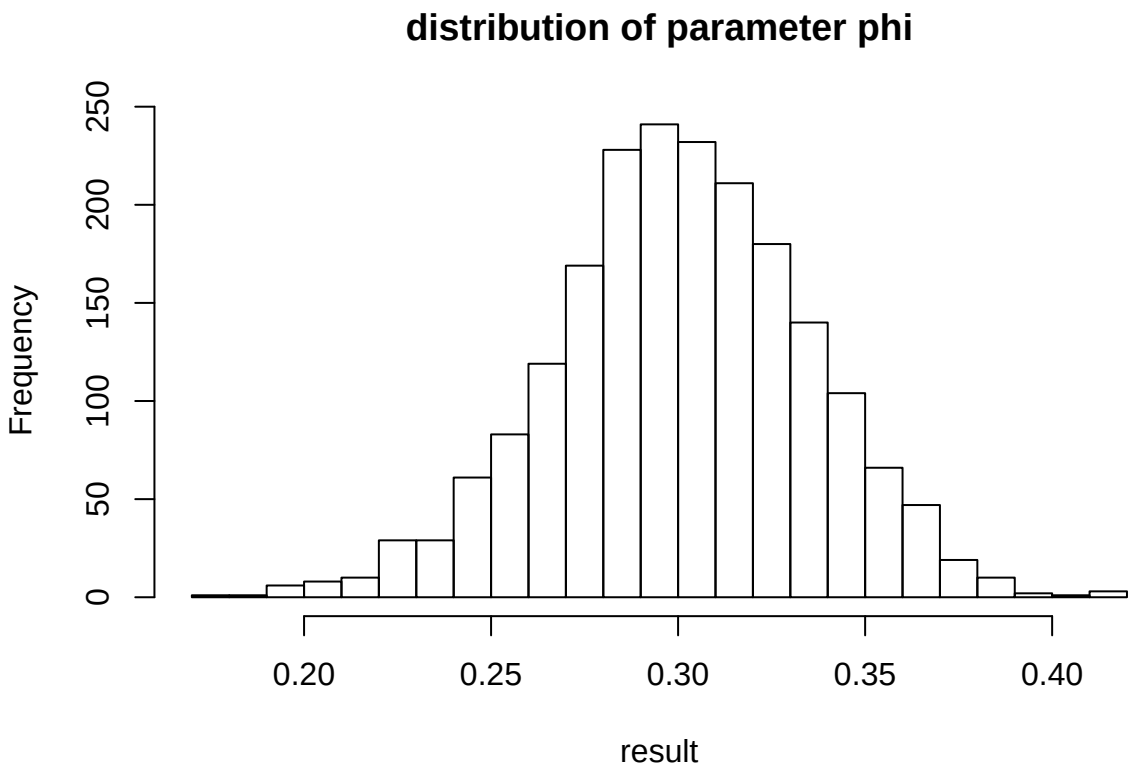
```
## [1] 0.2298915
```

```
fit$coef[1]
```

```
##       ar1
## 0.3035761
```

use residual based bootstrap

```
X <- diff(dat$GS10)
fit<-arima(X,order=c(1,0,0))

result<-numeric()
for(iter in 1:B){
  pseudo.residual<-sample(fit$residuals,length(fit$residuals),replace = TRUE)
  Y<-numeric()
  Y[1]<-X[1]
  for(i in 2:length(X)){
    Y[i]<-Y[i-1]*fit$coef[1]+pseudo.residual[i]
  }
  result[iter]<-arima(Y,order=c(1,0,0))$coef[1]
}

hist(result,breaks=30,main="distribution of parameter phi")
```




```
mean(result)
```

```
## [1] 0.3006882
```

```
fit$coef[1]
```

```
##       ar1
## 0.3035615
```