W4.Partial Autocorrelation(PACF)

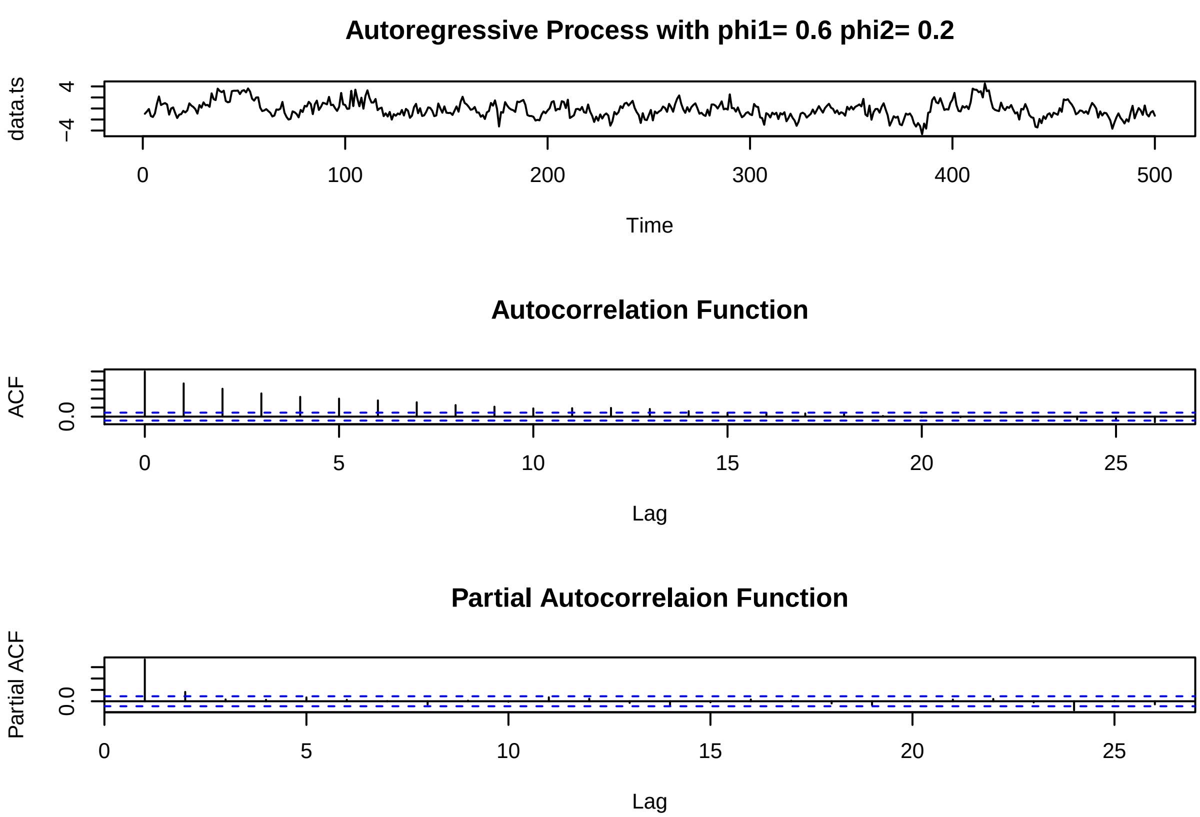
Yansong Liu

5/21/2020

time series data-not obvious model stachastic process ACF-characterize an AR or MA process:formula,predict shape -MA(q) has ACF cut off after q lags

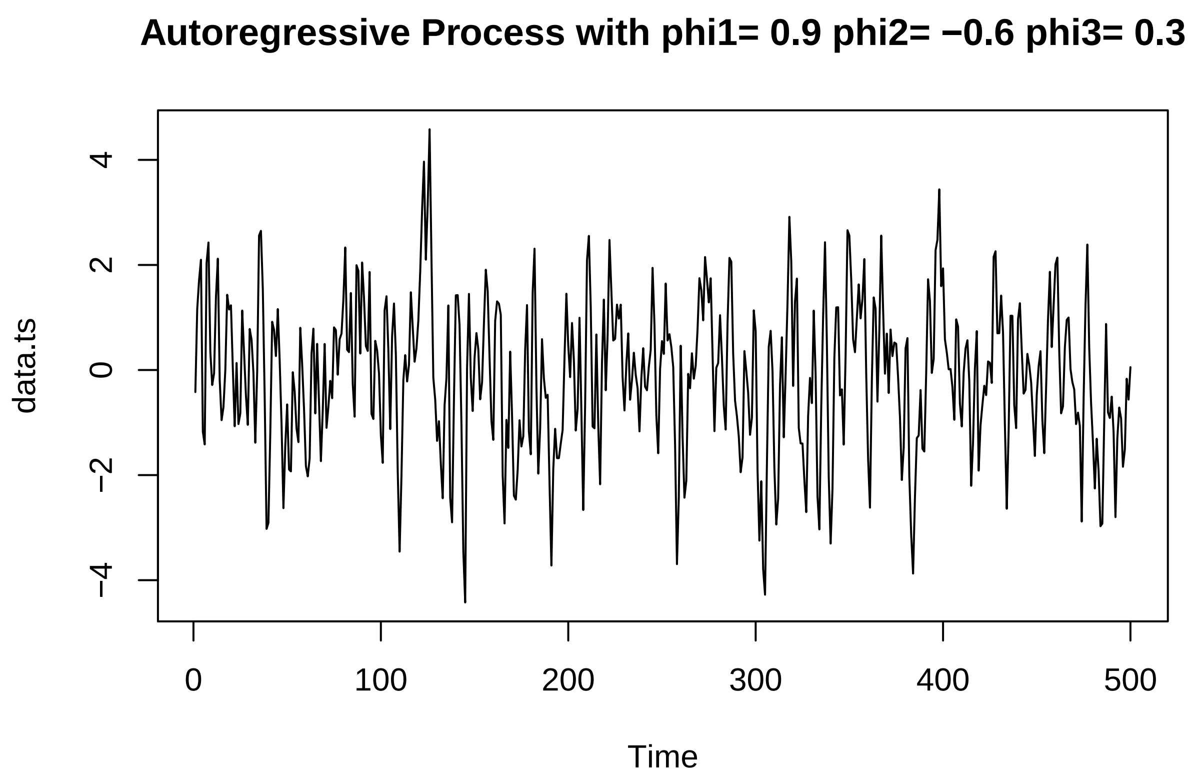
Simulation AR(2) process Give model parameters xt=phi1*xt-1+phi2*xt-2+zt

```r
rm(list = ls(all=TRUE))
#generate data-get rid of all arrays stored
#-keep nice,organized=erase blackboard

#model parameters
phi.1=0.6
phi.2=0.2

#simulate AR process-generate data
data.ts=arima.sim(n=500,list(ar=c(phi.1,phi.2)))

par(mfrow=c(3,1))
plot(data.ts,main=paste('Autoregressive Process with phi1=', phi.1, 'phi2=', phi.2))#time series plot
acf(data.ts, main='Autocorrelation Function')  #ACF
acf(data.ts, type='partial',  main='Partial Autocorrelaion Function')  #PACF
```



AR(3) process simulation Given parameters-random->stationary

```r
phi.1=0.9
phi.2=-0.6
phi.3=0.3

#simulate AR process-list of AR parameters-generate 500 data
data.ts=arima.sim(list(ar=c(phi.1,phi.2,phi.3)),n=500)

plot(data.ts,main=paste('Autoregressive Process with phi1=',phi.1,'phi2=',phi.2,'phi3=',phi.3))
```



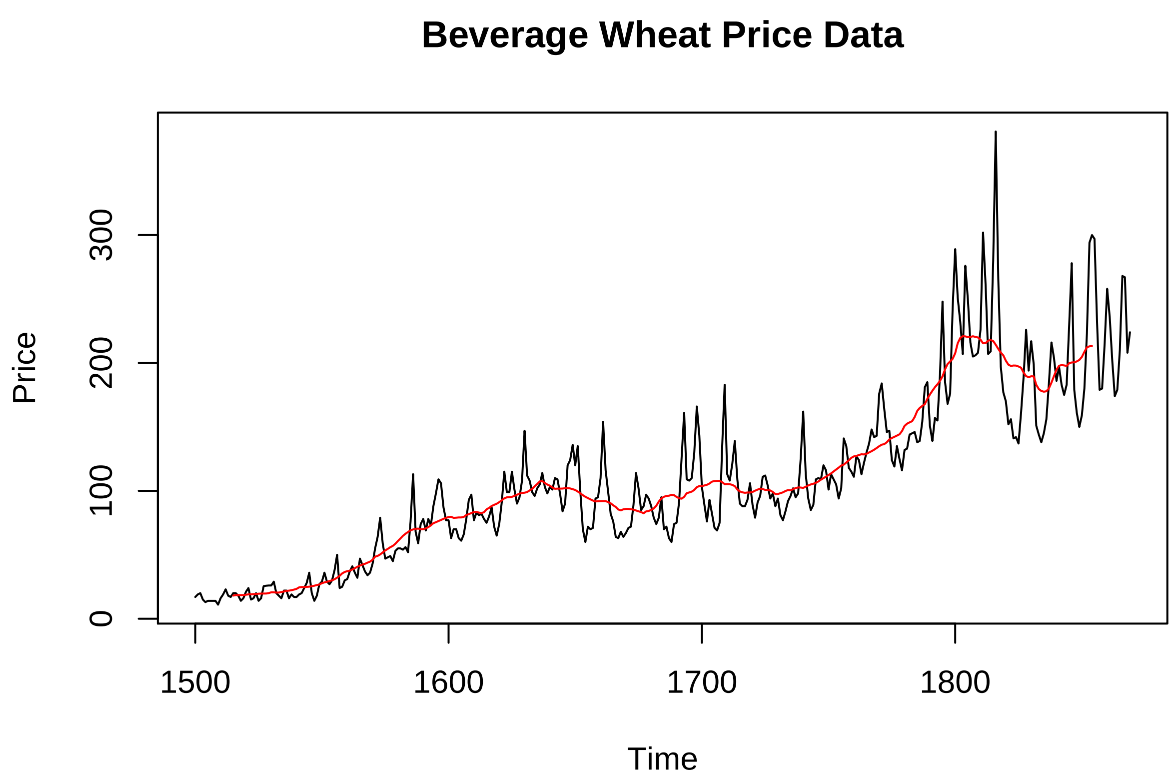
Beveridge Wheat Price Data Set

```r
library(tseries)
```

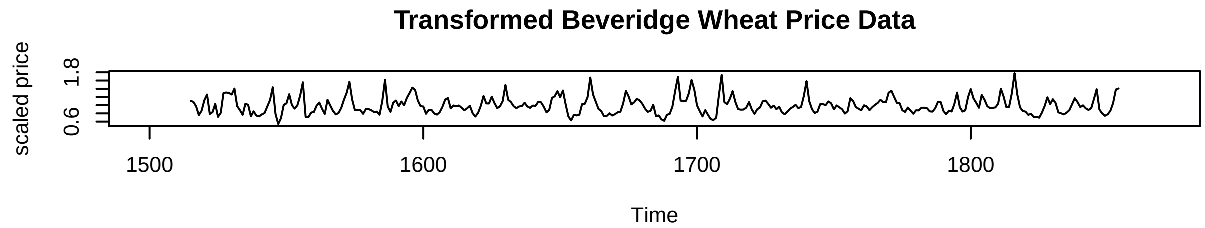
```
## Registered S3 method overwritten by 'xts':
##   method     from
##   as.zoo.xts zoo

## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

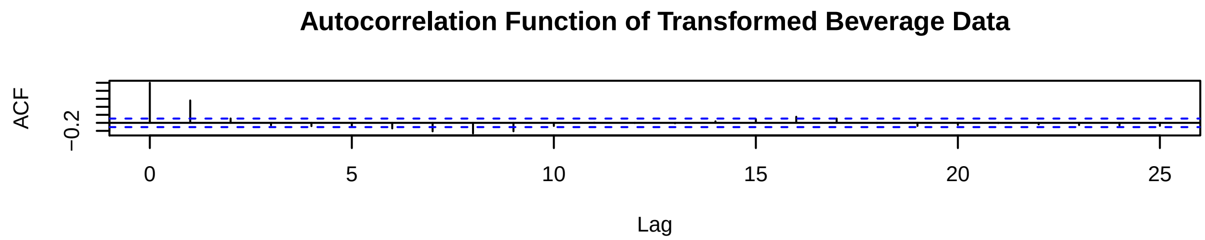
```r
data(bev)
plot(bev,ylab='Price',main='Beverage Wheat Price Data')
```

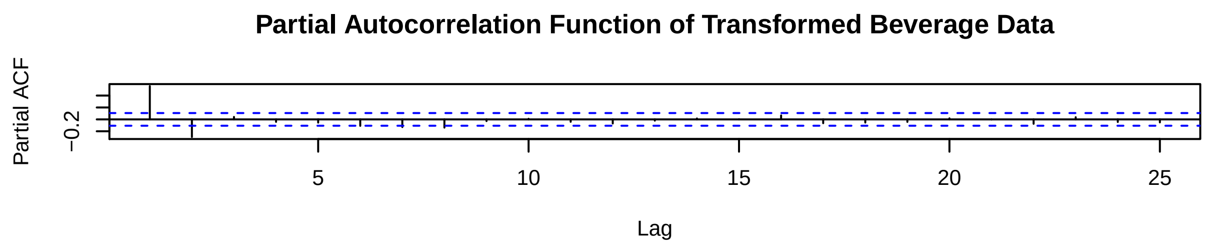
```r
#simple moving average-each data point->scaling-smoothing-create stationary process
#-filter
#-grab 31 data points-15 data points on 2 side-15 up, 15 down
#-surround particular (1/31)data point at any time
#-along representation 31 data points
#-MA:get rid of fluctuation-along with trend
bev.MA=filter(bev,sides = 2,rep(1/31,31))

lines(bev.MA,col='red')
```




```r
par(mfrow=c(3,1))

# bev.MA-scale at each point by MA process
Y=bev/bev.MA

# some data points at beginning & end -not represent in Y-show NA
plot(Y,ylab='scaled price',main='Transformed Beveridge Wheat Price Data')

# na.omit-clean-up function-get rid of NA data points
#-acf doesn't like missing data-first,last 15 numbers are omitted
acf(na.omit(Y),main='Autocorrelation Function of Transformed Beverage Data')

acf(na.omit(Y),type = 'partial',main='Partial Autocorrelation Function of Transformed Beverage Data')
```








```r
#ar()-estimate coefficents-fit time series model AR process
#-order.max=5-allow up to 5 terms in model
ar(na.omit(Y),order.max=5)
```

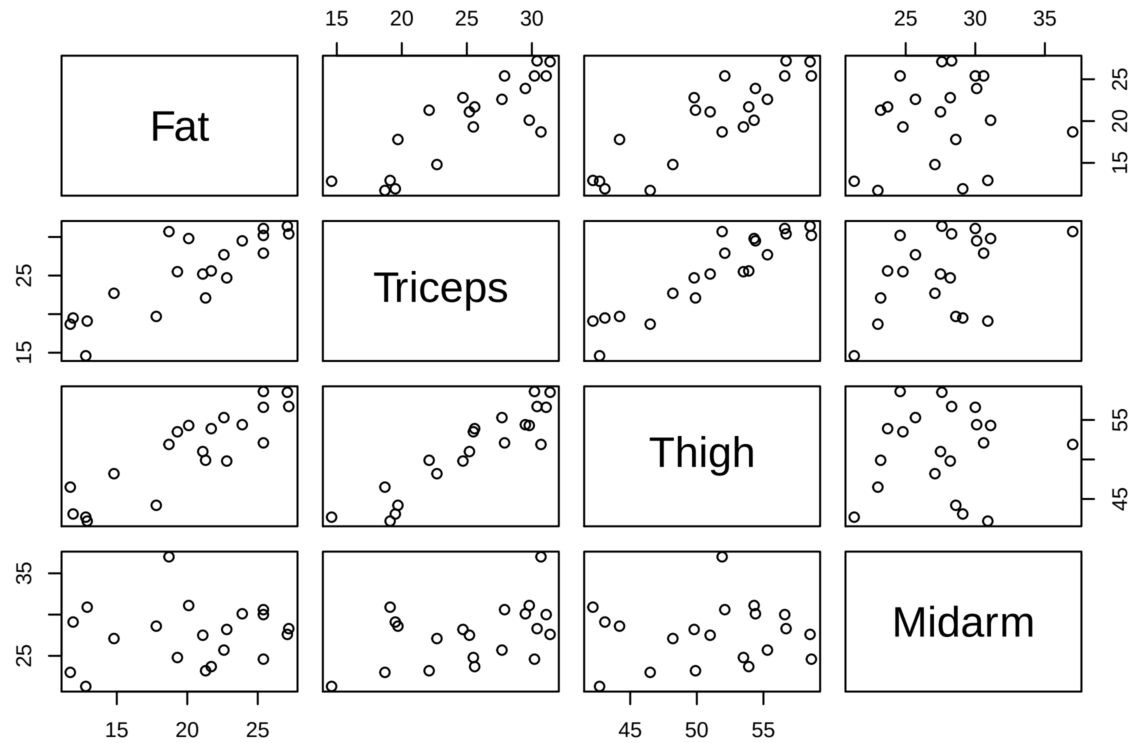
```
##
## Call:
## ar(x = na.omit(Y), order.max = 5)
##
## Coefficients:
##       1        2
##  0.7239  -0.2957
##
## Order selected 2  sigma^2 estimated as  0.02692
```

```r
#order selected 2-fit PACF plan
#-AR(p) has a PACF-cut off after p lags
```

PACF Concept & Development time series-a set of related r.v.-redundancy-conditional dependencies -decide a proposed variable is useful-control for the presence of terms already in a model -measure a correlation between some r.v-remove effect of other r.v

```r
library(isdals)
data("bodyfat")

# related variables-see with pair plot
```

```r
attach(bodyfat)  #call variables directly
pairs(cbind(Fat,Triceps,Thigh,Midarm))
```




```r
cor(cbind(Fat, Triceps,Thigh,Midarm))
```

```
##                Fat   Triceps     Thigh    Midarm
## Fat      1.0000000 0.8432654 0.8780896 0.1424440
## Triceps  0.8432654 1.0000000 0.9238425 0.4577772
## Thigh    0.8780896 0.9238425 1.0000000 0.0846675
## Midarm   0.1424440 0.4577772 0.0846675 1.0000000
```

```r
# Fat~Triceps, Fat~Thigh, Triceps~Thigh highly correlated
```

control for/partial out-measure correlation

```r
# do a linear regression of Fat on Thigh-account for effect of Thigh on Fat
Fat.hat=predict(lm(Fat~Thigh))
Triceps.hat=predict(lm(Triceps~Thigh))

#remove linear relationship of Thigh-residuals-control for/partial out Thigh
#find partial correlation of Fat-Triceps
cor((Fat-Fat.hat),(Triceps-Triceps.hat))
```

```
## [1] 0.1749822
```

```r
Fat.hat=predict(lm(Fat~Thigh+Midarm))
Triceps.hat=predict(lm(Triceps~Thigh+Midarm))
cor((Fat-Fat.hat),(Triceps-Triceps.hat))
```

```
## [1] 0.33815
```

```r
library(ppcor)
```

```
## Loading required package: MASS
```

```r
pcor(cbind(Fat,Triceps,Thigh,Midarm))$estimate
```

```
##                 Fat   Triceps      Thigh     Midarm
## Fat       1.0000000 0.3381500 -0.2665991 -0.3240520
## Triceps   0.3381500 1.0000000  0.9963725  0.9955918
## Thigh    -0.2665991 0.9963725  1.0000000 -0.9926612
## Midarm   -0.3240520 0.9955918 -0.9926612  1.0000000
```

Estimate model parameters-AR(2) process -Yule-Walker equations in matrix form xt=phi1*x(t-1)+phi2*x(t-2)+zt zt~N(0,sigma^2)

```r
#set seed a common number-reproduce the same datasets
set.seed(2017)

#model parameters-estimate them
sigma=4
phi=NULL
phi[1:2]=c(1/3,1/2)
phi
```

```
## [1] 0.3333333 0.5000000
```

```r
#number of data points
n=10000

#simulate ar process
ar.process=arima.sim(model=list(ar=c(1/3, 1/2)),sd=4,n)
ar.process[1:5]
```

```
## [1] 4.087685 5.598492 3.019295 2.442354 5.398302
```

```r
#find and name 2nd and 3rd sample autocorrelation-r(1),r(2)
#acf[1]=r(0)
r=NULL
r[1:2]=acf(ar.process,plot = F)$acf[2:3]
r
```

```
## [1] 0.6814103 0.7255825
```

```r
#matrix R
R=matrix(1,2,2)  #matrix-entries all 1-dimension 2x2
R
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    1
```

```r
# edit R
#-only diagonal entries are 1
#-others are r(1)
R[1,2]=r[1]
R[2,1]=r[1]
R
```

```
##           [,1]      [,2]
## [1,] 1.0000000 0.6814103
## [2,] 0.6814103 1.0000000
```

```r
#b-column vector on the right
b=matrix(r,nrow = 2,ncol = 1)
b
```

```
##           [,1]
## [1,] 0.6814103
## [2,] 0.7255825
```

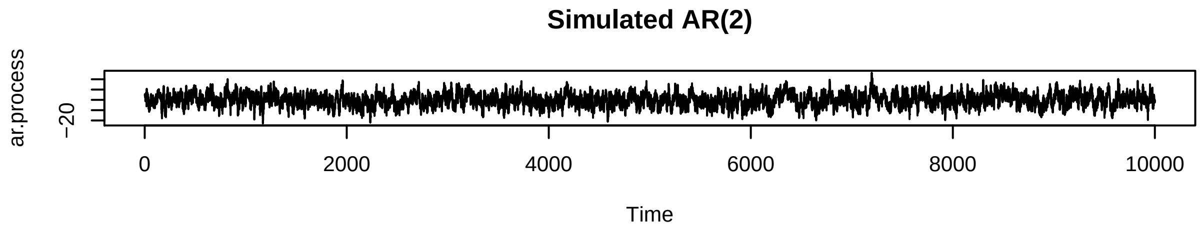
```r
#solve(R,b)-solve Rx=b-give x=R^(-1)b
phi.hat=solve(R,b)
phi.hat
```

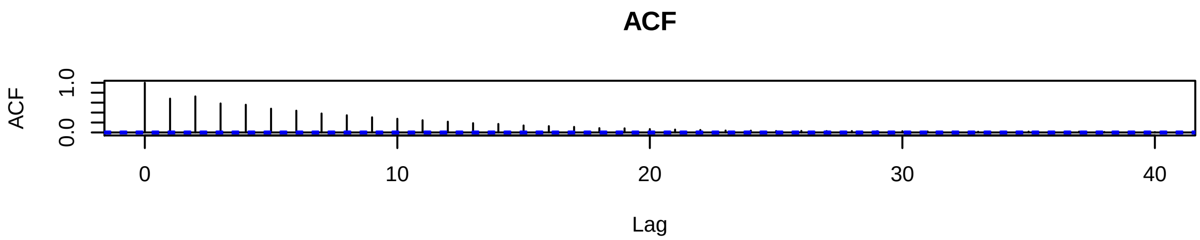
```
##           [,1]
## [1,] 0.3490720
## [2,] 0.4877212
```

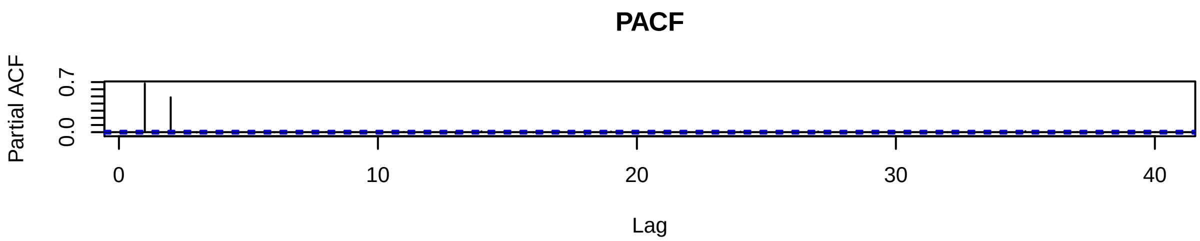
```r
#variance estimation
c0=acf(ar.process, plot = F)$acf[1]
var.hat=c0*(1-sum(phi.hat*r))
var.hat
```

```
## [1] 0.4082568
```

```r
#plot time series, along with ACF, PACF
par(mfrow=c(3,1))
plot(ar.process,main='Simulated AR(2)')
acf(ar.process,main='ACF')
pacf(ar.process,main='PACF')
```







Estimation of model parameters of an AR(3) simulation xt=phi1*x(t-1)+phi2*x(t-2)+phi3*x(t-3)+zt zt~N(0,sigma^2)

```
set.seed(2017)
sigma=4
phi=NULL
phi[1:3]=c(1/3,1/2,7/100)
n=100000

#Simulation AR(3) process
ar3.process=arima.sim(model = list(ar=c(1/3,1/2,7/100)),sd=4,n)

r=acf(ar3.process,plot = F)$acf[2:4]
r
```

```
## [1] 0.7859646 0.8180901 0.7369167
```

```
R=matrix(1,3,3)
R[1,2]=r[1]
R[1,3]=r[2]
R[2,1]=r[1]
R[2,3]=r[1]
R[3,1]=r[2]
R[3,2]=r[1]
R
```

```
##             [,1]      [,2]      [,3]
```

```
## [1,] 1.0000000 0.7859646 0.8180901
## [2,] 0.7859646 1.0000000 0.7859646
## [3,] 0.8180901 0.7859646 1.0000000
```

```r
#b-column vector on the right
b=matrix(,3,1)  #b-column vector with no entries
b[1,1]=r[1]
b[2,1]=r[2]
b[3,1]=r[3]
b
```

```
##           [,1]
## [1,] 0.7859646
## [2,] 0.8180901
## [3,] 0.7369167
```

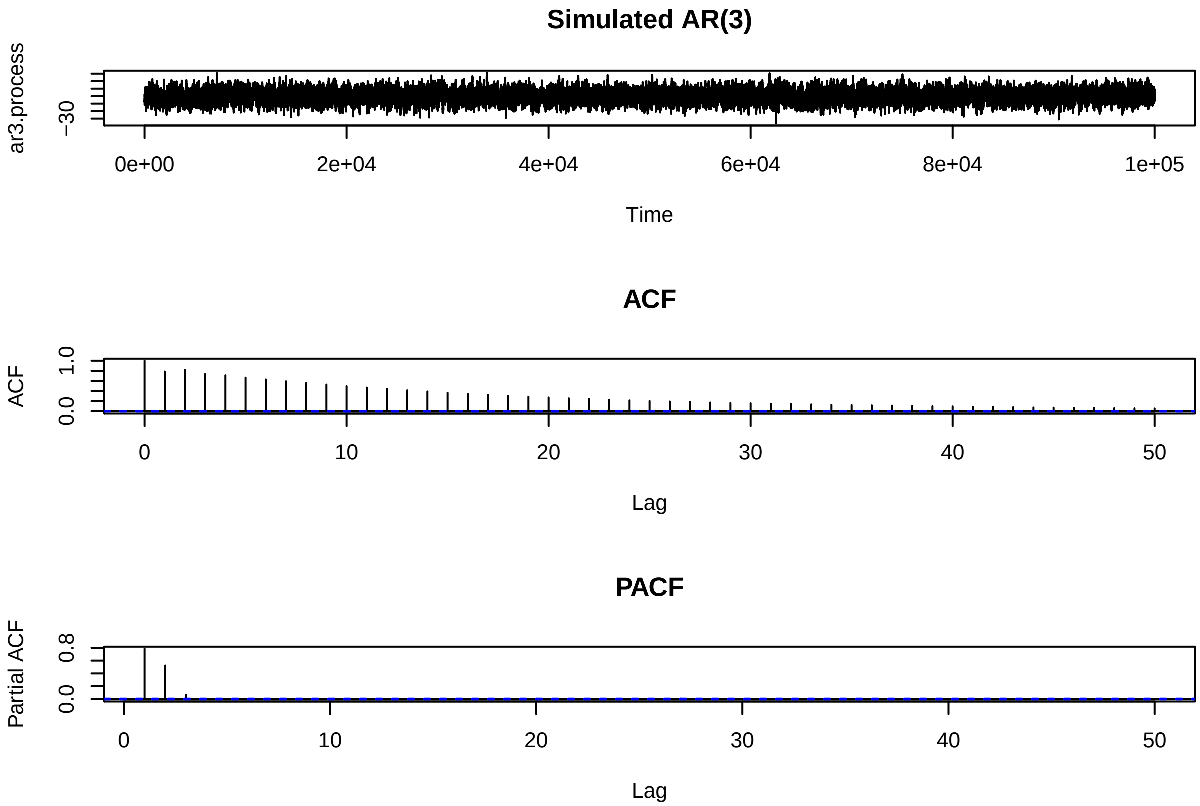
```r
#solve Rx=b and find phi
phi.hat=solve(R,b)
phi.hat
```

```
##            [,1]
## [1,] 0.33812448
## [2,] 0.49849991
## [3,] 0.06849712
```

```r
#sigma estimation
c0=acf(ar3.process,type = 'covariance',plot = F)$acf[1]
var.hat=c0*(1-sum(phi.hat*r))
var.hat
```

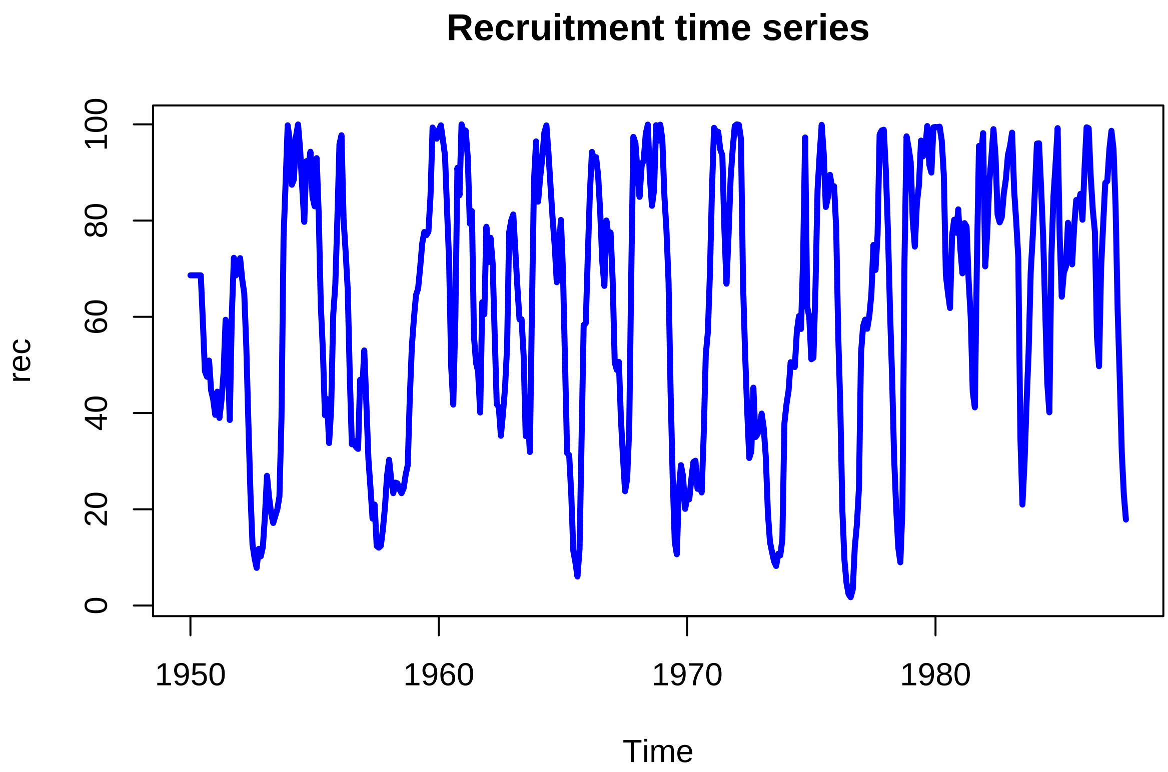
```
## [1] 15.979
```

```r
#plot
par(mfrow=c(3,1))
plot(ar3.process,main='Simulated AR(3)')
acf(ar3.process,main='ACF')
pacf(ar3.process,main='PACF')
```

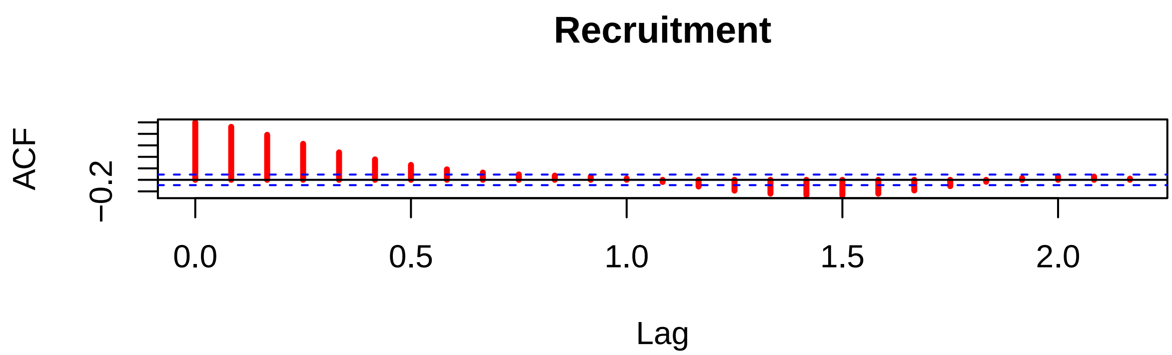


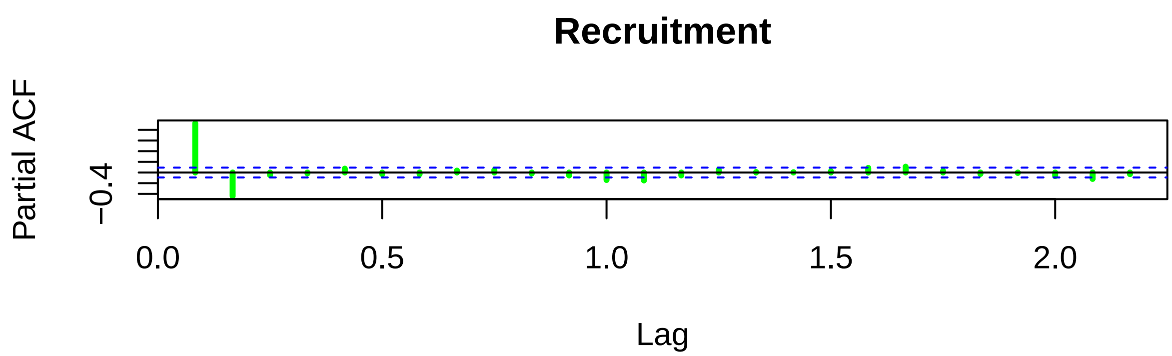
Modeling recruitment time series from 'astsa' package as an AR process

```
library(astsa)
my.data=rec

#plot rec
plot(rec,main='Recruitment time series', col='blue',lwd=3)
```




```
#subtract mean-get a time series with zero mean
ar.process=my.data-mean(my.data)

#ACF and PACF of the process
par(mfrow=c(2,1))
acf(ar.process, main='Recruitment',col='red',lwd=3)
pacf(ar.process,main='Recruitment',col='green',lwd=3)
```






```r
#order
p=2

#sample autocorrelation function r
r=NULL
r[1:p]=acf(ar.process,plot=F)$acf[2:(p+1)]
cat('r=',r,'\n')
```

```
## r= 0.9218042 0.7829182
```

```r
#matrix R
R=matrix(1,p,p)  #entries all 1-dimension 2x2

#define non-diagonal entries of R
for (i in 1:p) {
  for (j in 1:p) {
    if (i!=j)
      R[i,i]=r[abs(i-j)]
  }
}
R
```

```
##           [,1]      [,2]
## [1,] 0.9218042 1.0000000
## [2,] 1.0000000 0.9218042
```

```
#b-column vector on the right
b=NULL
b=matrix(r,p,1)
b
```

```
##           [,1]
## [1,] 0.9218042
## [2,] 0.7829182
```

```
#solve Rx=b-give x=R^(-1)b vector
phi.hat=NULL
phi.hat=solve(R,b)[,1]
phi.hat
```

```
## [1] -0.4445447  1.3315874
```

```
#variance estimation using Yule-Walker Estimator
c0=acf(ar.process,type='covariance',plot = F)$acf[1]
c0
```

```
## [1] 780.991
```

```
var.hat=c0*(1-sum(phi.hat*r))
var.hat
```

```
## [1] 286.8261
```

```
#constant term in the model
phi0.hat=mean(my.data)*(1-sum(phi.hat))
phi0.hat
```

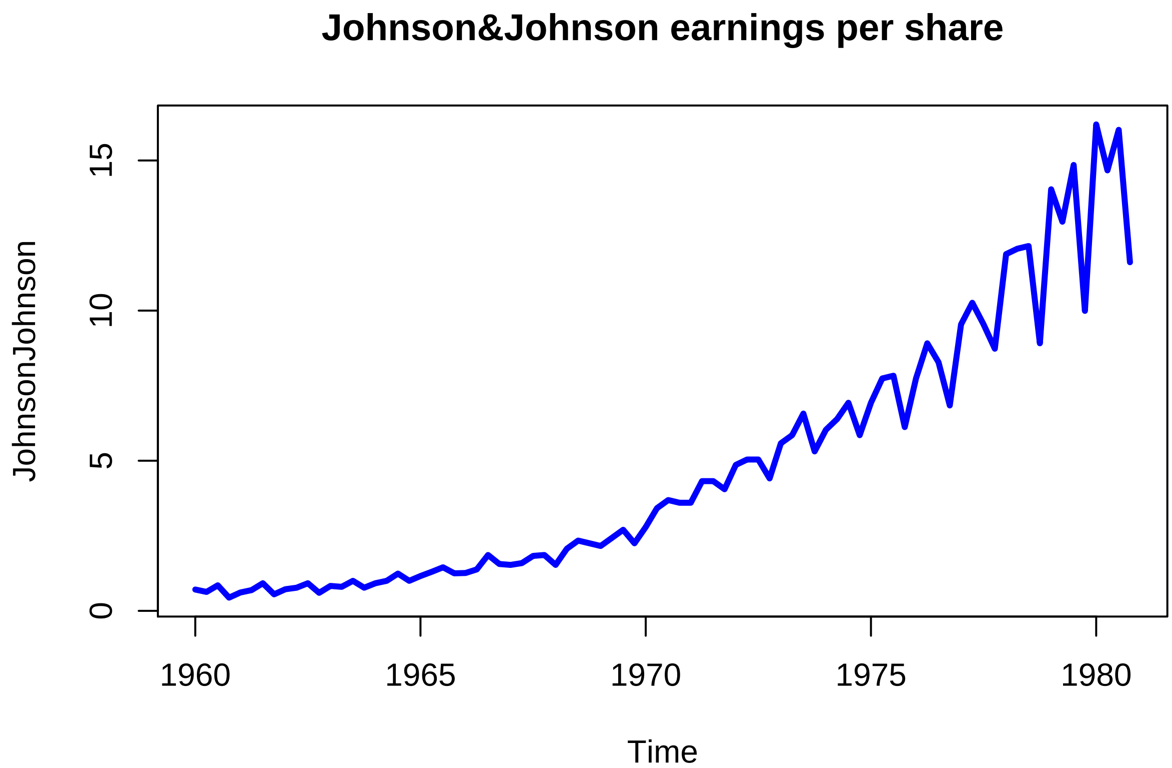
```
## [1] 7.033036
```

```
#fitted model
cat('Constant:',phi0.hat,'Coefficents:',phi.hat,'and Variance:', var.hat,'\n')
```

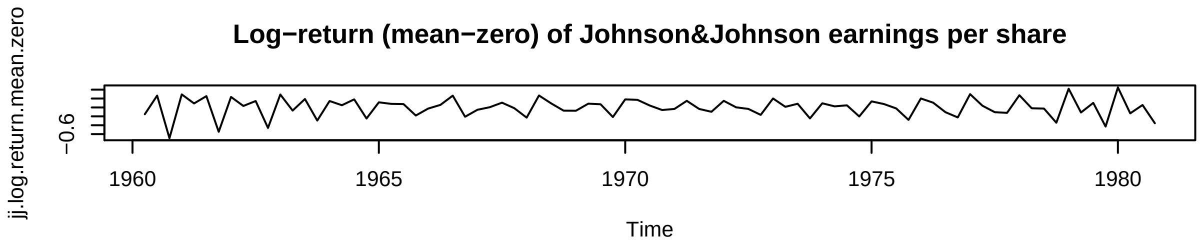
```
## Constant: 7.033036 Coefficents: -0.4445447 1.331587 and Variance: 286.8261
```

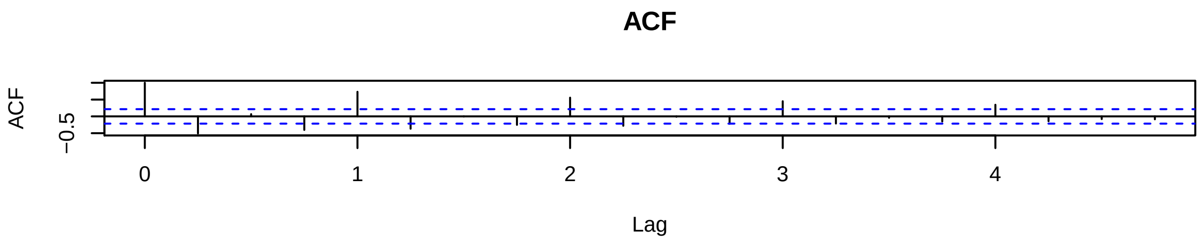
Johnson & Johnson quarterly earnings per share

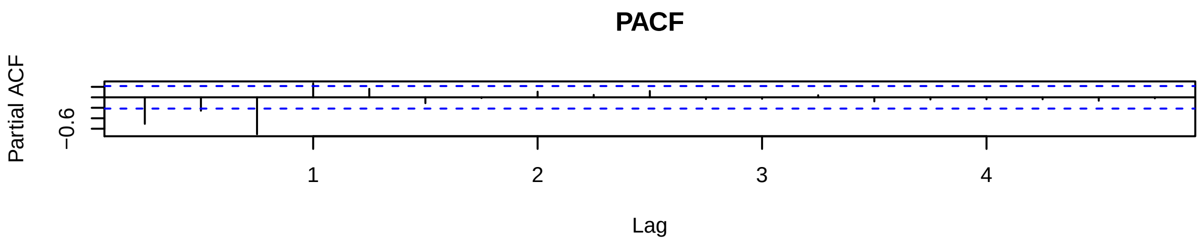
```
#Time plot for Johnson&Johnson
#trend go up-variance increase
#-systematic difference in trend,variation-not stationary dataset
#-not fit AR model to dataset
#-transform dataset
plot(JohnsonJohnson,main='Johnson&Johnson earnings per share', col='blue',lwd=3)
```




```
#log-return of Johnson&Johnson
jj.log.return=diff(log(JohnsonJohnson))
jj.log.return.mean.zero=jj.log.return-mean(jj.log.return)

#plots for log-return
par(mfrow=c(3,1))
plot(jj.log.return.mean.zero,
     main='Log-return (mean-zero) of Johnson&Johnson earnings per share')
acf(jj.log.return.mean.zero,main='ACF')
pacf(jj.log.return.mean.zero,main='PACF')
```








```
#order
p=4

#sample autocorrelation funtion r
r=NULL
r[1:p]=acf(jj.log.return.mean.zero,plot = F)$acf[2:(1+p)]
r
```

```
## [1] -0.50681760  0.06710084 -0.40283604  0.73144780
```

```
#matrix R
R=matrix(1,p,p)  #matrix-entries all 1-dimention 2x2
#define non-diagonal
for (i in 1:p) {
  for (j in 1:p) {
    if(i!=j)
      R[i,j]=r[abs(i-j)]
  }
}
R
```

```
##             [,1]        [,2]        [,3]        [,4]
## [1,]  1.00000000 -0.50681760  0.06710084 -0.40283604
## [2,] -0.50681760  1.00000000 -0.50681760  0.06710084
## [3,]  0.06710084 -0.50681760  1.00000000 -0.50681760
## [4,] -0.40283604  0.06710084 -0.50681760  1.00000000
```

```r
#b-column vector on the right
b=matrix(r,p,1)
b
```

```
##             [,1]
## [1,] -0.50681760
## [2,]  0.06710084
## [3,] -0.40283604
## [4,]  0.73144780
```

```r
# phi
phi.hat=solve(R,b)[,1]
phi.hat
```

```
## [1] -0.6293492 -0.5171526 -0.4883374  0.2651266
```

```r
#variance estimation using Yule-Walker Estimation
c0=acf(jj.log.return.mean.zero,type = 'covariance',plot = F)$acf[1]
c0
```

```
## [1] 0.04365692
```

```r
var.hat=c0*(1-sum(phi.hat*r))
var.hat
```

```
## [1] 0.01419242
```

```r
#constant term in the model
phi0.hat=mean(jj.log.return)*(1-sum(phi.hat))
phi0.hat
```

```
## [1] 0.079781
```

```r
cat('constant:',phi0.hat,'Coefficents:',phi.hat,'and Variance:',var.hat,'\n')
```

```
## constant: 0.079781 Coefficents: -0.6293492 -0.5171526 -0.4883374 0.2651266 and Variance: 0.01419242
```