R Notebook

Code

Hide

```
#Loading the data

library(quantmod)
getSymbols("EA",src="yahoo",from=as.Date("2018-02-06"),to=as.Date("2023-02-06"))
```

```
[1] "EA"
```

Hide

```
head(EA)
```

```
           EA.Open EA.High EA.Low EA.Close EA.Volume EA.Adjusted
2018-02-06  118.86  123.35 117.76   123.13   4652300    121.4598
2018-02-07  122.86  125.00 122.18   123.05   4066900    121.3809
2018-02-08  123.00  123.00 116.52   116.54   5478900    114.9592
2018-02-09  117.96  122.14 114.67   120.64   5945100    119.0036
2018-02-12  121.78  124.16 121.53   122.22   3695100    120.5622
2018-02-13  120.85  123.13 120.58   122.28   2388700    120.6214
```

Hide

```
tail(EA)
```

```
           EA.Open EA.High EA.Low EA.Close EA.Volume EA.Adjusted
2023-01-27  129.14  130.57 128.79   128.87   1786200    128.6496
2023-01-30  128.92  129.47 128.11   128.99   2446900    128.7694
2023-01-31  129.19  129.99 128.38   128.68   3067700    128.4599
2023-02-01  116.78  117.22 112.58   116.76  14492300    116.5603
2023-02-02  117.50  117.52 114.10   115.99   6355600    115.7916
2023-02-03  115.15  115.54 113.78   113.92   4393500    113.7252
```

Hide

```
getSymbols("ATVI",src="yahoo",from=as.Date("2018-02-06"),to=as.Date("2023-02-06"))
```

```
[1] "ATVI"
```

Hide

```
head(ATVI)
```

```
           ATVI.Open ATVI.High ATVI.Low ATVI.Close ATVI.Volume ATVI.Adjusted
2018-02-06     66.00     69.84    65.72      69.70    10524300      67.60927
2018-02-07     69.62     70.86    69.43      69.46     6255200      67.37649
2018-02-08     69.63     69.79    65.76      65.83    11179300      63.85537
2018-02-09     66.99     67.78    63.32      67.08    18582300      65.06787
2018-02-12     67.16     69.20    67.16      68.32     8315700      66.27067
2018-02-13     67.96     68.21    67.18      68.03     5373600      65.98937
```

Hide

```
tail(ATVI)
```

```
           ATVI.Open ATVI.High ATVI.Low ATVI.Close ATVI.Volume ATVI.Adjusted
2023-01-27     75.50     76.76    75.22      76.61     4382700         76.61
2023-01-30     76.63     77.08    75.84      75.96     4247400         75.96
2023-01-31     76.13     77.00    75.85      76.57     4118000         76.57
2023-02-01     76.00     76.82    75.58      76.70     4575400         76.70
2023-02-02     76.50     77.39    76.07      77.11     4696100         77.11
2023-02-03     76.64     76.78    75.03      75.24     5781000         75.24
```

Hide

```
#EA - Electronic Arts
#ATVI - Activision Blizzard

#EA Log Return Calculation
EA_log_return_1<-diff(log(EA[,6]))
EA_log_return<-as.numeric(EA_log_return_1[-1])
head(EA_log_return)
```

```
[1] -0.0006498993 -0.0543561108  0.0345762877  0.0130117506  0.0004909456
[6]  0.0121113083
```

Hide

```
#Mean and sd EA Log Return Calculation
mean_EA<-mean(EA_log_return)
mean_EA
```

```
[1] -5.234592e-05
```

Hide

```
sd_EA<- sd(EA_log_return)
sd_EA
```

```
[1] 0.01994566
```

Hide

```
#EA Log Return Calculation
ATVI_log_return_1<-diff(log(ATVI[,6]))
ATVI_log_return<-as.numeric(ATVI_log_return_1[-1])
head(ATVI_log_return)
```

```
[1] -0.003448960 -0.053675432  0.018810212  0.018316492 -0.004253687
[6]  0.023533879
```

Hide

```
#Mean and sd EA Log Return Calculation
mean_ATVI<-mean(ATVI_log_return)
mean_ATVI
```

```
[1] 8.507391e-05
```

Hide

```
sd_ATVI<- sd(ATVI_log_return)
sd_ATVI
```

```
[1] 0.02164873
```

Hide

```
library(mnormt)
library(MASS)


df <- seq(2.25, 6, 0.01)
n <- length(df)
loglik <- rep(0, n)

dat <- cbind(ATVI_log_return, EA_log_return)

for(i in 1:n) {
  fit <- cov.trob(dat, nu = df[i])
  loglik[i] <- sum(log(dmt(dat, mean = fit$center, S = fit$cov, df = df[i])))
}

aic_t <- -max(2 * loglik) + 2 * (8 + 10 + 1) + 64000
z1 <- (2 * loglik > 2 * max(loglik) - qchisq(0.95, 1))

# best degree of freedom
best_index <- which.max(loglik)
best_df <- df[best_index]
best_df
```

```
[1] 3.49
```

Hide

```
# best fit using the best df
bestfit <- cov.trob(dat,nu=best_df,cor=TRUE)
bestfit
```

```
$cov
                ATVI_log_return EA_log_return
ATVI_log_return    0.0001958947  0.0001258700
EA_log_return      0.0001258700  0.0001845842

$center
ATVI_log_return   EA_log_return
   0.0007788793    0.0006633164

$n.obs
[1] 1257

$cor
                ATVI_log_return EA_log_return
ATVI_log_return       1.0000000     0.6619324
EA_log_return         0.6619324     1.0000000

$call
cov.trob(x = dat, cor = TRUE, nu = best_df)

$iter
[1] 4
```

Hide

```
#Kendall
ρτ<-cor(ATVI_log_return, EA_log_return, method = "kendall") # kendall
ρτ
```

```
[1] 0.4676686
```

Hide

```
#Pearson
ρ<-cor(ATVI_log_return, EA_log_return, method = "pearson") #pearson
ρ
```

```
[1] 0.5981103
```

Hide

```
# Pearson estimation based on Kendall
omega<-sin(ρτ*pi/2)
omega
```

```
[1] 0.6702994
```

Hide

```
# Fit t-distribution to ATVI log-returns
fit_ATVI <- fitdistr(ATVI_log_return, "t")
```

```
Warning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarni
ng: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: N
aNs produced
```

Hide

```
cat("ATVI log-return:\n")
```

```
ATVI log-return:
```

Hide

```
cat("Mean:", fit_ATVI$estimate[1], "\n")
```

```
Mean: 0.0007499526
```

Hide

```
cat("Scale parameter:", fit_ATVI$estimate[2], "\n")
```

```
Scale parameter: 0.01323661
```

Hide

```
cat("Degrees of freedom:", fit_ATVI$estimate[3], "\n\n")
```

```
Degrees of freedom: 3.052435
```

Hide

```
# Fit t-distribution to EA log-returns
fit_EA <- fitdistr(EA_log_return, "t")
```

```
Warning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarni
ng: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: NaNs producedWarning: N
aNs producedWarning: NaNs producedWarning: NaNs produced
```

Hide

```
cat("EA log-return:\n")
```

```
EA log-return:
```

Hide

```
cat("Mean:", fit_EA$estimate[1], "\n")
```

```
Mean: 0.00048499
```

Hide

```
cat("Scale parameter:", fit_EA$estimate[2], "\n")
```

```
Scale parameter: 0.01406657
```

Hide

```
cat("Degrees of freedom:", fit_EA$estimate[3], "\n")
```

```
Degrees of freedom: 4.018493
```

Hide

```
# Now convert estimated scale parameters to estimated standard deviations

cat("Standard deviation:", fit_ATVI$estimate[2] * sqrt((fit_ATVI$estimate[3] )/(fit_ATVI$esti
mate[3]-2)), "\n")
```

```
Standard deviation: 0.02254251
```

Hide

```
cat("Standard deviation:", fit_EA$estimate[2] * sqrt((fit_EA$estimate[3])/(fit_EA$estimate[3]
-2)), "\n")
```

```
Standard deviation: 0.01984752
```

Hide

```
library(copula)
library(fGarch)

# ATVI data percentiles
ATVI_data<-pstd(ATVI_log_return,fit_ATVI$estimate[1], fit_ATVI$estimate[2] * sqrt((fit_ATVI$e
stimate[3] )/(fit_ATVI$estimate[3]-2)), fit_ATVI$estimate[3])

# EA data percentiles
EA_data<-pstd(EA_log_return,fit_EA$estimate[1], fit_EA$estimate[2] * sqrt((fit_EA$estimate[3]
)/(fit_EA$estimate[3]-2)), fit_EA$estimate[3])

#fit the copulas to the uniform -transformed data

data1<- cbind(ATVI_data, EA_data)
```

Hide

```
#t copula
# t-copula values
cop_t_dim2<-tCopula(omega, dim = 2, dispstr = "un", df=best_df)
cop_t_dim2
```

```
t-copula, dim. d = 2
Dimension:  2
Parameters:
  rho.1   = 0.6702994
  df      = 3.4900000
```

Hide

```
ft<- fitCopula(cop_t_dim2, data1, method="ml", start=c(omega, best_df) )
summary(ft)
```

```
Call: fitCopula(cop_t_dim2, data = data1, ... = pairlist(method = "ml", start = c(omega,
    best_df)))
Fit based on "maximum likelihood" and 1257 2-dimensional observations.
t-copula, dim. d = 2
      Estimate Std. Error
rho.1   0.6687      0.016
df      4.5859      0.703
The maximized loglikelihood is 384
Optimization converged
Number of loglikelihood evaluations:
function gradient
      19        6
```

Hide

```
#Gaussian copula
fnorm<- fitCopula(copula = normalCopula(dim=2), data=data1, method="ml")
summary(fnorm)
```

```
Call: fitCopula(normalCopula(dim = 2), data = data1, ... = pairlist(method = "ml"))
Fit based on "maximum likelihood" and 1257 2-dimensional observations.
Normal copula, dim. d = 2
      Estimate Std. Error
rho.1   0.6497      0.014
The maximized loglikelihood is 344
Optimization converged
Number of loglikelihood evaluations:
function gradient
       8        8
```

Hide

```
#Clayton copula
fclayton<- fitCopula(copula = claytonCopula(1,dim=2), data=data1, method="ml")
summary(fclayton)
```

```
Call: fitCopula(claytonCopula(1, dim = 2), data = data1, ... = pairlist(method = "ml"))
Fit based on "maximum likelihood" and 1257 2-dimensional observations.
Clayton copula, dim. d = 2
      Estimate Std. Error
alpha    1.757      0.072
The maximized loglikelihood is 233.3
Optimization converged
Number of loglikelihood evaluations:
function gradient
       3        3
```

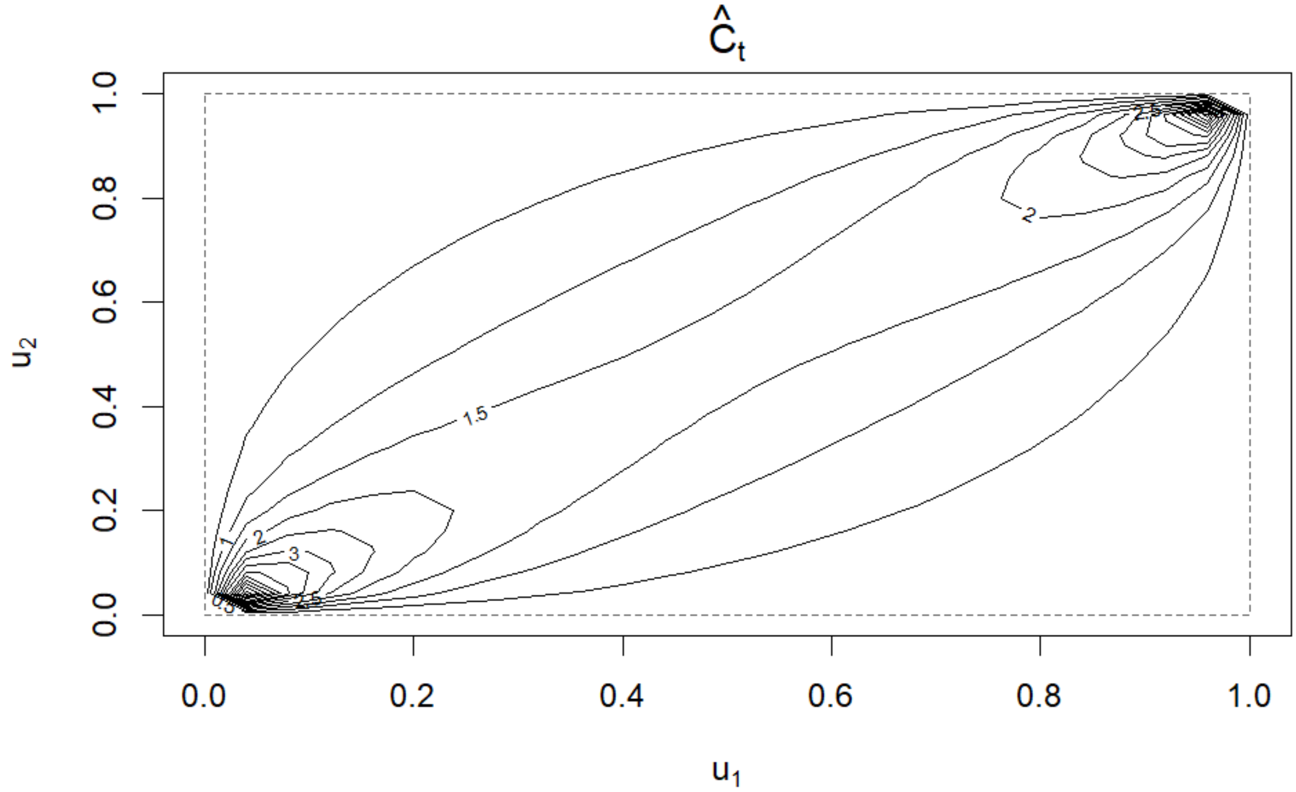
Hide

```
#Joe copula
fjoe<- fitCopula(copula = joeCopula(2,dim=2), data=data1, method="ml")
summary(fjoe)
```

```
Call: fitCopula(joeCopula(2, dim = 2), data = data1, ... = pairlist(method = "ml"))
Fit based on "maximum likelihood" and 1257 2-dimensional observations.
Joe copula, dim. d = 2
      Estimate Std. Error
alpha     2.09       0.06
The maximized loglikelihood is 276.8
Optimization converged
Number of loglikelihood evaluations:
function gradient
       7        7
```

Hide

```
# 1- use contour and dcopula

#tCopula
contour(tCopula(param=0.6702994, dim=2, df=round(best_df)), dCopula,main=expression(hat(C)
[t]))
```

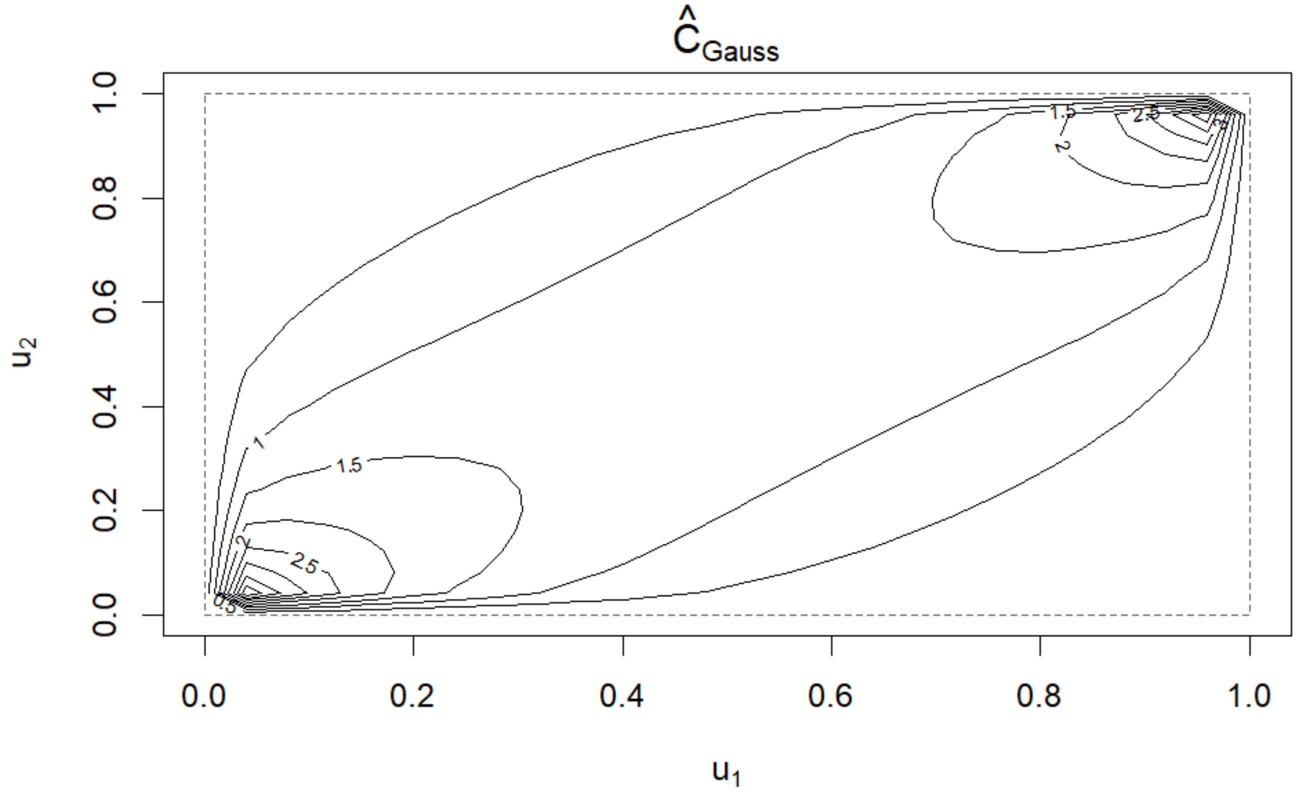


Hide

```
# The t-copula contour has both upper and lower tail dependence, which normally can be contro
lled by its df and correlation parameter. In the generated contour plot, it is possible to ob
serve a higher concentration of contours in both the lower left and upper right corners when
tail dependence is present. The lower the df, the stronger the tail dependence, meaning that
they tend to have extreme values simultaneously either positive or negative
```

Hide

```
#NormalCopula
contour(normalCopula(param=0.6497, dim=2), dCopula, main=expression(hat(C)[Gauss]))
```

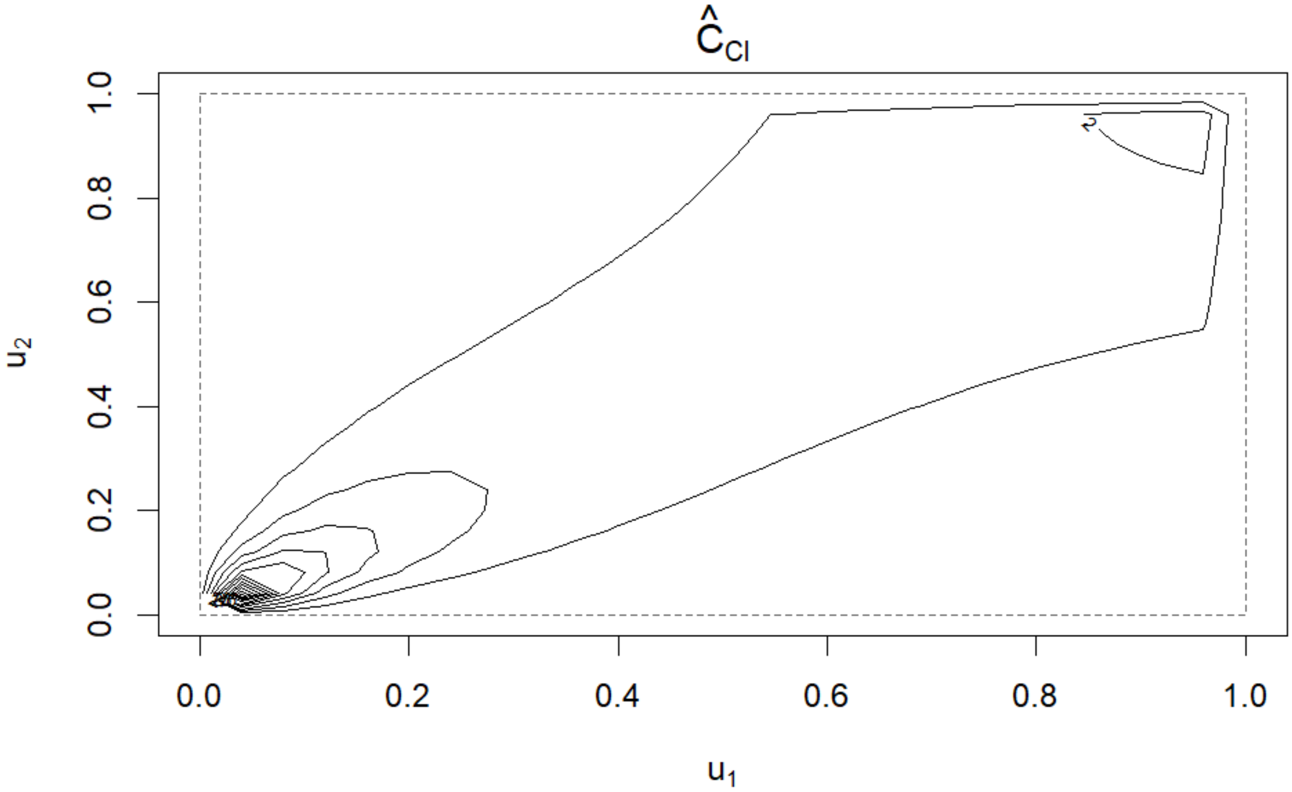


Hide

```
# The proposed Gaussian contour copula does not exhibit tail dependence in either the upper o
r lower tails. In the contour plot, the contours will appear symmetric and elliptical, withou
t higher concentrations in the corners. As a result, extreme events in one asset don't necess
arily correspond to extreme events in the other asset
```

Hide

```
#claytonCopula
contour(claytonCopula(param=1.757, dim=2), dCopula,main=expression(hat(C)[Cl]))
```

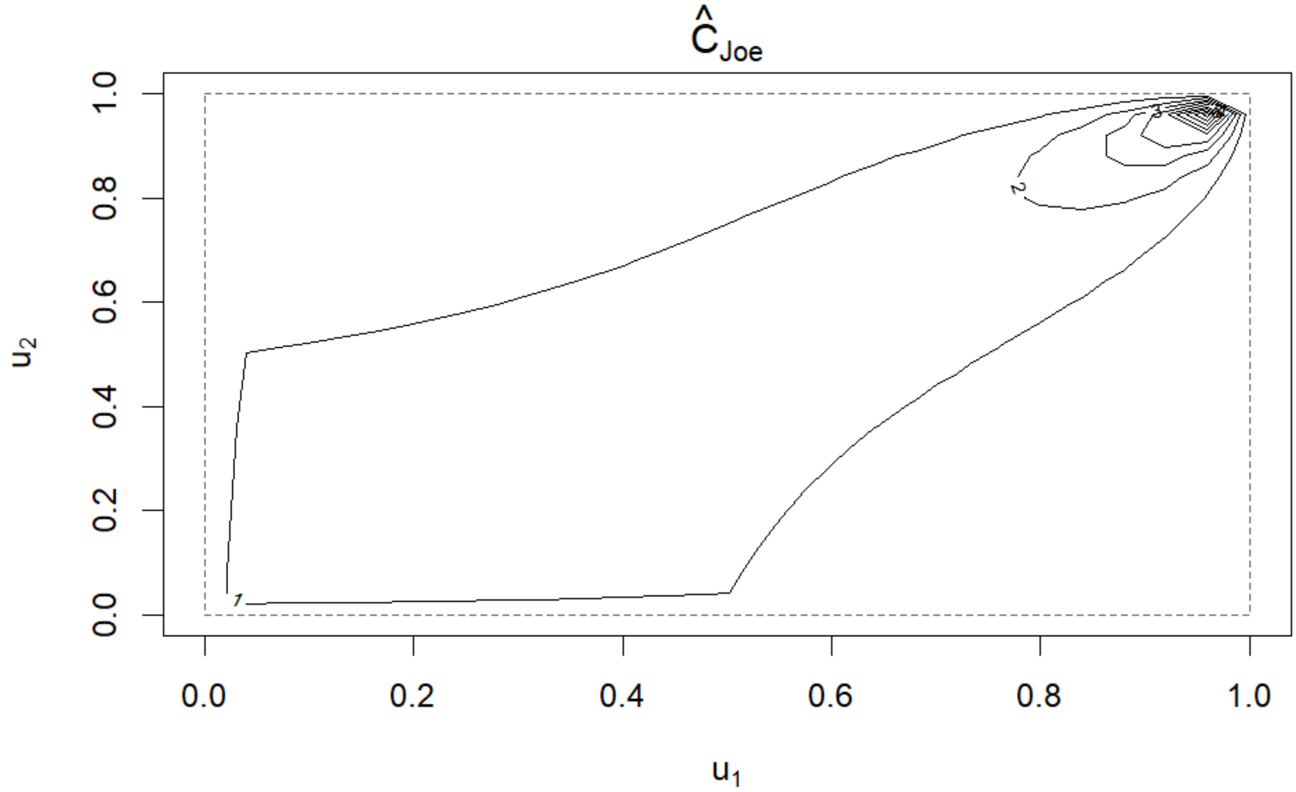


Hide

```
# The Clayton copula exhibits lower tail dependence, but not upper tail dependence. In the co
ntour plot, it is possible to observe a higher concentration of contours in the lower left co
rner, indicating that the dependence is stronger in the lower tail, but not in the upper righ
t, so absence of upper tail dependence, meaning that they tend to have extreme negative value
s simultaneously
```

Hide

```
#joeCopula
contour(joeCopula(param=2.09, dim=2), dCopula,main=expression(hat(C)[Joe]))
```

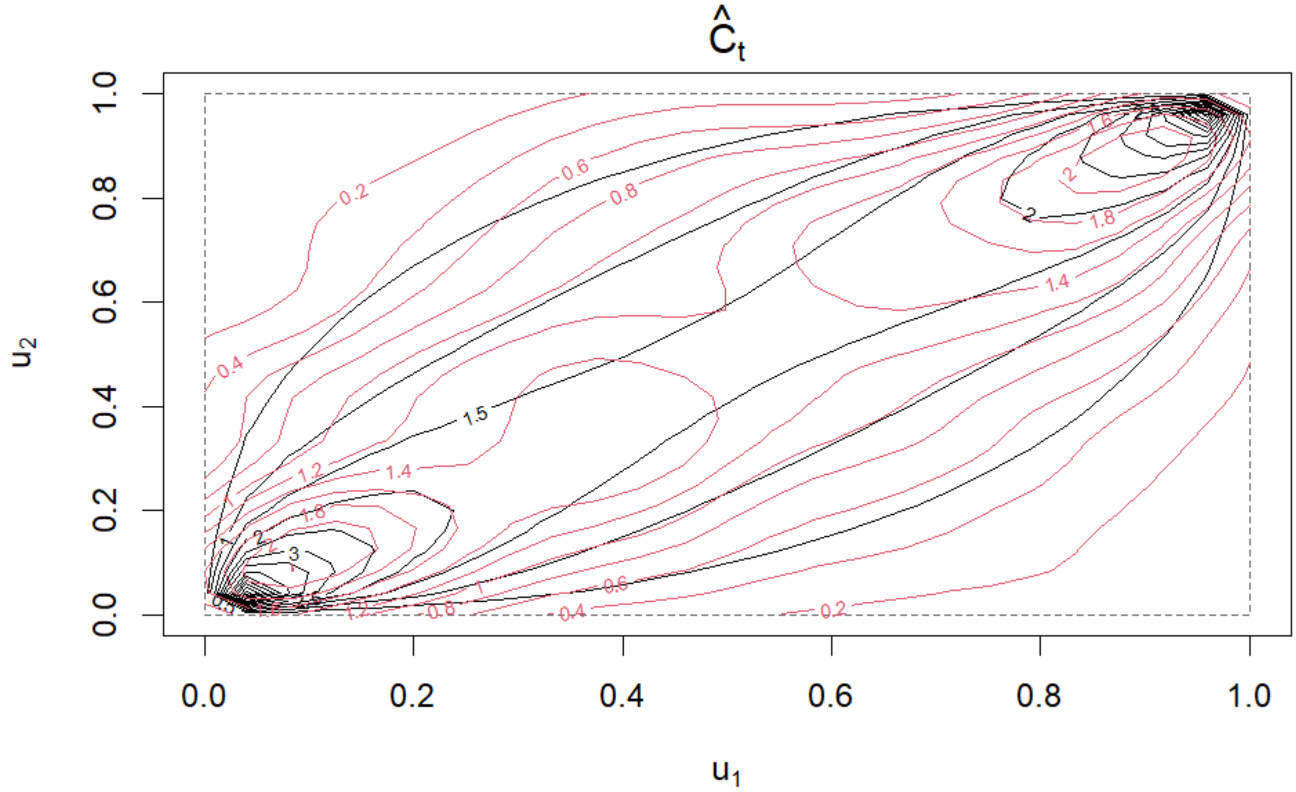


Hide

```
# The joe copula exhibits upper tail dependence, but not lower tail dependence. In the contou
r plot, it is possible to observe a higher concentration of contours in the upper right corne
r, indicating that the dependence is stronger in the upper tail, but not in the lower left, s
o absence of lower tail dependence, meaning that they tend to have extreme positive values si
multaneously.
```

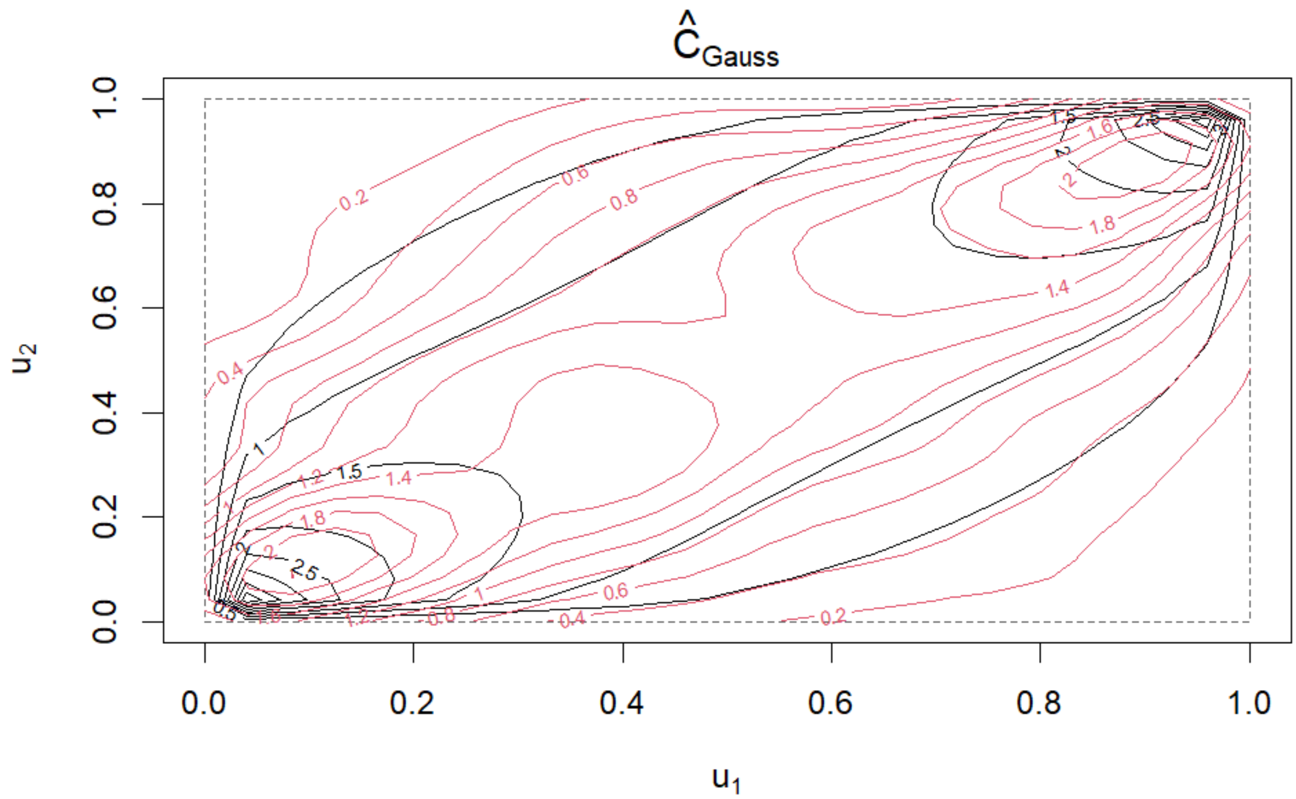
Hide

```
#2- Use KDE to superimpose the kernel density estimate for the percentiles for the bivariate
log-returns

#tCopula

contour(tCopula(param=0.6702994, dim=2, df=round(best_df)), dCopula,main=expression(hat(C)
[t]))
contour(kde2d(data1[,1], data1[,2]), col=2, add=TRUE)
```



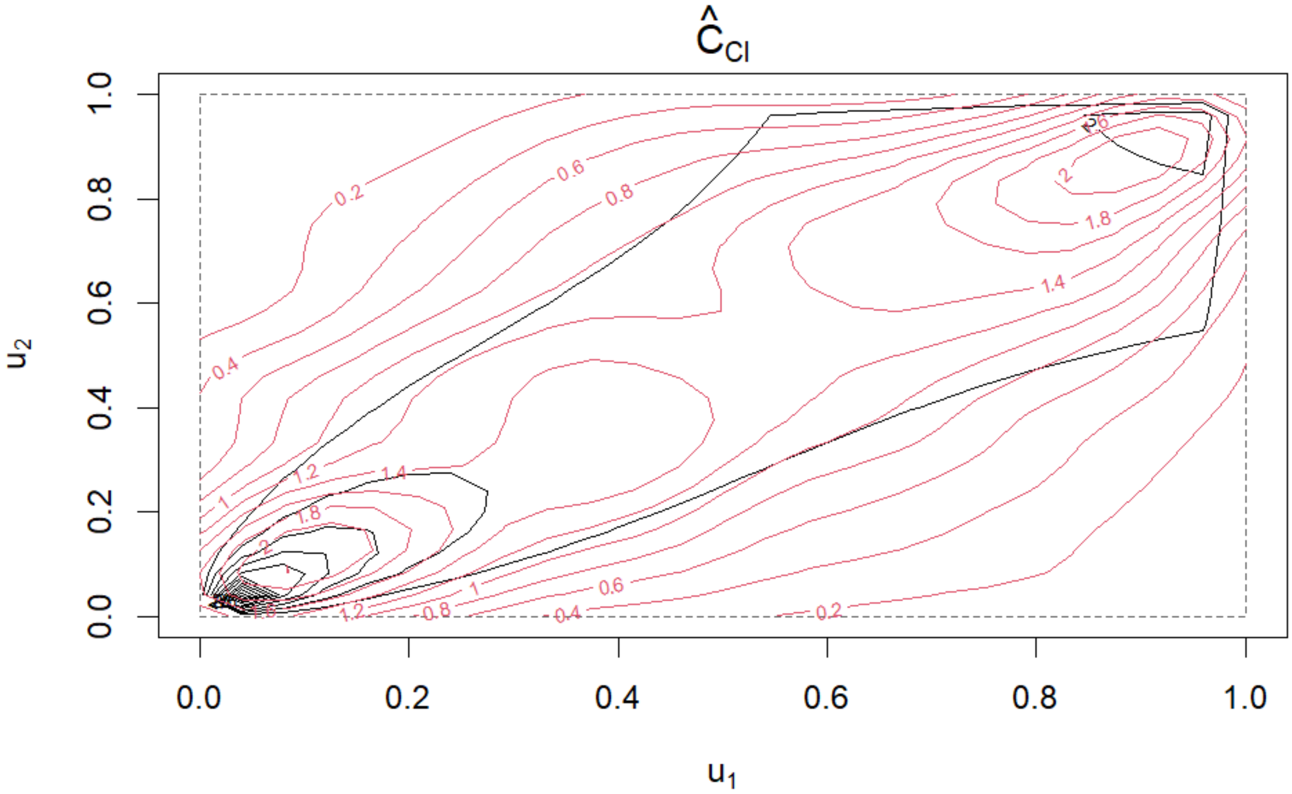
Hide

```
#NormalCopula

contour(normalCopula(param=0.6497, dim=2), dCopula, main=expression(hat(C)[Gauss]))
contour(kde2d(data1[,1], data1[,2]), col=2, add=TRUE)
```



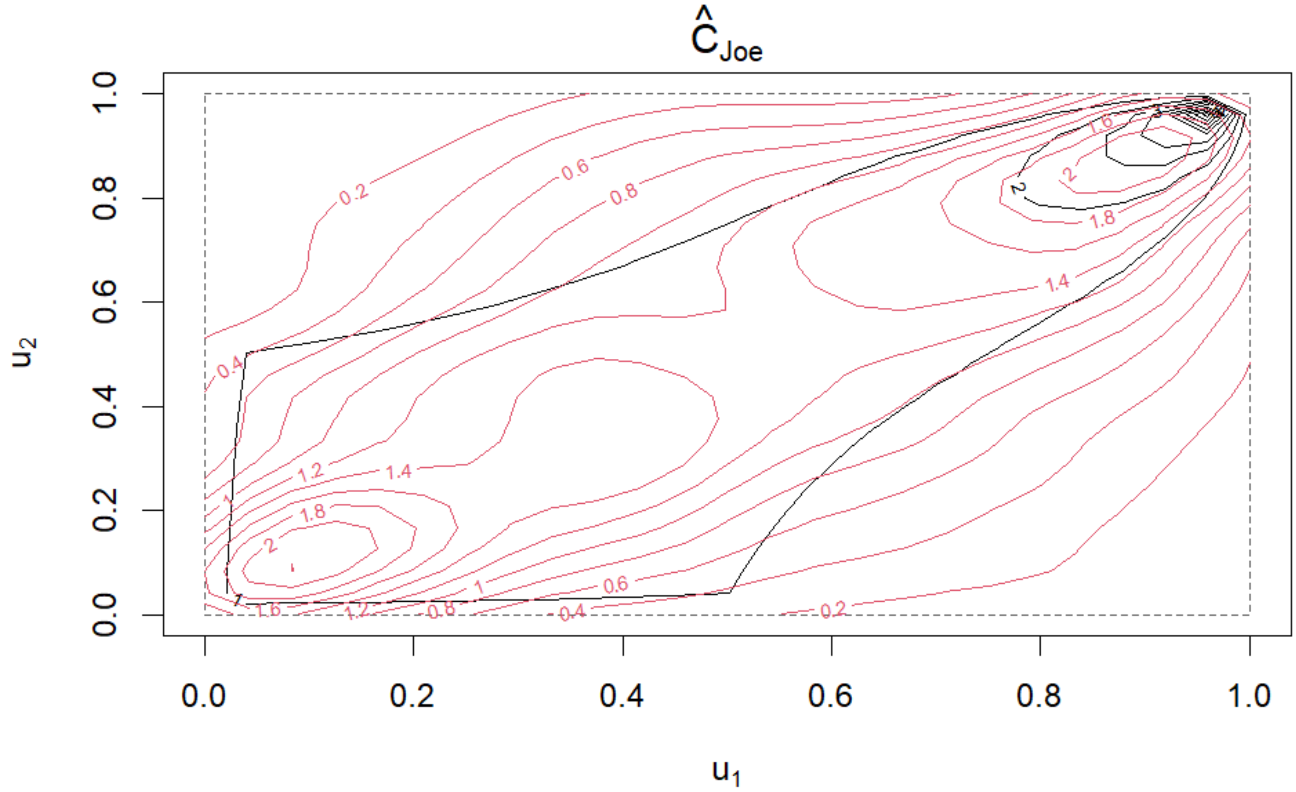
Hide

```
#claytonCopula

contour(claytonCopula(param=1.757, dim=2), dCopula,main=expression(hat(C)[Cl]))
contour(kde2d(data1[,1], data1[,2]), col=2, add=TRUE)
```



Hide

```
#joeCopula

contour(joeCopula(param=2.09, dim=2), dCopula,main=expression(hat(C)[Joe]))
contour(kde2d(data1[,1], data1[,2]), col=2, add=TRUE)
```



Hide

```
#  The contour plots show the estimated joint density of the two assets. The contour lines in
dicate regions of high and low density, with denser regions indicating a stronger dependence
between the two assets. From the shape of the contour lines we can see insights of the type o
f dependence, such as whether it is symmetric or asymmetric, linear or nonlinear.
```

Hide

```
#3- find the estimate parameter that minimazes the variance of αX + (1 − α)Y, where X and Y a
re log returns.
# calculate the covariance of ATVI and EA
df <- data.frame(ATVI_log_return, EA_log_return)
cov.ATVI.EA <- cov(ATVI_log_return, EA_log_return)
cat("The covariance of ATVI and EA is:", cov.ATVI.EA, "\n")
```

```
The covariance of ATVI and EA is: 0.0002582628
```

Hide

```
# calculate the variance of ATVI
var.ATVI <- var(ATVI_log_return)
cat("The variance of ATVI is:", var.ATVI, "\n")
```

```
The variance of ATVI is: 0.0004686673
```

Hide

```
# calculate the variance of EA
var.EA <- var(EA_log_return)
cat("The variance of EA is:", var.EA, "\n")
```

```
The variance of EA is: 0.0003978292
```

Hide

```
alpha.theoretical<- (var.EA-cov.ATVI.EA)/(var.ATVI+var.EA-2*cov.ATVI.EA)
cat("The alpha for the minimum portfolio value is:", alpha.theoretical, "\n")
```

```
The alpha for the minimum portfolio value is: 0.3987943
```

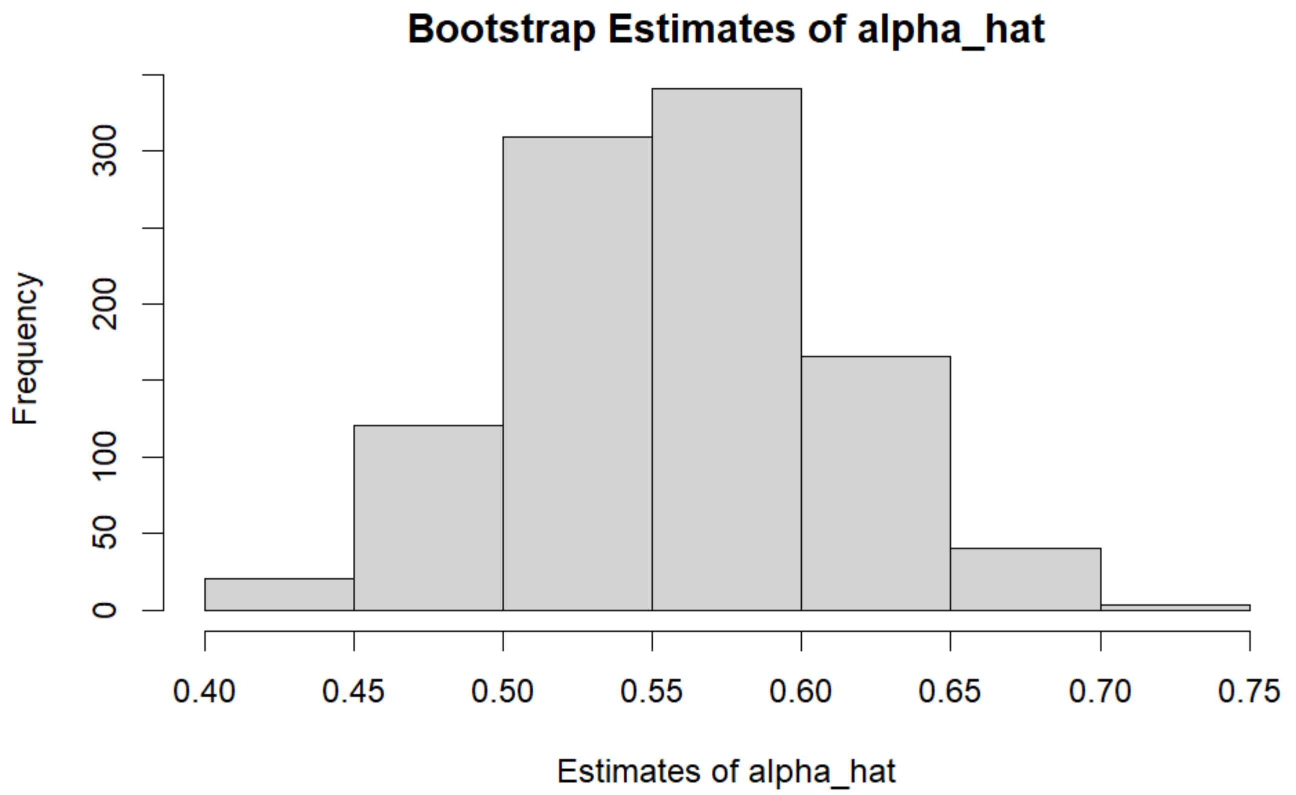
Hide

```
# Bootstrap Analysis

S <- data.frame(ATVI_log_return, EA_log_return)

# define the number of bootstrap iterations
N <- 1000

# initialize a vector to store the coefficient estimates from each iteration
alpha_hat_boot <- numeric(N)

for (i in 1:N) {
  # generate a bootstrap sample by selecting N pairs with replacement
  bootstrap_sample <- S[sample(nrow(S), replace = TRUE),]

  # calculate the coefficient estimate using the same method as for ˆα
  alpha_hat_boot[i] <- lm(EA_log_return ~ ATVI_log_return, data = bootstrap_sample)$coef[2]
}

# calculate the sample variance of the saved coefficient estimates
var_alpha_hat_boot <- var(alpha_hat_boot)
cat("The bootstrap estimate of the var of ˆα is:", var_alpha_hat_boot, "\n")
```

```
The bootstrap estimate of the var of ˆα is: 0.002777311
```

Hide

```
# Histogram of the 1000 estimates for the parameter
hist(alpha_hat_boot, main = "Bootstrap Estimates of alpha_hat",
     xlab = "Estimates of alpha_hat", ylab = "Frequency")
```



Hide

```
# Looking at the theoretical and the estimate values from the bootstrap analysis we can try t
o decide how good our theoretical value for alpha is, but in the end we can see that 0.55 is
the most common value for the alpha estimate. To better decide this outcome we could apply a
confidence interval to see if the theoretical value is within the reasonable range for a good
fit as we have a variance of  about 0.003 for the values.
```

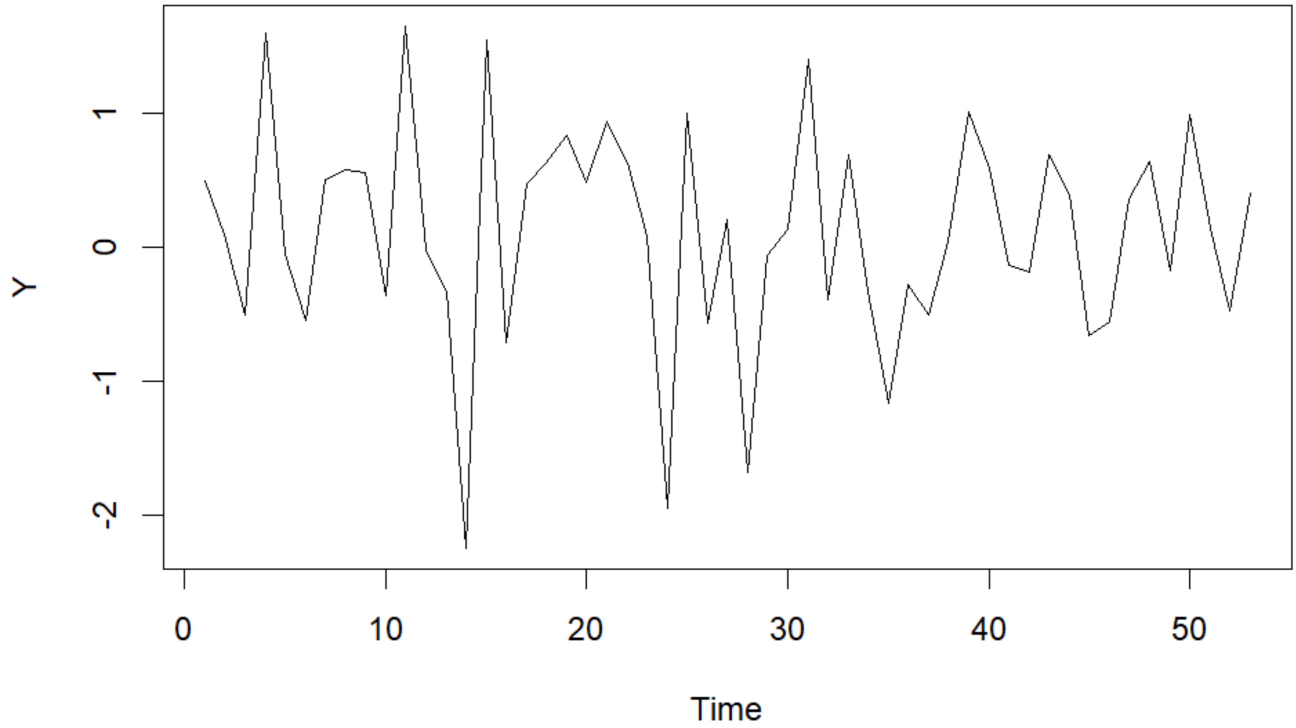
Hide

```
#4a) Set parameters in use
phi1 <- -0.5
phi2 <- 0.1
theta1 <- 0.3
n <- 50  # process or series length
burn_in <- 3  # length period

# Generate a white noise series
set.seed(1)
white_noise <- rnorm(n + burn_in)

# Set initial values
Y <- rep(0, n)
Y[1] <- 0.5
Y[2] <- 0.1
Y[3] <- -0.5

# Generate series using ARMA(2,1) relationship
for (t in (burn_in + 1):(n + burn_in)) {
  Y[t] = phi1*Y[t-1] + phi2*Y[t-2] + white_noise[t] + theta1*white_noise[t-1]
}

#List of the series
Y
```

```
 [1]  0.50000000  0.10000000 -0.50000000  1.60459222 -0.04420410 -0.53905478
 [7]  0.50639552  0.57745018  0.55919322 -0.35450558  1.65333676 -0.01874135
[13] -0.32958326 -2.23815457  1.54663991 -0.70458975  0.47728852  0.62987590
[19]  0.83716296  0.48467379  0.93852717  0.63703331  0.08454194 -1.94554984
[25]  1.00424935 -0.56686068  0.21122114 -1.67978768 -0.05835982  0.13569769
[31]  1.41037720 -0.38680269  0.69127436 -0.32182101 -1.16316313 -0.27871297
[37] -0.49574815  0.04240240  1.01145534  0.59169593 -0.13027330 -0.17841251
[43]  0.69713380  0.39934406 -0.65171539 -0.54832977  0.36132676  0.64241116
[49] -0.16685924  0.99507460  0.14821498 -0.46719466  0.40593060
```

Hide

```
#Plot
plot(Y, type="l", xlab="Time", ylab="Y")
```



Hide

```
#4b) Apply arima.sim to the same white nose series

#Set new parameters
# Set ARMA parameters and white noise series
phi <- c(phi1, phi2)
theta <- theta1
innov <- white_noise
n.start <- burn_in
start.innov <- c(0,0,0)

# Simulate time series using arima.sim
set.seed(1)
Y_sim <- arima.sim(list(order=c(2,0,1), ar=phi, ma=c(.3)), n=n, innov=innov, n.start=n.start,
start.innov=start.innov)

# time series values after burn-in period
Y_sim
```

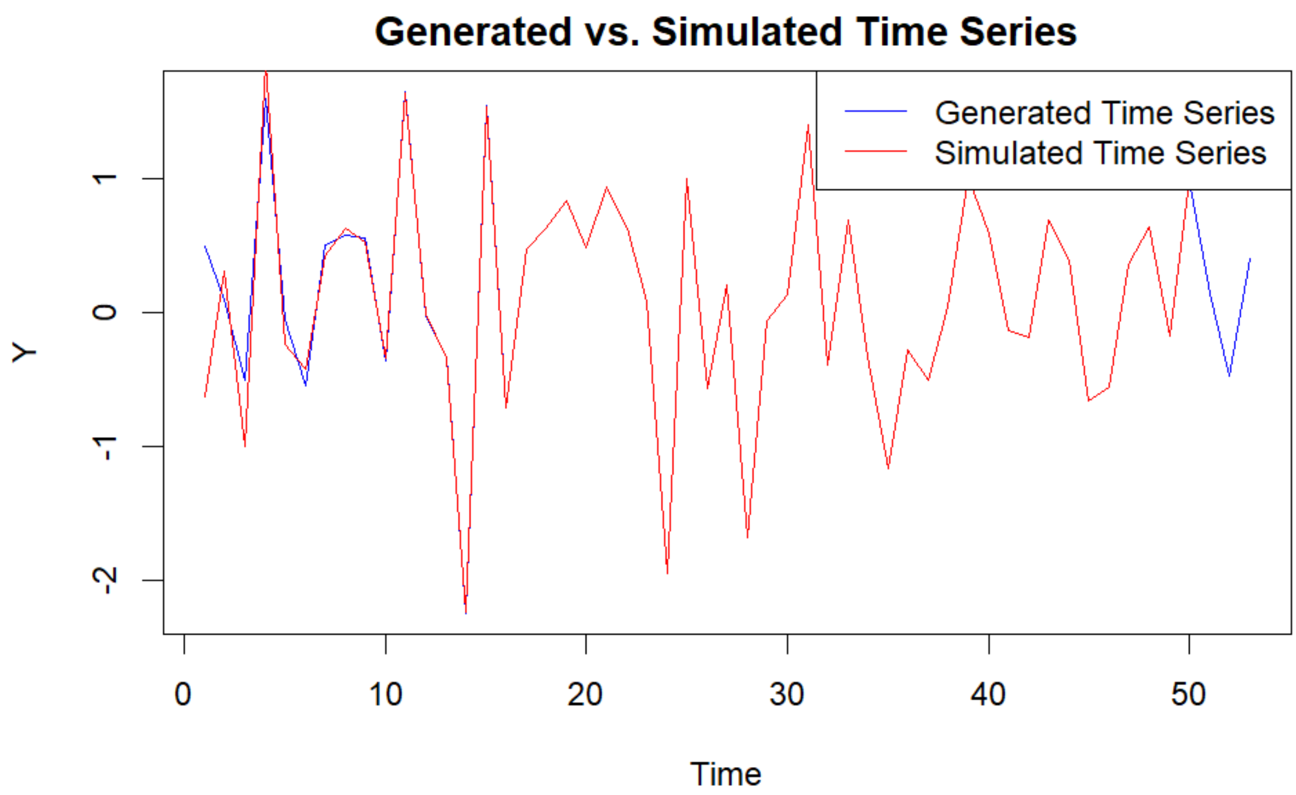
```
Time Series:
Start = 1
End = 50
Frequency = 1
 [1] -0.62645381  0.30893409 -0.99764804  1.87430965 -0.22882761 -0.41977128
 [7]  0.42829142  0.62843058  0.52589261 -0.33275723  1.63913253 -0.00946440
[13] -0.33564216 -2.23419742  1.54405545 -0.70290180  0.47618610  0.63059590
[19]  0.83669272  0.48498091  0.93832658  0.63716431  0.08445638 -1.94549396
[25]  1.00421286 -0.56683684  0.21120558 -1.67977751 -0.05836646  0.13570202
[31]  1.41037436 -0.38680084  0.69127315 -0.32182022 -1.16316365 -0.27871263
[37] -0.49574837  0.04240254  1.01145525  0.59169599 -0.13027334 -0.17841249
[43]  0.69713378  0.39934407 -0.65171539 -0.54832976  0.36132676  0.64241116
[49] -0.16685924  0.99507460
```

Hide

```
plot(Y, type="l", col="blue", xlab="Time", ylab="Y", main="Generated vs. Simulated Time Serie
s")
lines(Y_sim, type="l", col="red")
```

Hide

```
legend("topright", legend=c("Generated Time Series", "Simulated Time Series"), col=c("blue",
"red"), lty=1)
```



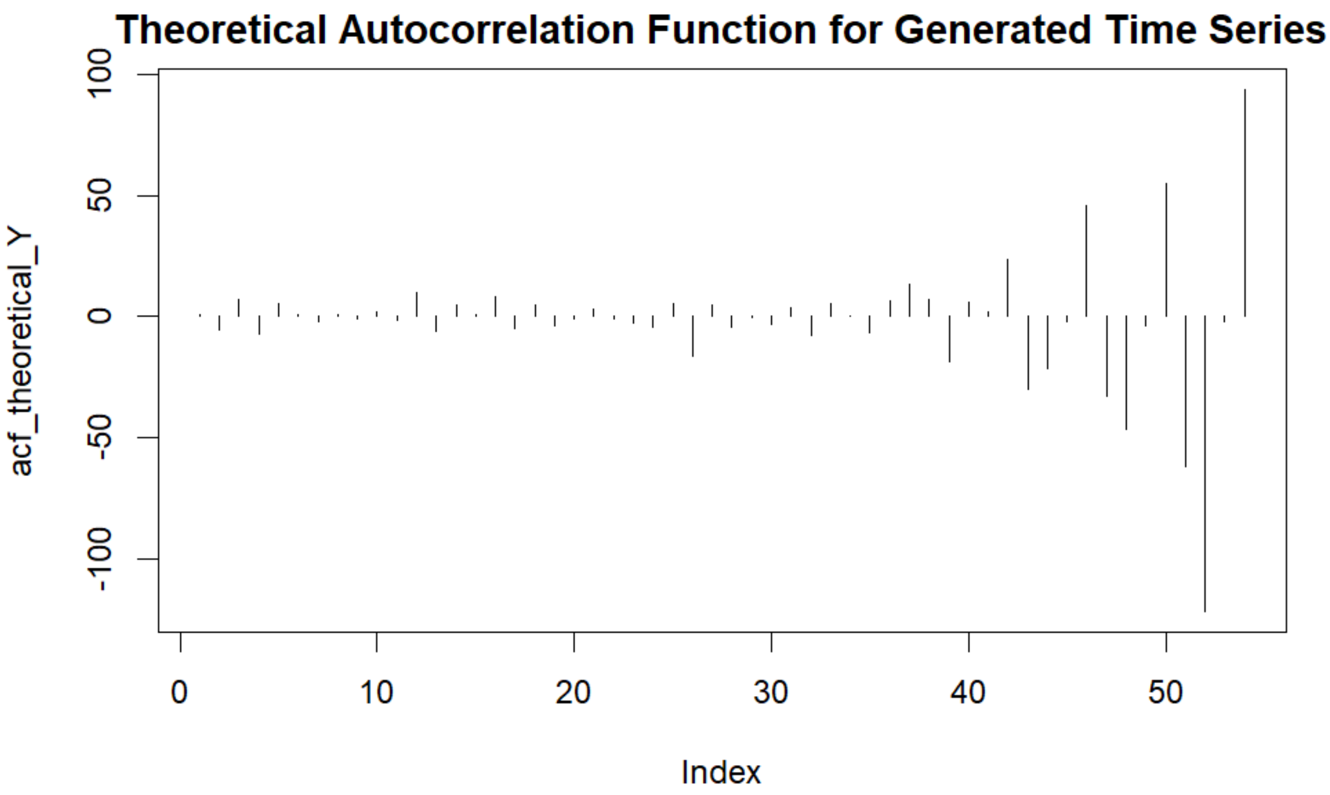
Hide

```
#4c) Obtain theoretical ACF for ARMA(2,1) model
acf_theoretical_Y <- ARMAacf(Y)
acf_theoretical_Y
```

```
          0           1           2           3           4           5 
  1.0000000  -5.2856849   6.9755937  -7.2682486   5.4566724   0.6603926 
          6           7           8           9          10          11 
 -2.1359475   0.9725904  -1.1723598   2.0826342  -1.3219214   9.7370861 
         12          13          14          15          16          17 
 -6.1957709   4.7430537   0.9356696   8.3614270  -4.8292057   4.7518839 
         18          19          20          21          22          23 
 -3.8846363  -0.6864472   2.9193329  -0.8070636  -2.3906402  -4.1508623 
         24          25          26          27          28          29 
  5.5185323 -16.0901814   4.7767261  -4.5492356  -0.2898370  -3.1328688 
         30          31          32          33          34          35 
  3.5992104  -7.7970225   5.0770951   0.3212728  -6.5117887   6.6432868 
         36          37          38          39          40          41 
 13.3183433   7.1514577 -18.7743612   5.7979365   1.9118380  23.5482312 
         42          43          44          45          46          47 
-29.9211011 -21.4228378  -2.0701902  45.7012404 -33.1441743 -46.6737315 
         48          49          50          51          52          53 
 -4.0669878  54.7484296 -61.8036990 -121.5260377  -2.2665024  93.6404363 
```

Hide

```
# Plot theoretical ACF
plot(acf_theoretical_Y, type='h', main='Theoretical Autocorrelation Function for Generated Ti
me Series')
```



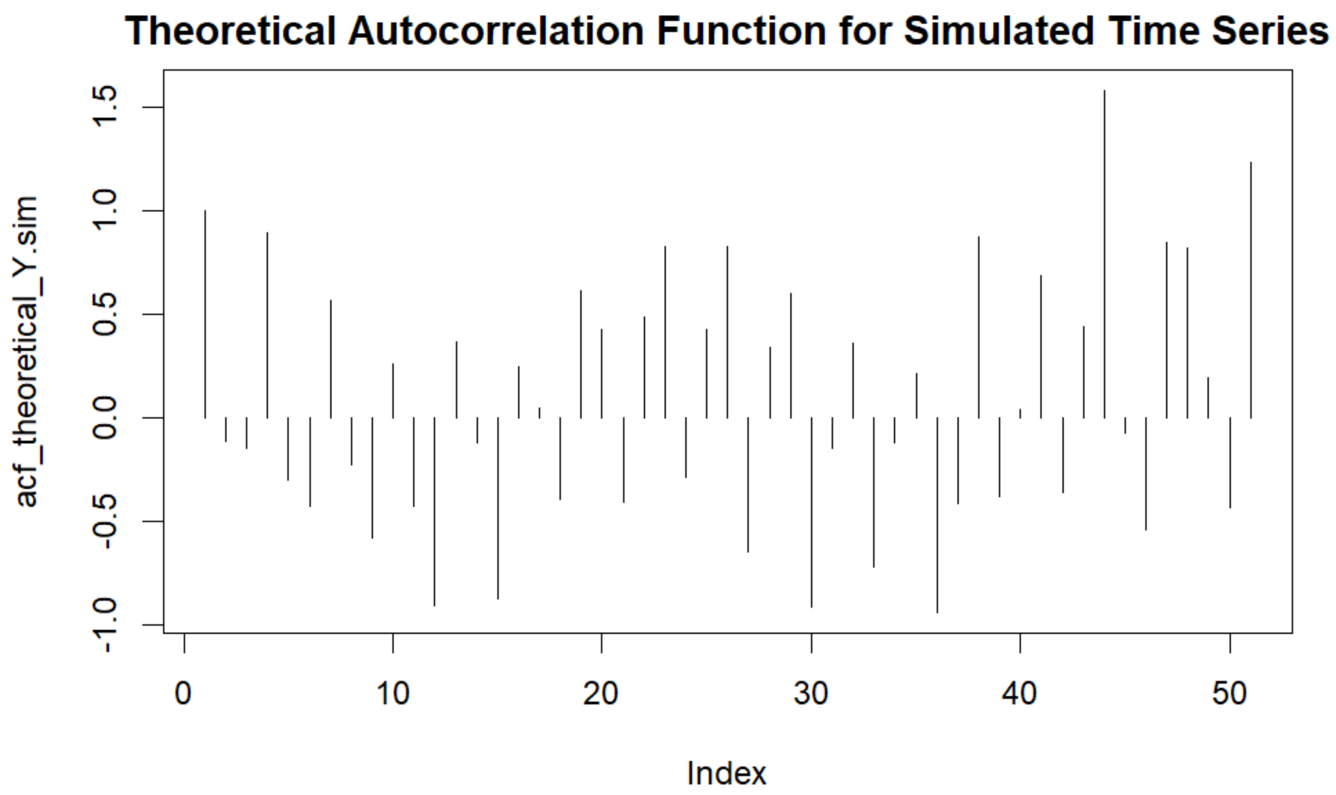
Hide

```
# Obtain theoretical ACF for Simulated ARMA(2,1) model
acf_theoretical_Y.sim <- ARMAacf(Y_sim)
acf_theoretical_Y.sim
```

```
          0           1           2           3           4           5 
 1.00000000 -0.11836897 -0.14839824  0.88887439 -0.30086498 -0.42838786 
          6           7           8           9          10          11 
 0.56645866 -0.22767914 -0.58070223  0.25610767 -0.42893741 -0.90638725 
         12          13          14          15          16          17 
 0.36448755 -0.12202829 -0.87732277  0.24544192  0.04387127 -0.39917035 
         18          19          20          21          22          23 
 0.61257883  0.42034527 -0.40628633  0.48403539  0.82678218 -0.29083885 
         24          25          26          27          28          29 
 0.42095450  0.82466900 -0.65237854  0.33962076  0.59493158 -0.91737759 
         30          31          32          33          34          35 
-0.15046747  0.35881445 -0.72061990 -0.12426214  0.20899999 -0.93956423 
         36          37          38          39          40          41 
-0.41320052  0.87087358 -0.38427591  0.03875614  0.68067493 -0.36280083 
         42          43          44          45          46          47 
 0.43813214  1.57624206 -0.07785286 -0.54197879  0.84581176  0.81992664 
         48          49          50 
 0.18874054 -0.43936611  1.22960956 
```

Hide

```
plot(acf_theoretical_Y.sim, type='h', main='Theoretical Autocorrelation Function for Simulate
d Time Series')
```



Hide

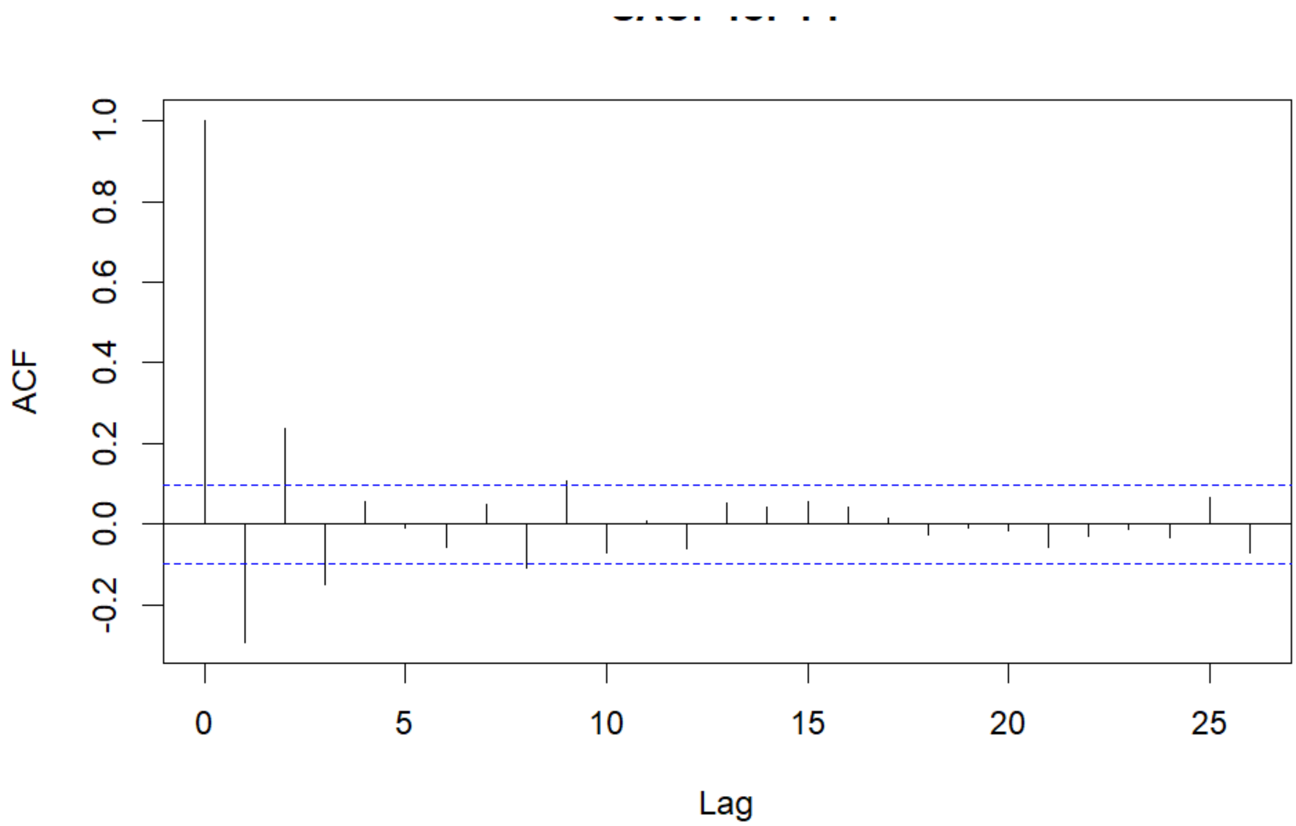
```
#From the both results it suggests a possible trend or seasonality in the time series. Also t
heres balance of positive and negative bars which could suggest a possible stationary process
```

Hide

```
#4d) Create four realizations using arima.sim with random white noise
set.seed(1)
Y1 <- arima.sim(list(order=c(2,0,1), ar=c(phi1,phi2), ma=theta1), n=400)
set.seed(2)
Y2 <- arima.sim(list(order=c(2,0,1), ar=c(phi1,phi2), ma=theta1), n=400)
set.seed(3)
Y3 <- arima.sim(list(order=c(2,0,1), ar=c(phi1,phi2), ma=theta1), n=400)
set.seed(4)
Y4 <- arima.sim(list(order=c(2,0,1), ar=c(phi1,phi2), ma=theta1), n=400)
```

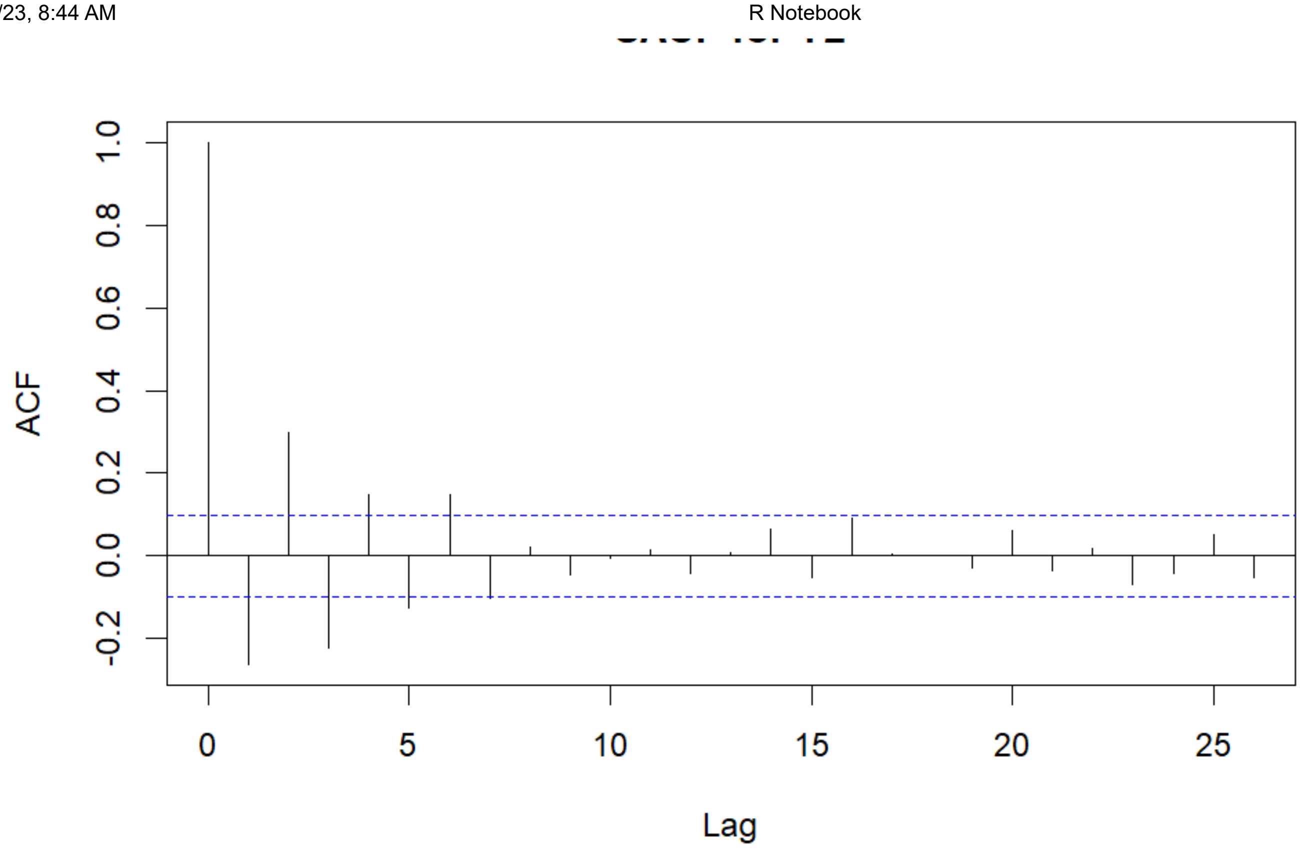
Hide

```
acf(Y1, main="SACF for Y1")
```



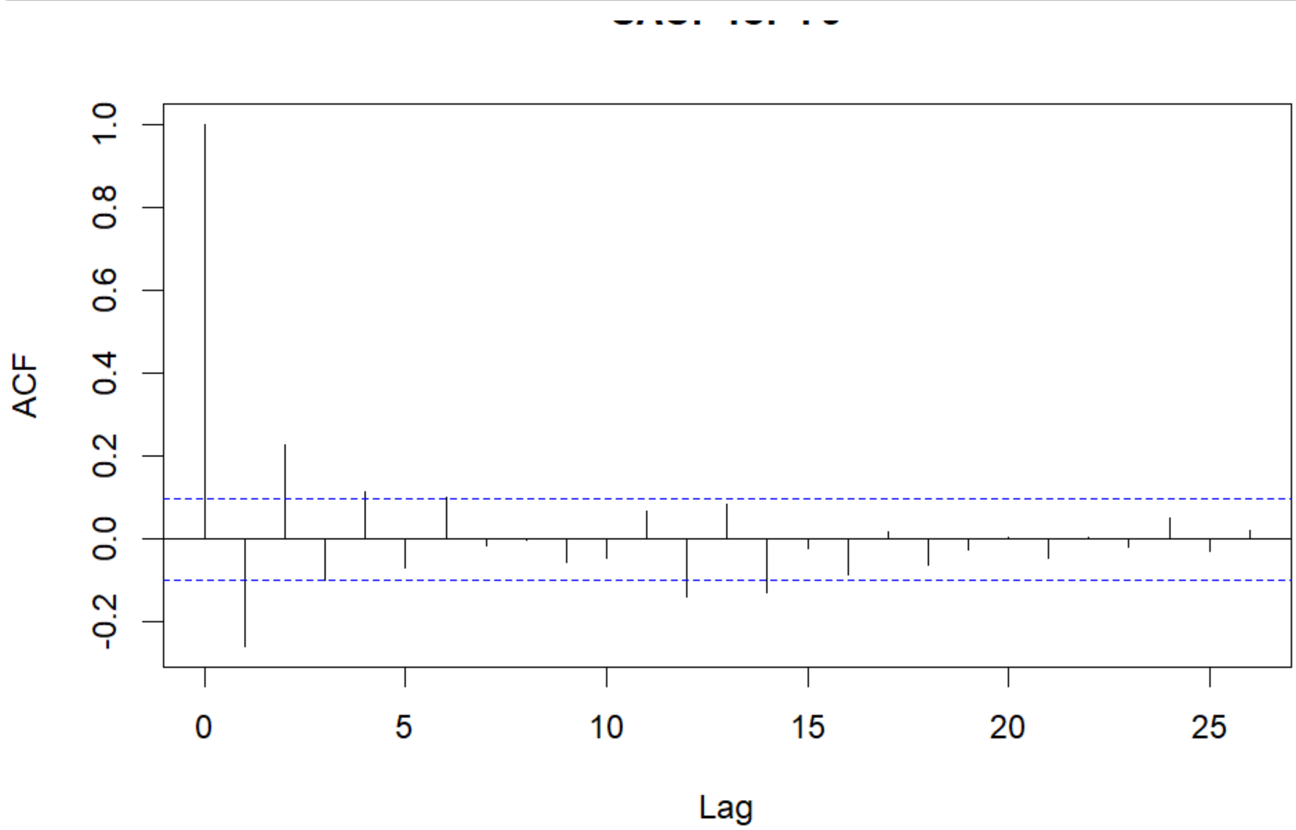
Hide

```
acf(Y2, main="SACF for Y2")
```



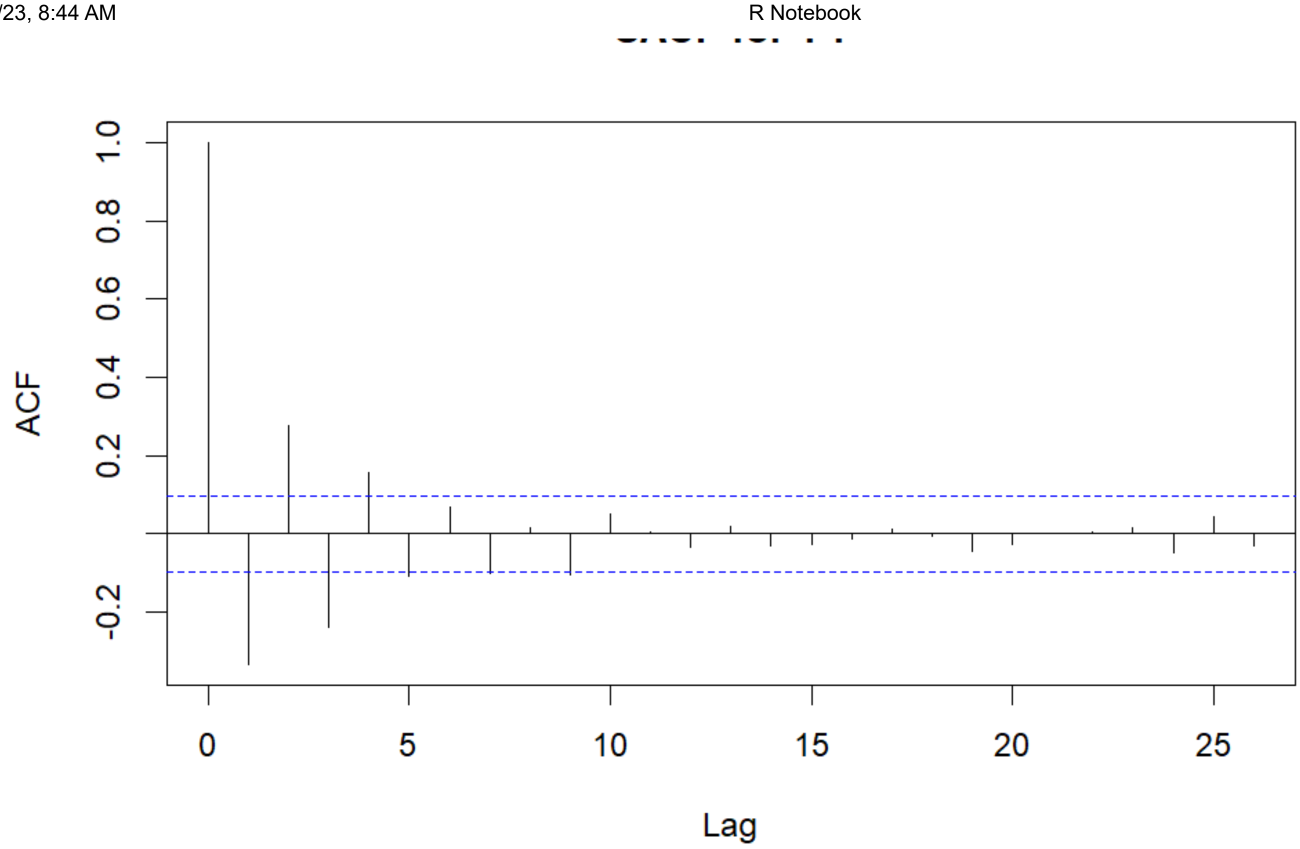
Hide

```
acf(Y3, main="SACF for Y3")
```



Hide

```
acf(Y4, main="SACF for Y4")
```



Hide

```
# Summarizing the overall shape and decay rate of the SACF for all four time series may sugge
st that the underlying ARMA(2,1) process is stable and stationary
```