## Contents

# Basics in Exploratory Data Analysis Using R

This is a simple tutorial introducing some of the basic packages and the useful functions they contain. These tools have wide-ranging applicability and can be easily adapted to financial modeling and risk management purposes.

We will roughly follow the order of topics outlined below.

1. First, we learn to simulate some price processes
2. Then, we try to find ways to fit models to the processes we just went through
3. Finally, we try to make predictions or inferences from the work above

The packages used in this tutorial are as follows:

- sde (stochastic differential equations)
- lubridate (for massaging dates)
- fGarch (for fitting Garch models)
- xlsx (for importing Excel files)
- evir (for working with Extreme Value distributions and plots)
- ggplot2 (for creating some pretty charts)
- mvtnorm (for multivariate normal)
- MASS (for fitting some common distributions)
- copula (for fitting copulas)
- gtools (used here for applying functions to running windows of data)
- tseries (for using functions/tests associated with time series)
- forecast (for forecasting)
- boot (for bootstraping)

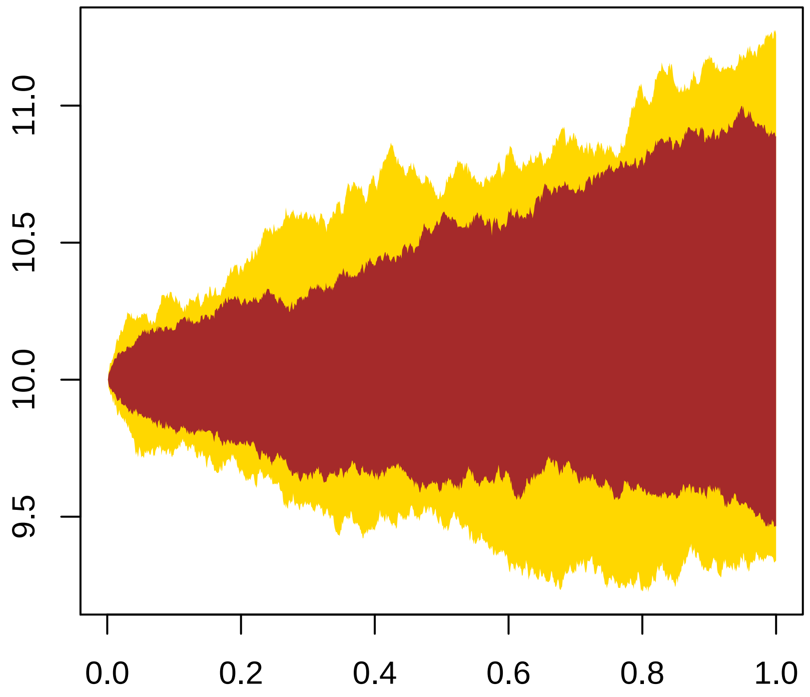
## 1 Price processes

We simulate 100 paths of each of the processes described below, plot them on a chart and show the 5th and 95th percentiles.

### 1.1 Random Walk

$$log(S_{t+\Delta t}) = log(S_t) + \mu\Delta t + \sigma\sqrt{\Delta t}Z_t$$

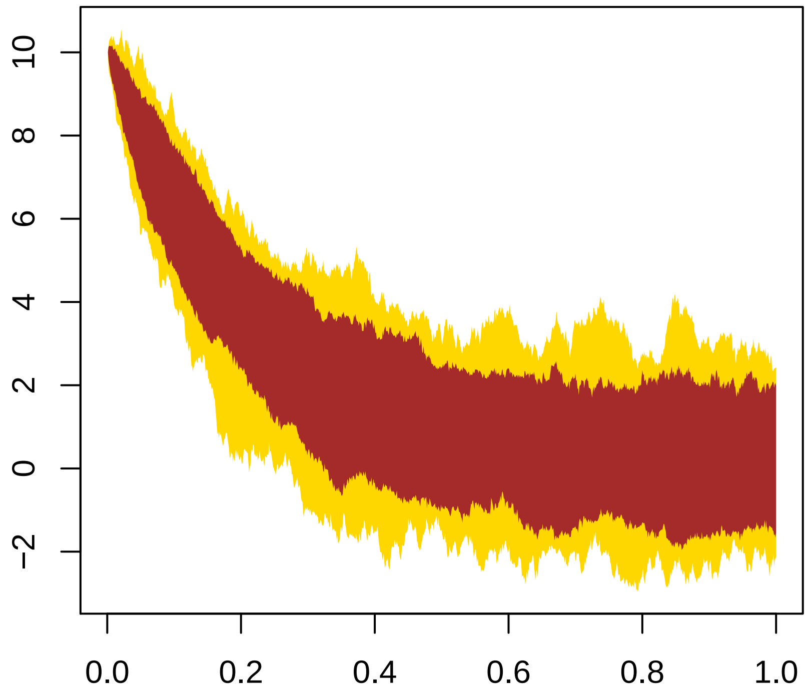
```
# Model:
# s     =log(S)
# r     =diff(s)
# mu    =mean(r)
# sigma =std(r)
# z     =(r-mu)/sigma
s_0=10
mu=0.05/250
sigma=0.2/sqrt(250)
set.seed(123)
x=matrix(s_0,ncol=100,nrow=10^3)    #store the 100 bm's in a matrix for plotting
for (t in 2: 10^3)                  #note how loop is done in R
  x[t,]=x[t-1,]+rnorm(100)*sigma+mu #note the subsetting syntax in R
plot(seq(0,1,le=1000),x[,1],"n",    #note the syntax of the plotting function
     ylim=range(x),xlab="",ylab="")    #note the beginning of the string in function call
polygon(c(1:10^3,10^3:1)/10^3,c(apply(x,1,max),
                                rev(apply(x,1,min))),col="gold",bor=F)
polygon(c(1:10^3,10^3:1)/10^3,c(apply(x,1,quantile,.95),
                                rev(apply(x,1,quantile,.05))),col="brown",bor=F)
```



## 1.2 Mean-Reverting Ornstein-Uhlenbeck Process

$$dX_t = (\theta_1 - \theta_2)dt + \theta_3 dW_t$$

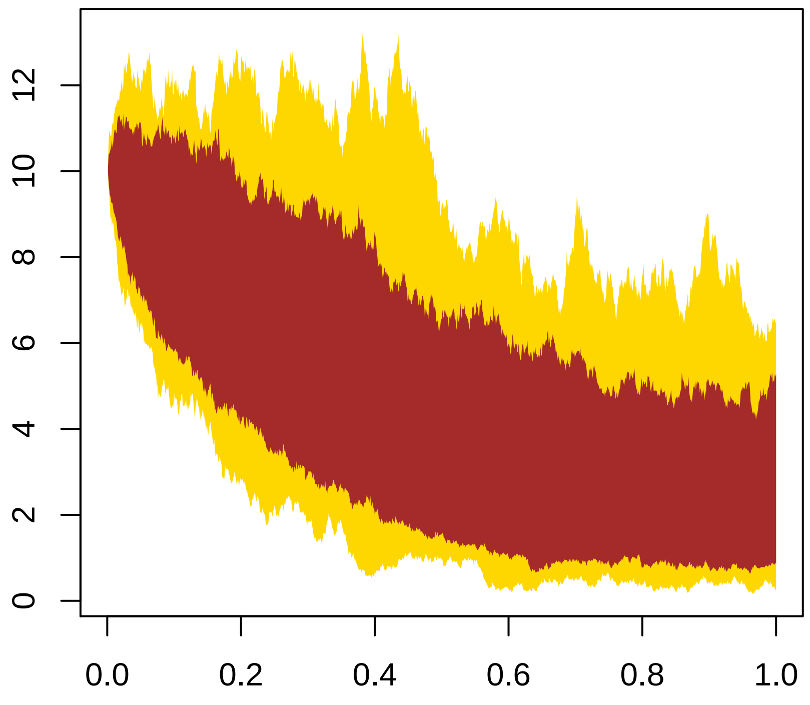
```
# Model: dXt=(theta1-theta2)*dt+theta3*dWt
require(sde)
d_OU = expression(-5 * x)
s_OU = expression(3.5)
x_OU = matrix(s_0, ncol = 100, nrow = 10^3)
x_OU = sde.sim(X0 = s_0, drift = d_OU, sigma = s_OU, N = 999, M = 100)  #N is # of simulation steps
# M is # of paths
plot(seq(0, 1, le = 1000), x_OU[, 1], "n", ylim = range(x_OU), xlab = "", ylab = "")
polygon(c(1:10^3, 10^3:1)/10^3, c(apply(x_OU, 1, max), rev(apply(x_OU, 1, min))),
    col = "gold", bor = F)
polygon(c(1:10^3, 10^3:1)/10^3, c(apply(x_OU, 1, quantile, 0.95), rev(apply(x_OU,
    1, quantile, 0.05))), col = "brown", bor = F)
```



## 1.3 Cox-Ingersoll-Ross (CIR) Process

$$dX_t = (\theta_1 - \theta_2)dt + \theta_3\sqrt{X_t}dW_t$$

```
# Model: dXt=(theta1-theta2)*dt+theta3*sqrt(Xt)*dWt
d_CIR = expression(6 - 3 * x)
s_CIR = expression(2 * sqrt(x))
x_CIR = matrix(s_0, ncol = 100, nrow = 10^3)
x_CIR = sde.sim(X0 = s_0, drift = d_CIR, sigma = s_CIR, N = 999, M = 100)
plot(seq(0, 1, le = 1000), x_CIR[, 1], "n", ylim = range(x_CIR), xlab = "",
    ylab = "")
polygon(c(1:10^3, 10^3:1)/10^3, c(apply(x_CIR, 1, max), rev(apply(x_CIR, 1,
    min))), col = "gold", bor = F)
polygon(c(1:10^3, 10^3:1)/10^3, c(apply(x_CIR, 1, quantile, 0.95), rev(apply(x_CIR,
    1, quantile, 0.05))), col = "brown", bor = F)
```

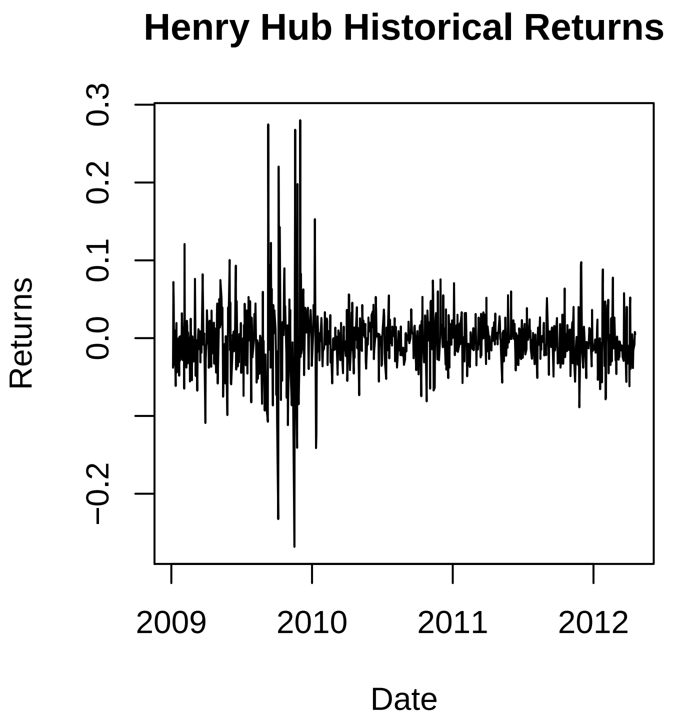


## 2 Fitting Models

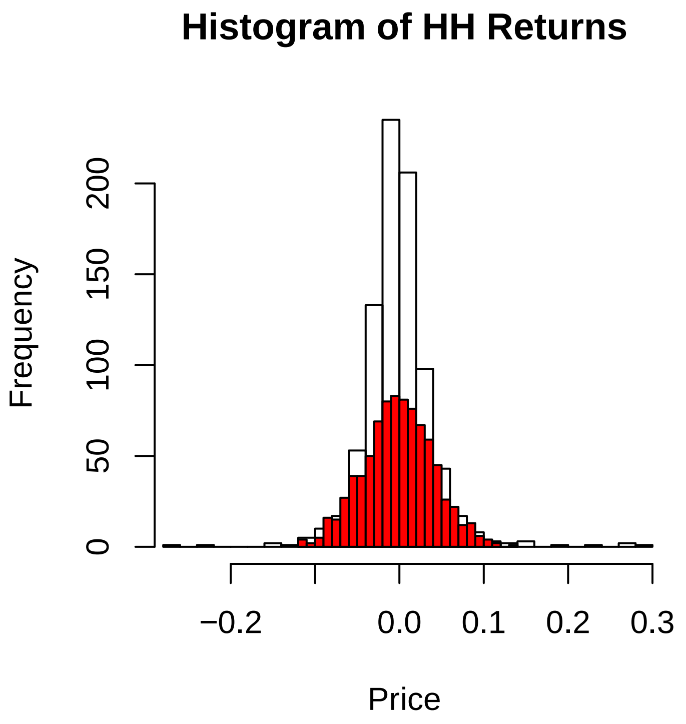
### 2.1 Some Exploratory Data Analysis

We plot the histogram of the historical Henry Hub returns data and compare it to that of a normal distribution

```
require(xlsx)
histRet = read.xlsx("C:\\Users\\Yangster\\Dropbox\\Private\\HistPowerGasSpot_Values2.xlsx",
    1)
retHH = histRet[which(histRet$Hub == "HH"), ]

attach(retHH)
require(lubridate)  #Lubridate is a useful dates package
idx.remove = which(Price == 0)  #Clean up HH prices/returns for non-trading days
retHH = retHH[-idx.remove, ]

y = ymd(as.character(Date))
par(mar = c(4, 5, 4, 2))  #Note how to set margins of the plotting device
plot(y, Price, "l", xlab = "Date", ylab = "Returns", main = "Henry Hub Historical Returns")
```

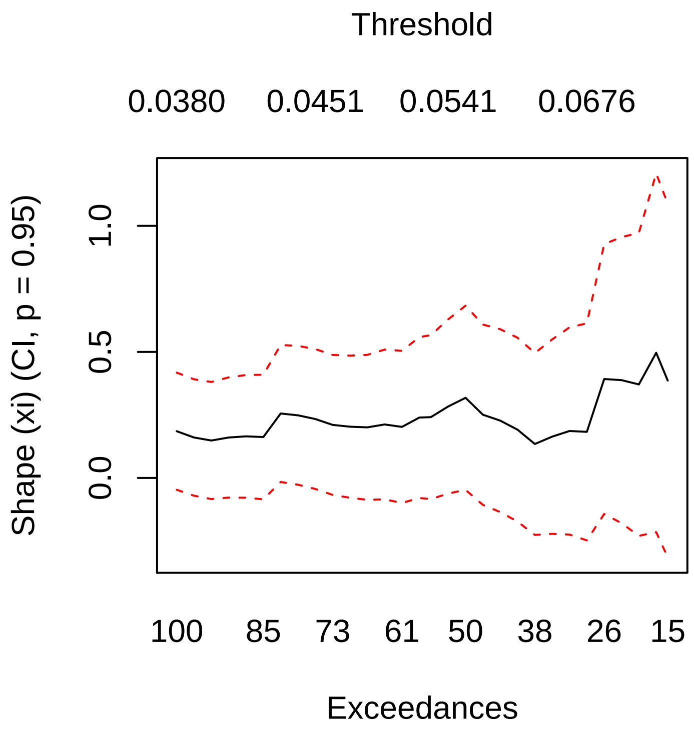


```r
hist(Price, breaks = 30, xlim = range(Price), main = "Histogram of HH Returns")
set.seed(123)
pSim = rnorm(length(Price), mean = mean(Price), sd = sd(Price))
hist(pSim, breaks = 30, add = T, col = "red")
```

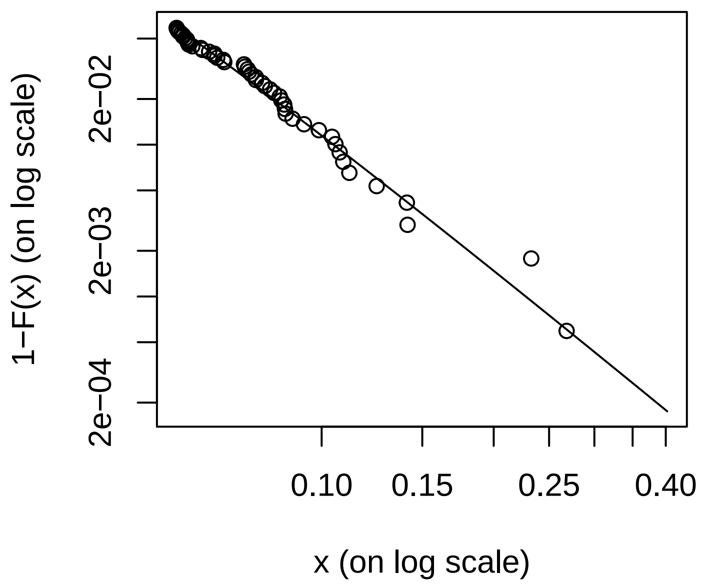


Since we are more concerned about the tails of the distribution, we use the functionality of fitting Generalized Pareto Distribution in the **evir** package and examine the fit of the model with some diagnostic plots.

```
require(evir)
# Creates a plot showing how the estimate of shape varies with threshold or
# number of extremes.
shape(Price * -1, models = 30, start = 15, end = 100)
```

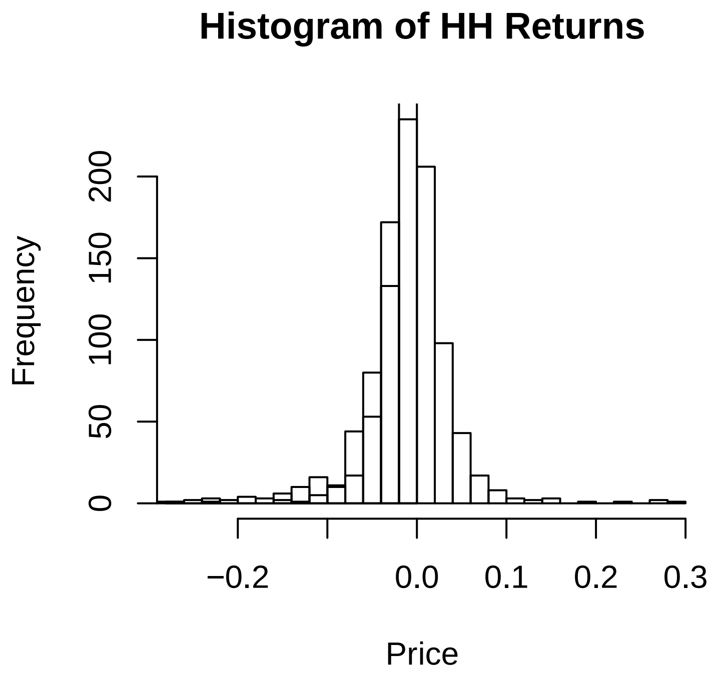


```
# Fitting Generalized Pareto Distribution to data
hh.est = gpd(Price * -1, nextremes = 50)
tailplot(hh.est)
```



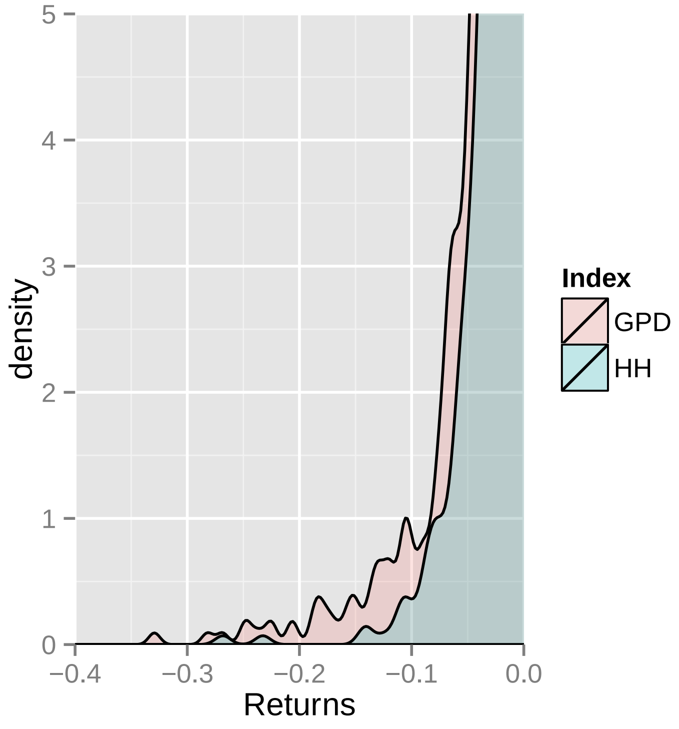
Additionally, we can choose from diagnostic graphs created by the command `plot.gpd()`.

```
# Simulate from the fitted GPD and compare to the empirical data
xi.est = hh.est$par.ests[1]
beta.est = hh.est$par.ests[2]
hist(Price, breaks = 30, xlim = range(Price), main = "Histogram of HH Returns")
set.seed(123)
pSim.gpd = rgpd(length(Price), xi = xi.est, beta = beta.est) * -1
hist(pSim.gpd, breaks = 30, add = T)  #add=T argument overlays the second histogram.
```



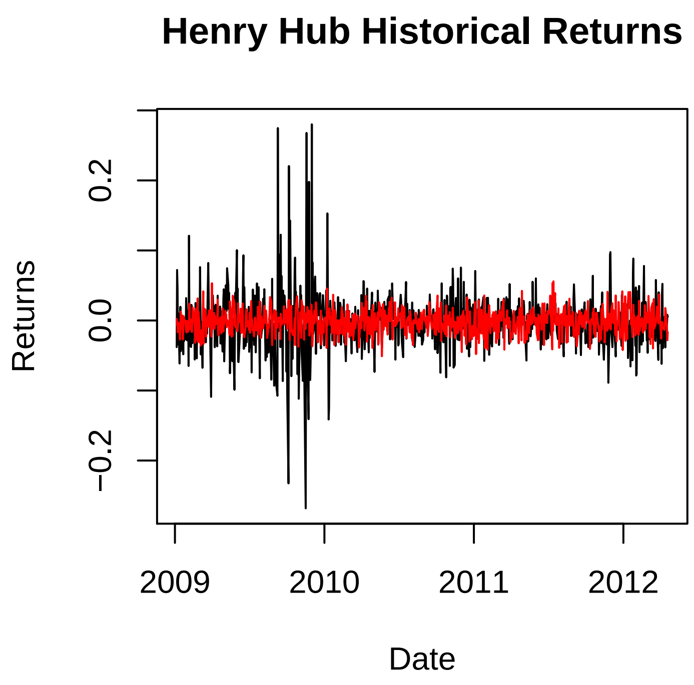
The overlay of histograms has become harder to see in the graph above. This prompts us to choose a different plotting method so we have more control over how the output is shown.

```
# Using the ggplot2 package for visualization
require(ggplot2)
idx = c(rep("HH", length(Price)), rep("GPD", length(Price)))
hist.Comb = data.frame(idx, c(Price, pSim.gpd))
colnames(hist.Comb) = c("Index", "Returns")
ggplot(hist.Comb, aes(Returns, fill = Index)) + geom_density(alpha = 0.2) +
    coord_cartesian(xlim = c(-0, -0.4), ylim = c(0, 5))
```



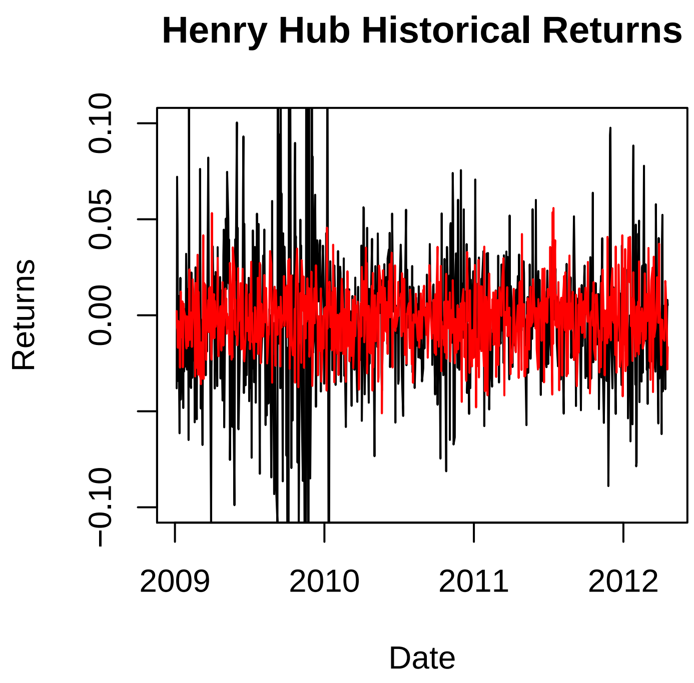
### 2.2 Fitting a GARCH (1,1) Model

```
require(fGarch)
modGarch <- garchFit(formula = ~garch(1, 1), data = Price, cond.dist = "norm")
summary(modGarch)
str(modGarch)  #To see the structure of the object returned by modGarch for later use
set.seed(123)
parList = modGarch@fit$par
modGarchSpec = garchSpec(model = parList, cond.dist = "norm")
simGarch <- garchSim(modGarchSpec, n = length(y))
plot(y, Price, "l", xlab = "Date", ylab = "Returns", main = "Henry Hub Historical Returns")
lines(y, simGarch, col = "red")
```

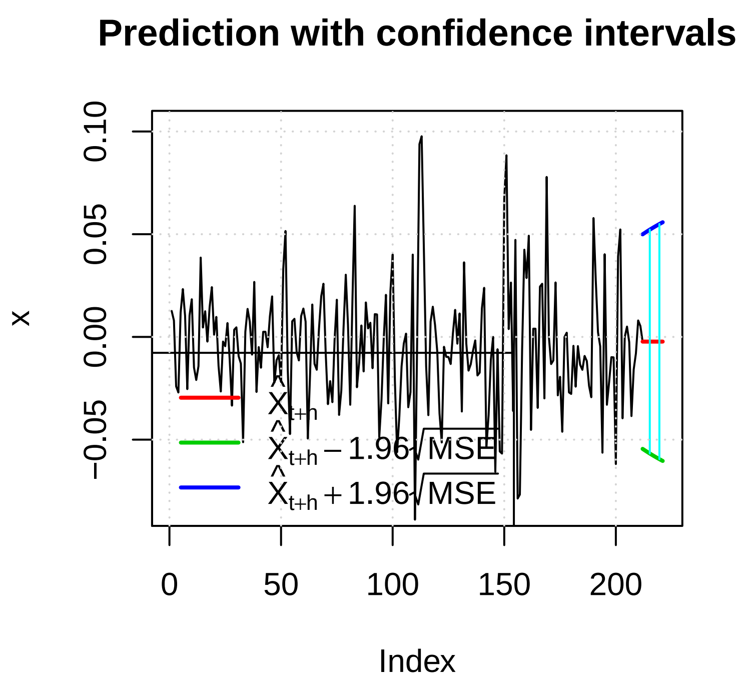


The larger swings don't seem to be captured very well in the plot above. Is this because the Garch model is not calibrating to the outlier values in 2009-2010 period? Let's zoom in on a smaller band.

```r
plot(y, Price, "l", xlab = "Date", ylab = "Returns", main = "Henry Hub Historical Returns",
    ylim = c(-0.1, 0.1))
lines(y, simGarch, col = "red")
```

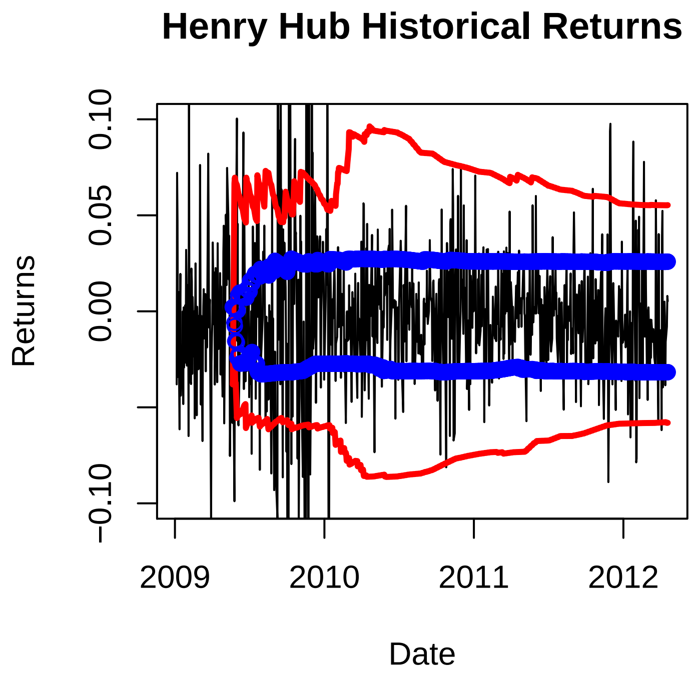


```
predict(modGarch, n.ahead = 10, plot = TRUE)
```



We can also plot the 5th and 95th quantiles of the empirical data against the same quantiles of the simulated data using the fitted Garch model to gaugue how well the model performs.

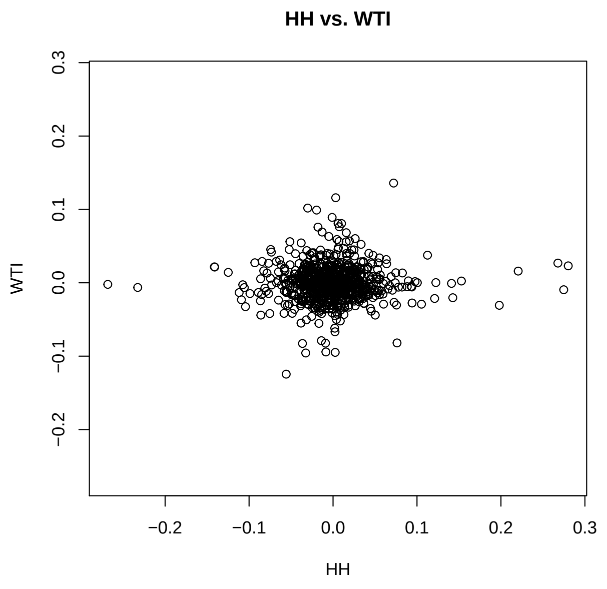
```
quant = matrix(0, ncol = 2, nrow = (length(y) - 99))
quantSim = matrix(0, ncol = 2, nrow = (length(y) - 99))
for (i in 100:length(y)) {
    quant[i - 99, 1] = quantile(Price[i - 99:i], probs = 0.05)
    quant[i - 99, 2] = quantile(Price[i - 99:i], probs = 0.95)
}

plot(y, Price, "l", xlab = "Date", ylab = "Returns", main = "Henry Hub Historical Returns",
    ylim = c(-0.1, 0.1))
lines(y[100:length(y)], quant[, 1], col = "red", ty = "l", lwd = 3)
lines(y[100:length(y)], quant[, 2], col = "red", ty = "l", lwd = 3)
for (i in 100:length(y)) {
    quantSim[i - 99, 1] = quantile(simGarch[i - 99:i], probs = 0.05)
    quantSim[i - 99, 2] = quantile(simGarch[i - 99:i], probs = 0.95)
}
lines(y[100:length(y)], quantSim[, 1], col = "blue", ty = "p")
lines(y[100:length(y)], quantSim[, 2], col = "blue", ty = "p")
```

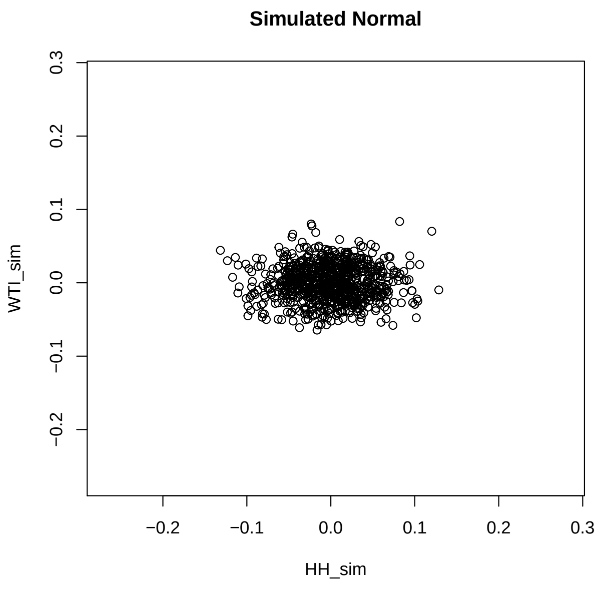


### 2.3 Analysis of Dependence Structure

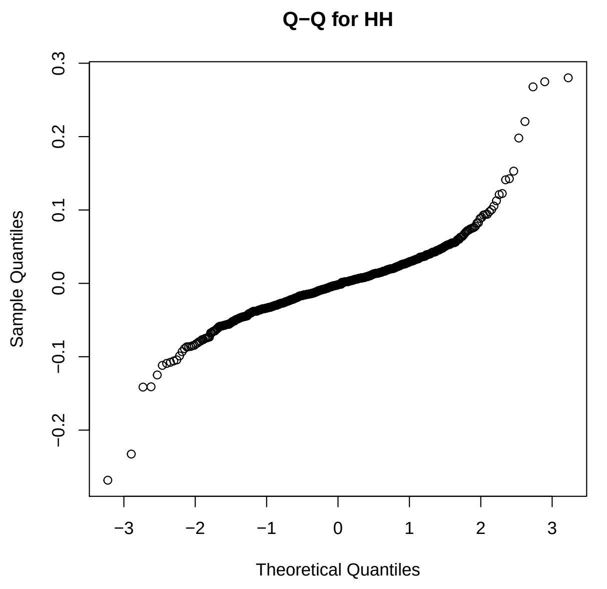
We begin with some simple plots showing the relationship between WTI and Henry Hub returns. We plot in addition the Q-Q plots for the marginals (WTI and HH) separately, a simulated bivariate normal, and side-by-side plot of the two returns series.

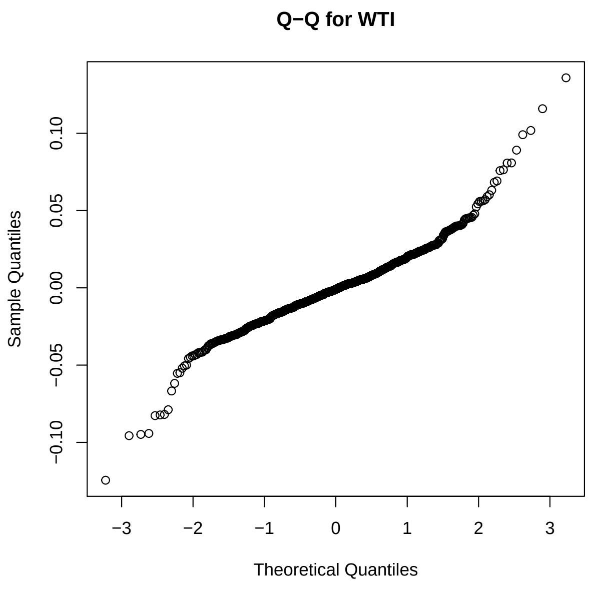
```r
require(xlsx)
histRet2 = read.xlsx("C:\\Users\\Yangster\\Dropbox\\Private\\HistOilGasSpot.xlsx",
    1)
attach(histRet2)
require(lubridate)
y = ymd(as.character(Date))
range_axis = range(HH)
par(pty = "s")
plot(HH, WTI, xlim = range_axis, ylim = range_axis, main = "HH vs. WTI")
par(pty = "m")
mu_2 = c(mean(HH), mean(WTI))
sigma_2 = var(cbind(HH, WTI))
require(mvtnorm)
simNorm_2 = rmvnorm(length(HH), mean = mu_2, sigma = sigma_2)
par(mfrow = c(1, 2))
plot(HH, WTI, xlim = range_axis, ylim = range_axis, main = "HH vs. WTI")
plot(simNorm_2, xlim = range_axis, ylim = range_axis, main = "Simulated Normal",
    xlab = "HH_sim", ylab = "WTI_sim")
```



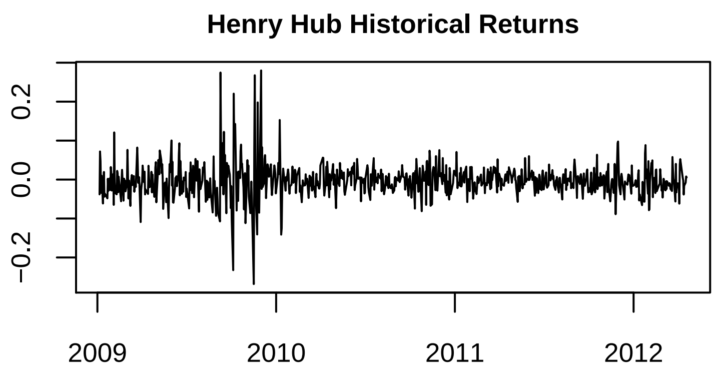


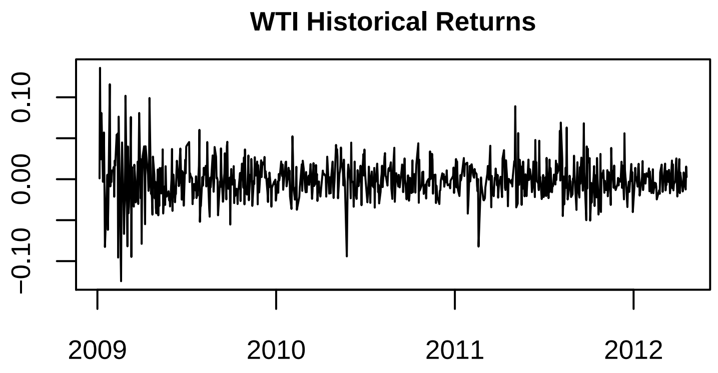
```
qqnorm(HH, main = "Q-Q for HH")
qqnorm(WTI, main = "Q-Q for WTI")
```





```
# Plot the returns series side by side
par(mfrow = c(2, 1))
par(mar = c(2, 2.5, 2, 1))
plot(y, HH, "l", main = "Henry Hub Historical Returns", cex.lab = 0.8, cex.main = 0.8,
    cex.axis = 0.8)
plot(y, WTI, "l", main = "WTI Historical Returns", cex.lab = 0.8, cex.main = 0.8,
    cex.axis = 0.8)
```





```r
par(mfrow = c(1, 1))
par(mar = c(4, 5, 4, 2))
```

Next we look at some measures of dependence.

```r
# Pearson's linear correlation
cor(HH, WTI, method = "pearson")

## [1] 0.02568

# Kendall's tau
cor(HH, WTI, method = "kendall")

## [1] 0.005194

# Spearman's rho
cor(HH, WTI, method = "spearman")

## [1] 0.008645
```

We also look at the distribution for Henry Hub and WTI separately.

```r
# Fit both t and normal distributions to compare loglik
require("MASS")
fitdistr(HH, "t")

##          m             s            df
##    -0.002285     0.026349     3.184829
##   ( 0.001139)  ( 0.001171)  ( 0.382361)

fitdistr(HH, "normal")

##       mean          sd
##   -0.001208    0.042982
##  ( 0.001525) ( 0.001079)

fitdistr(WTI, "t")

##        m            s            df
##   -0.0010697    0.0173135    3.8717878
##  ( 0.0007314) ( 0.0007525) ( 0.5441782)

fitdistr(WTI, "normal")

##       mean          sd
##   -0.0004216    0.0245263
##  ( 0.0008704) ( 0.0006155)
```

To fit the full dependence structure, we use the marginal + copula approach (See *Statistics and Data Analysis for Financial Engineering* (P198) for further details).

1. we fit t distribution to each of the marginals
2. we define a mvdc object (in the **copula** package)
3. we pass some initial values to an *optim* routine to searh for parameters (the search can fail)

Before we do so, we briefly mention some classes of copulas commonly encountered in the literature.

1. Copula families: Elliptical Copula
   (a) Gausssian
   (b) Student-t
2. Copula families: Arhimedian Copula
   (a) Clayton's Copula
   (b) Gumbel's Copula
   (c) Frank's Copula
3. Copula families: Extreme Value Copula

```
est.WTI = as.numeric(fitdistr(WTI, "t")$estimate)
est.HH = as.numeric(fitdistr(HH, "t")$estimate)
# The scale parameter of the fitted t is converted to sd below
est.WTI[2] = est.WTI[2] * sqrt(est.WTI[3]/(est.WTI[3] - 2))
est.HH[2] = est.HH[2] * sqrt(est.HH[3]/(est.HH[3] - 2))
cor_tau = cor(WTI, HH, method = "kendall")
require(copula)
require(fGarch)  # needed for pstd function
cop_t_dim2 = tCopula(cor_tau, dim = 2, dispstr = "un", df = 4)
# The obs were standardized into the [0,1] interval to work with the est
# method
data1 = cbind(pstd(WTI, mean = est.WTI[1], sd = est.WTI[2], nu = est.WTI[3]),
    pstd(HH, mean = est.HH[1], sd = est.HH[2], nu = est.HH[3]))
```

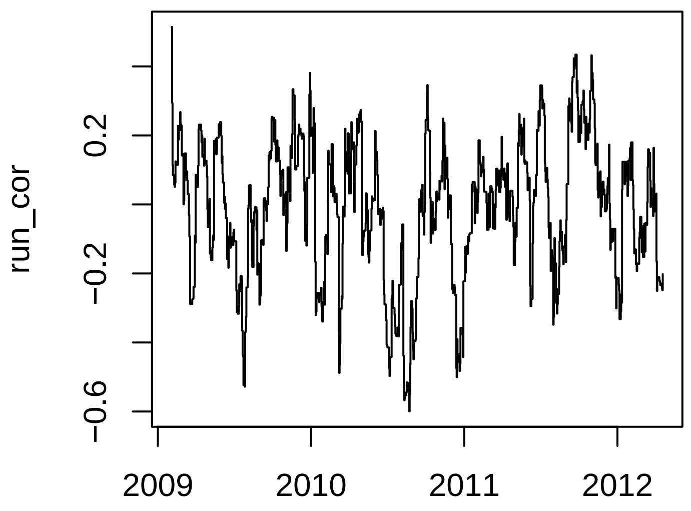
```
# Convergence issue below!!!!
ft1 = fitCopula(cop_t_dim2, method = "mpl", data = data1, start = c(cor_tau,
    4))

## Warning:  possible convergence problem:  optim() gave code=1
```

Although the optimization problem didn't converge, we finish the code for the sake of completion. If *ft1* did find solutions, the following code (which we leave unevaluated) would have been used to search for a viable copula.

```
mvdc_t = mvdc(cop_t_dim2, c("std", "std"), list(list(mean = est.WIT[1], sd = est.WIT[2],
    nu = est.WIT[3]), list(mean = est.HH[1], sd = est.HH[2], nu = est.HH[3])))

start = c(est.WIT, est.HH, ft1@estimate)
fitMvdc(cbind(WTI, HH), mvdc_t, start)
```

As a further exploration, we look at the running correlation between the two series at 20 data point intervals. The lack of structure from this plot may indicate why the copula approach may fail. In other words, there may not be much of dependence structure in the data itself, or it could be time-varying.

```
require(gtools)
run_cor = running(HH, WTI, width = 20, fun = cor, allow.fewer = FALSE)
plot(Date[20:length(Date)], run_cor, type = "s", xlab = "")
```

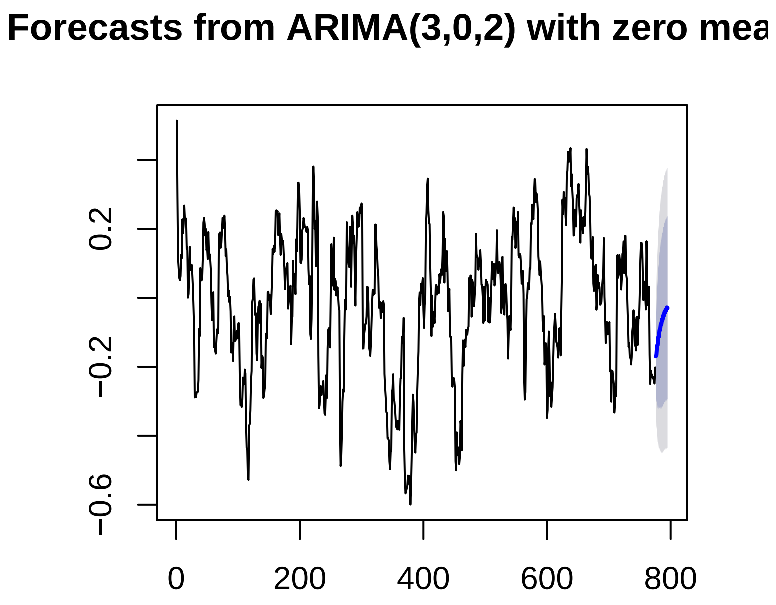


Is the running correlation a stationary series?

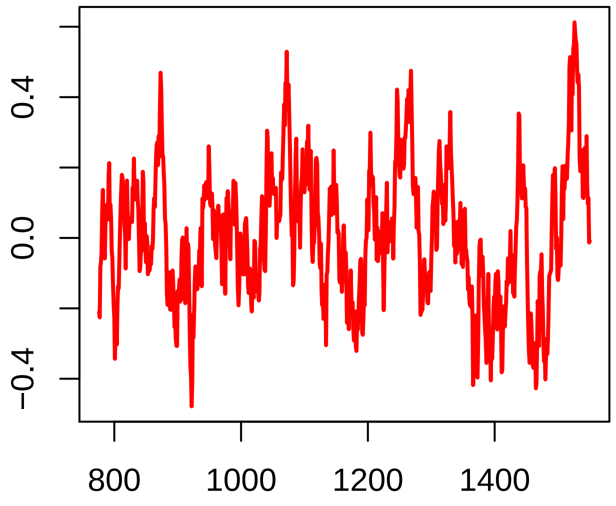
```
require(tseries)
adf.test(run_cor)

## Warning:  p-value smaller than printed p-value

##
##  Augmented Dickey-Fuller Test
##
## data:  run_cor
## Dickey-Fuller = -5.38, Lag order = 9, p-value = 0.01
## alternative hypothesis: stationary
```

Can we use a classic ARIMA model to try to predict future correlations? Below we let R automatically search for an ARIMA structure based on Maximum Likelihood. We also plot a series based on the simulated data generated by the ”best” model selected as well as its histogram.

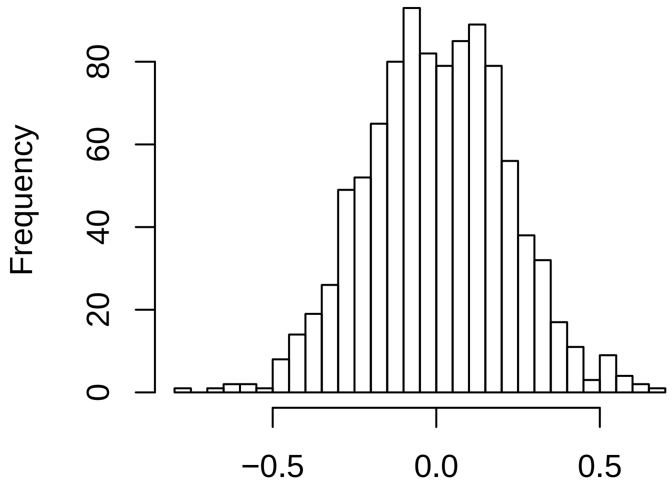
```r
require(forecast)
cor.fit = auto.arima(run_cor)
plot(forecast(cor.fit, h = 20))
```



```r
set.seed(123)
plot(simulate(cor.fit, nsim = length(run_cor)), lwd = "2", col = "red", ylab = "",
    xlab = "")
```



```
cor.mat = matrix(ncol = 1000, nrow = length(run_cor))
for (i in 1:1000) cor.mat[, i] = simulate(cor.fit, nsim = length(run_cor))
hist(cor.mat[length(run_cor), ], breaks = 30, main = "", xlab = "")
```



As a reference, the summary report of the **auto.arima** process used to select the ”best” model is shown below.

```
summary(cor.fit)

## Series: run_cor
## ARIMA(3,0,2) with zero mean
##
## Coefficients:
##         ar1    ar2     ar3    ma1     ma2
##       0.587  0.878  -0.533  0.439  -0.521
## s.e.  0.455  0.049   0.386  0.478   0.459
##
## sigma^2 estimated as 0.00536:  log likelihood=925.2
## AIC=-1838   AICc=-1838   BIC=-1811
##
## Training set error measures:
##                     ME    RMSE    MAE    MPE  MAPE   MASE      ACF1
## Training set -0.001502 0.07322 0.0516 -70.44 195.1 0.9959 -0.006017
```