# Univariate Return Modeling

cristal.wang111@gmail.com (mailto:cristal.wang111@gmail.com)

# 1.Univariate Return Modeling

```
library(tseries)
```

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

## 1.1.Data

- Get daily S&P500 data from Yahoo! Finance
    - `drop = TRUE` : It drops extra attributes/dimensions if only one quote is requested.
        - So you'll just get a **numeric time series (of class `ts` or `zoo` )**, instead of a multi-column object like `xts` or `data.frame` .
        - `drop = FALSE` , you'd get a more complex object with one column named `"AdjClose"` .

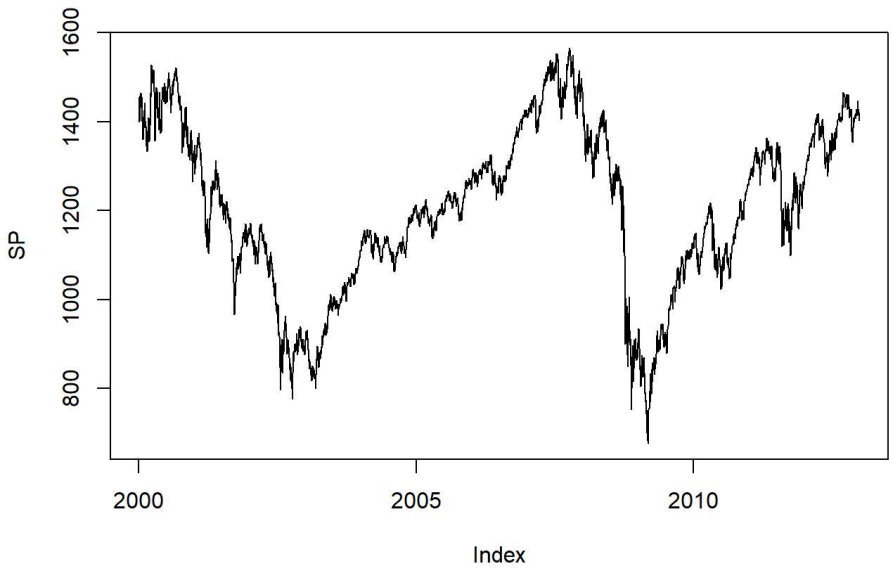
```
SP=get.hist.quote("^GSPC", start='2000-01-01', end="2012-12-31", quote="AdjClose")
```

```
## time series starts 2000-01-03
## time series ends   2012-12-28
```

```
class(SP)
```

```
## [1] "zoo"
```

```
plot(SP) # Index Level
```



# 1.2.Returns & time-series plot

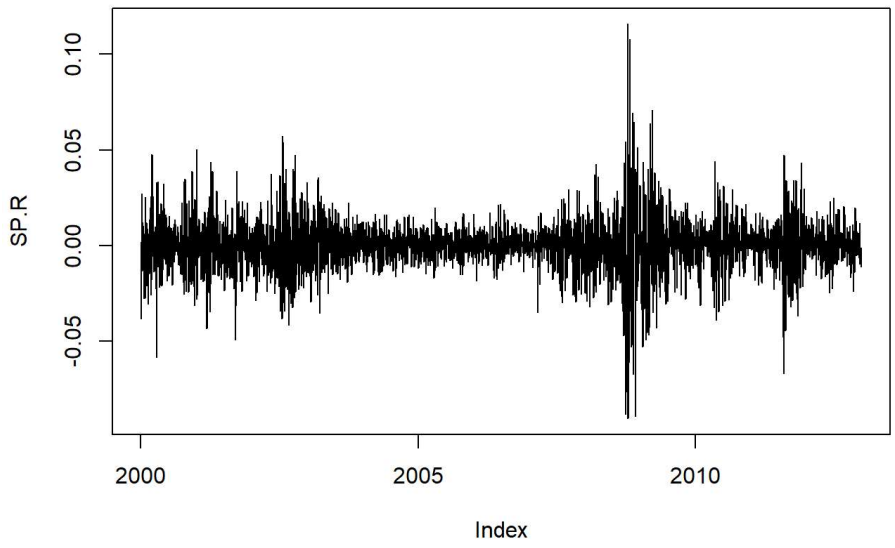
## 1.2.1.Net Return

The net return Rt is calculated as:

$$Net\ Return\ R_t = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{P_t}{P_{t-1}} - 1\ , where\ P_t = \text{price of asset at t; Gross Return} = \frac{P_t}{P_{t-1}} = 1 + R_t$$

- `R = diff(SP)/as.numeric(SP[-length(SP)])`
- `R = quantmod::Delt(SP)[-1]` # Remove first NA return
  - `Delt` : calculates percentage changes (returns) of a given time series. But 1th is NA. So here remove 1st NA.
- `R = diff(SP)/lag(SP,-1)` # lag(x, k): shifts values of a time series by k periods.

```
SP.R = diff(SP)/as.numeric(SP[-length(SP)]) # Net returns
plot(SP.R)
```

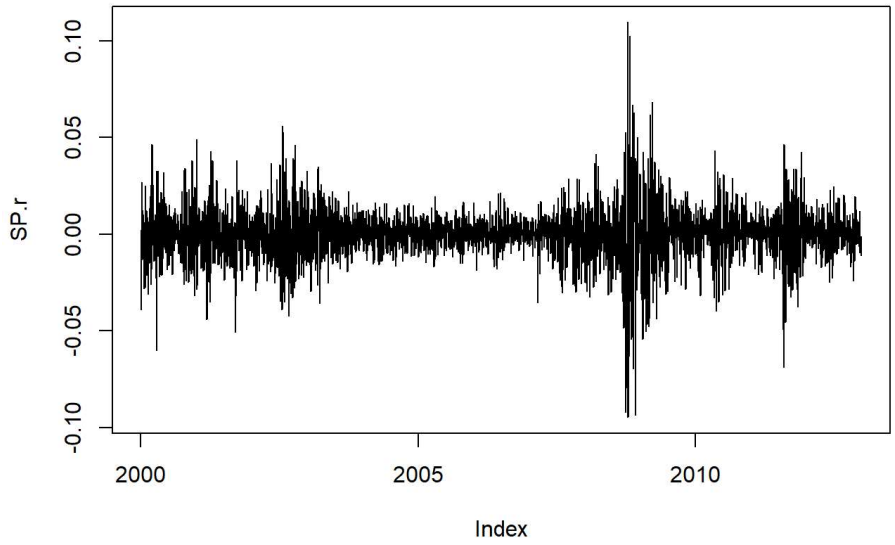



## 1.2.1.Log Return

$$Log\ Return\ r_t = \log(GrossReturn) = log(\frac{P_t}{P_{t-1}}) = log(1 + R_t) = p_t - p_{t-1},\ where\ p_t = log(P_t) = logPrice$$

- `SP.r = diff( log( SP ) )`
- `log(1+SP.R)`

```
SP.R = diff(SP)/as.numeric(SP[-length(SP)]) # Net returns
SP.r=log(1+SP.R) # Log returns. R = exp( r ) - 1 <=> r = log( R + 1 )
plot(SP.r)
```




Here, The returns exhibits volatility clustering (time series plot)

# 1.3.Descriptive Stats

## 1.3.0.Set Input

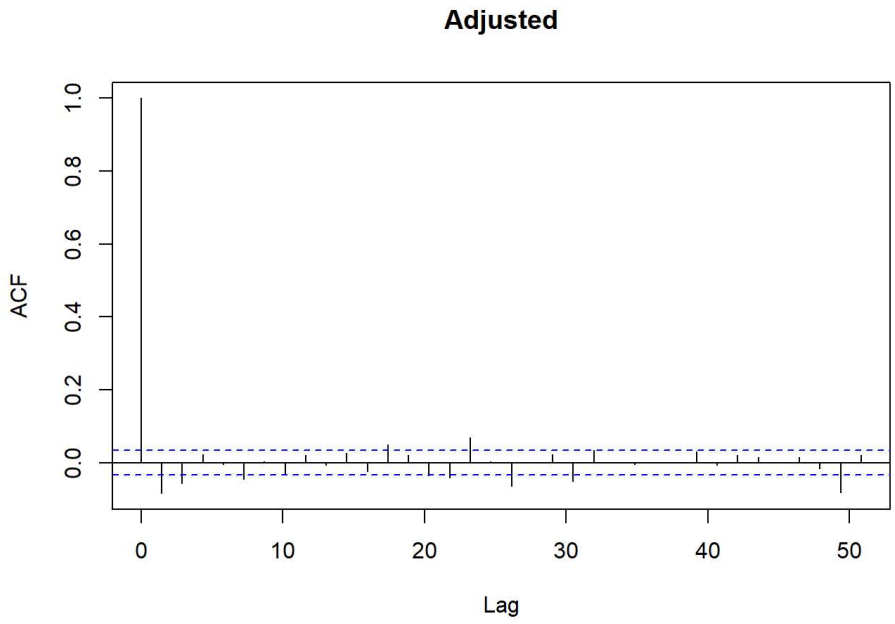
```
# INPUT (Return)
data = SP.r #Log return
```

## 1.3.1.autocorrelation

`acf` : The function to compute the **autocorrelation function.**

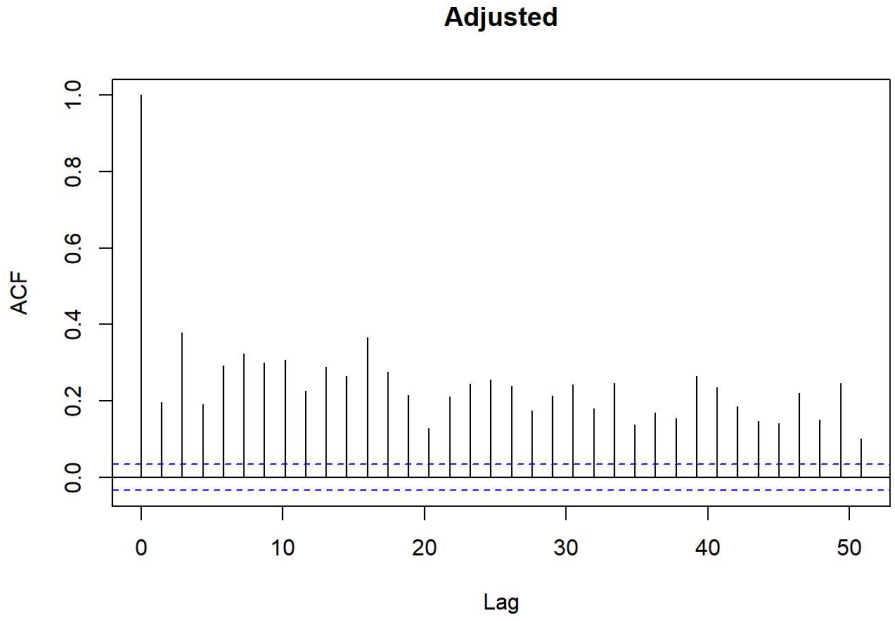
- **Y-axis ("ACF")**: Autocorrelation values (range -1 to 1)**.**
  - **Lag 0** has autocorrelation = 1 (always the case; a series is perfectly correlated with itself)**.**
- **X-axis ("Lag")**: Time lags (e.g., lag-1 is today vs yesterday, lag-2 is today vs 2 days ago, etc.)
- **Bars**: Each bar shows the autocorrelation at that specific lag**.**
- **Blue dashed lines**: 95% confidence bounds. If a bar lies **outside** these lines, it means the autocorrelation at that lag is **statistically significant** at the 5% level.
  - All other lags are **within the blue bounds**, meaning there is **no significant autocorrelation** at those lags**.**

```
# return autocorrelation
acf(data,  na.action = na.remove)
```




Here, The data ( log returns/returns) is uncorrelated (ACF plot).

```
# square-return autocorrelation
acf(data^2,  na.action = na.remove )
```




## 1.3.2.histogram

`data` : The numeric vector you're plotting.

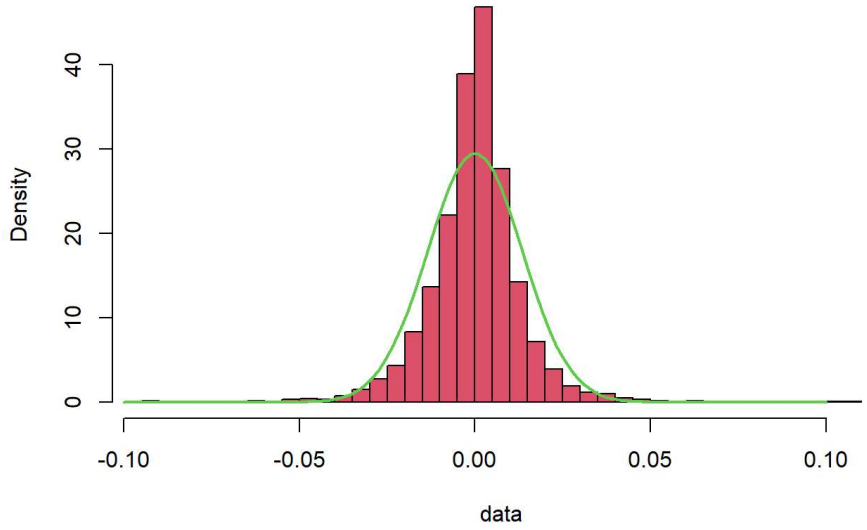
`60` : The number of **bins** (intervals) in the histogram. More bins = finer granularity.

`col=2` : Sets the color of the bars. `2` is red in base R.

`freq=FALSE` : This is key — it tells R to plot **density** (i.e., area under bars = 1), not raw counts.

`main=""` : Leaves the plot **title blank**.

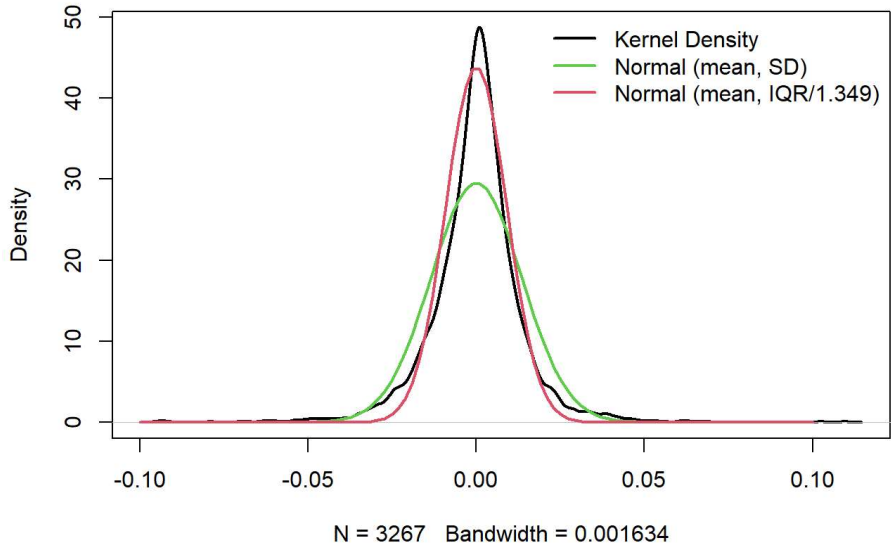
```
a=hist(data, 60, col=2, freq=FALSE,main="") # histogram

x=seq(-.1,.1,len=100)
lines(x,dnorm(x,mean(data),sd(data)), lwd=2, col=3) # overlay Normal
```




## 1.3.3.Kernel density, Normal,

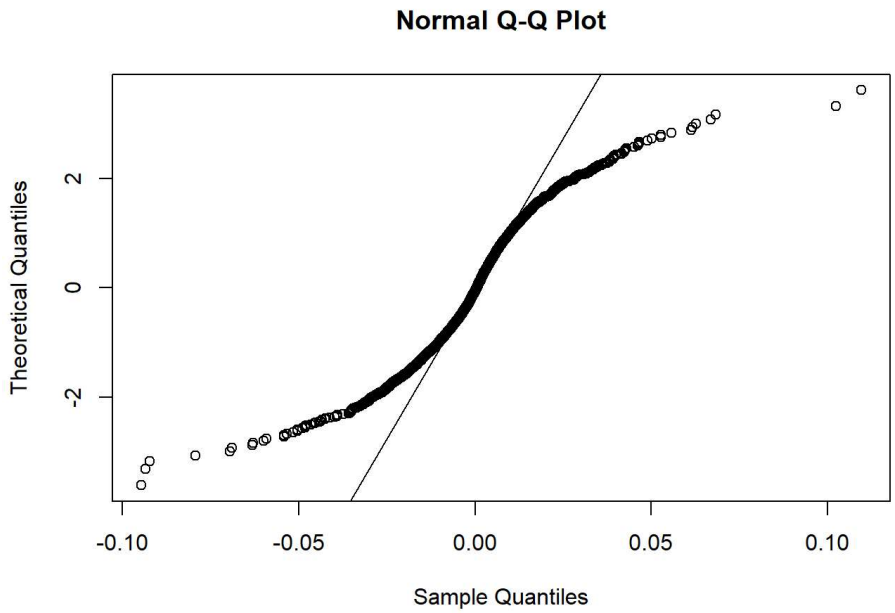
IQR is the difference between the 75th and 25th percentiles. IQR(Normal)= 0.674−(−0.674)=1.349

```
plot(density(data),lwd=2,main="") # Kernel density estimator
lines(x,dnorm(x,mean(data),sd(data)), lwd=2, col=3) # overlay sample normal
lines(x,dnorm(x,mean(data),IQR(data)/1.349), lwd=2, col=2) # overlay sample IQR normal

legend("topright", legend=c("Kernel Density", "Normal (mean, SD)", "Normal (mean, IQR/1.349)"),col=c("black", 3, 2), lwd=2, bty="n")
```




## 1.3.4.qqplot

```
qqnorm(data, datax=TRUE); qqline(data,datax=TRUE) # Normal qqplot
```




Here, Q-Q reveals differences in distribution's tails, it have much heavier tails than Normal (Normal QQ-plot).

# 1.4.Normality tests

Here all **Null Hypothesis ($H_0$)**: Data is normally distributed.

- **Kolmogorov-Smirnov:** Based on distance of empirical & Normal CDF.
- **Jarque-Bera**: Based on skewness & kurtosis, combined.
- **Shapiro-Wilk** (most powerful): Based on sample & theoretical quantiles (QQplot). Very sensitive to small departures from normality, but only works for n ≤ 5000.

```
## INPUT
data = SP.r

## Normality Test
ks.test( as.vector(data), 'pnorm', mean(data), sd(data) ) # Kolmogorov-Smirnov test
```

```
## Warning in ks.test.default(as.vector(data), "pnorm", mean(data), sd(data)):
## ties should not be present for the one-sample Kolmogorov-Smirnov test
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  as.vector(data)
## D = 0.079772, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
jarque.bera.test(data) # Jarque-Bera test { in package(tseries) }
```

```
##
##  Jarque Bera Test
##
## data:  data
## X-squared = 7332.5, df = 2, p-value < 2.2e-16
```

```
shapiro.test( as.vector(data) ) # Shapiro-Wilk test
```

```
##
##  Shapiro-Wilk normality test
##
## data:  as.vector(data)
## W = 0.91981, p-value < 2.2e-16
```

Here, all give p-value≈0 for S&P500 returns.

# 1.5.Estimating the tail index

## 1.5.1. Modeling Tail Behavior (Pareto MLE & Q-Q plots)

- Complementary CDF of heavy tail distribution behaves as:

$$\bar{F}(x) = 1 - F(x) = P(X > x) \sim x^{-\alpha}, \text{ as } x \uparrow$$

- Model (absolute) returns above cutoff $r_{\min}$ using Pareto distribution

$$\bar{F}(r) = \left(\frac{r}{r_{\min}}\right)^{-\alpha}, \quad \forall r > r_{\min}$$

### 1.5.1.0.Preprocess Settings (tail parts data prep)

This two method (Pareto MLE & Q-Q plots) needs tail data.

```
## INPUT
data = SP.r # log return
tail_cut_quantile = 0.75

## Get tail parts data
(cut.off=quantile(abs(data),tail_cut_quantile)) # cutoff point
```

```
##        75%
## 0.01210769
```

```
tail_data=abs(as.numeric(data[as.logical(abs(data)>cut.off)])) # values above cutoff -> tail data
n=length(tail_data)
```

Now, we will estimate tail index (a) using:

#### Method1. Maximum Likelihood

Recall Pareto distribution pdf:

$$f(r) = \frac{\alpha r_{\min}^{\alpha}}{r^{\alpha+1}}, \quad r \geq r_{\min} = l \implies \text{MLE}(\alpha) = \left[\frac{1}{n}\sum_{i=1}^{n} \ln\left(\frac{r_i}{r_{\min}}\right)\right]^{-1}$$

We will estimate by:

$$\hat{\alpha} = \frac{n}{\sum_{i=1}^{n} \ln\left(\frac{r_i}{r_{\min}}\right)}$$

```
## Method 1: Pareto MLE
(alpha.MLE=n/sum(log(tail_data/cut.off))) # Pareto MLE
```

```
## [1] 1.96558
```

```
print(paste("alpha from MLE:",alpha.MLE))
```

```
## [1] "alpha from MLE: 1.96558025207164"
```

#### Method2. Pareto Q-Q plots

- Plot empirical CDF vs returns in log-log-scale

$$\text{Estimate } \alpha \text{ using slope of best fitting line (simple linear regression) , and } \hat{\alpha} = -slope$$

- Detailed Explain:

$$\bar{F}(r) = \left(\frac{r}{r_{\min}}\right)^{-\alpha} <=> \log \bar{F}(r) = -\alpha \log\left(\frac{r}{r_{\min}}\right) = -\alpha \log r + \alpha \log r_{\min}$$

$$Y = \log \bar{F}(r); \; X = log(r)$$

$$fit\ X, Y \text{ use SLR(Simple Linear Regression)} => \text{ slope } = -\hat{\alpha} <=> \hat{\alpha} = -slope$$

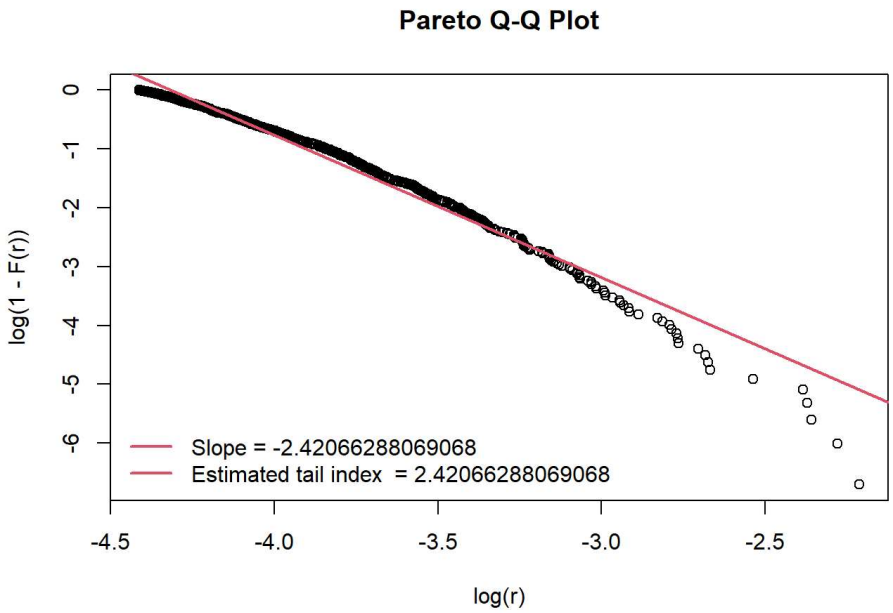
```
## Method 2: Pareto Q-Q plots
### Convert to log-log scale
X=log(sort(tail_data)) # log(r) - quantiles
Y=log( 1- (1:n - 1)/n ) # log(1 - F(r))- empirical complementary CDF
plot(X, Y, xlab="log(r)", ylab="log(1 - F(r))", main="Pareto Q-Q Plot"); abline(lm(Y~X), col=2, lwd=2)

### Estimate α using slope of best fitting line (simple linear regression)
fit <- lm(Y ~ X); slope <- coef(fit)[2]; alpha.QQ <- -slope


# Print values
print(paste("Slope of best-fit line:", round(slope, 4)))
```

```
## [1] "Slope of best-fit line: -2.4207"
```

```
print(paste("Tail index alpha from Q-Q plot:", round(alpha.QQ, 4)))
```

```
## [1] "Tail index alpha from Q-Q plot: 2.4207"
```

```
legend("bottomleft",col=2, lwd=2, bty="n",
       legend = c(paste("Slope =", slope),
                  paste("Estimated tail index  =",alpha.QQ)))
```




## 1.5.2.Modeling Heavy Tail Distributions (Univariate Student's t estimation)

- Student's t offers tractable heavy-tail model of **entire** return distribution (not just tail)
  - Typically adjust for location & scale as: (where Tail Index of Student's t = degree of freedom. )

$$Y = \mu + \sigma X,\ where\ X \sim t(df = v = \hat{\alpha}), \qquad Recall: E[X] = 0\ , \forall \nu > 1;\ V[X] = \frac{\nu}{\nu - 2}\ , \forall \nu > 2$$

  - Estimate $(\mu, \sigma, v)$ using Maximum Likelihood

```
## INPUT
data = SP.r #Log return

## Univariate Student's t estimation
library(MASS)
fit=fitdistr(data,'t') # Fits a Student's t-distribution
(alpha.t= (fit$estimate)['df']) # df is tail index estimate for t dist.
```

```
##       df
## 2.808734
```

```
print(paste("alpha from t:",alpha.t))
```

```
## [1] "alpha from t: 2.80873419156205"
```

```
## INPUT
data = SP.r #Log return

library(MASS)
(t_est = fitdistr(data, "t")$estimate)
```

```
##           m           s          df
## 0.000359427 0.008191191 2.808734192
```

```
# estimated parameters are: (mu, sigma, v),
# from Normal-chi mixture model: N(mu, sigma) * sqrt( v / chi2(v) )
mu = t_est[1]; sigma = t_est[2]; v = t_est[3]

# t mean & variance
mean.t = mu; var.t = sigma^2 * ( v/(v-2) )

# the sample moments are
mean.sample = mean(data) ; std.sample = sqrt(var(data)) ;var.sample = var(data)

print(paste("fitted t distribution mean:",mean.t,"std:",sigma, "variance:",var.t))
```

```
## [1] "fitted t distribution mean: 0.000359426961240315 std: 0.00819119138841417 variance: 0.000233023104185626"
```

```
print(paste("sample mean:",mean.sample,"std:",std.sample,"variance:",var.sample))
```

```
## [1] "sample mean: -1.13102501426956e-05 std: 0.0135069485752257 variance: 0.000182437659813793"
```

- Other modeling options include:
  - Discrete/continuous mixture models
  - GARCH models

# 1.6.Mixture models

RV generated from one out of a family of distributions, chosen at random according to another distribution (a.k.a. mixing distr.), resulting RV's distribution called a *mixture.* Mixtures used for modeling complicated distributions based on simple ones (Very easy to *generate* RVs, but harder to work with them *analytically ).*

## 1.6.1.Discrete Mixture

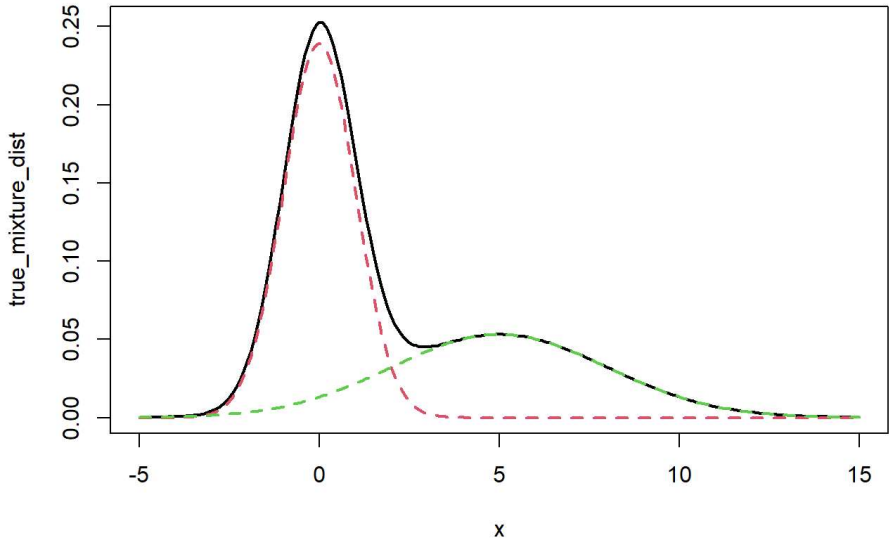
Select one out of a discrete (finite or countable) family of distributions

E.g. Generate RV from:

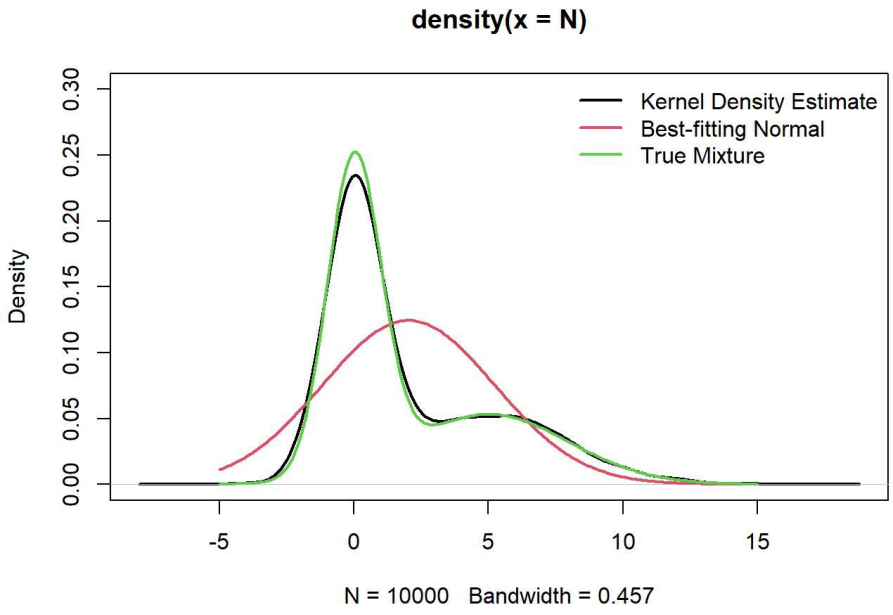
$$\begin{cases} N(0,1), & \text{with prob. } 60\% \\ N(5,3), & \text{with prob. } 40\% \end{cases}$$

Mixture Normal ; `dnorm(mean, sd=standard deviation)`

```
# Discrete mixture: 0.6*N(0,1) + 0.4*N(5,9=std^2)
x=seq(-5,15,by=.1); true_mixture_dist = .6*dnorm(x,0,1)+.4*dnorm(x,5,3)

plot( x, true_mixture_dist, type='l', lwd=2)
lines(x, .6*dnorm(x,0,1), lwd=2, col=2, lty=2)
lines(x, .4*dnorm(x,5,3), lwd=2, col=3, lty=2)
```




```
## Method1: generate combined sample
N1=rnorm(6000,0,1); N2=rnorm(4000,5,3)
N=sample(c(N1,N2)) # combined sample

## Method2: Use Bernoulli
N1=rnorm(10000,0,1); N2=rnorm(10000,5,3); Y=rbinom(10000,1,0.4)
N=Y*N2+(1-Y)*N1

x=seq(-5,15,by=.1); true_mixture_dist=.6*dnorm(x,0,1)+.4*dnorm(x,5,3)
plot(density(N),col=1,lwd=2,ylim=c(0,.3)) # kernel density estimator
lines( x, dnorm(x,mean(N),sd(N)), col=2, lwd=2) # best fitting Normal
lines( x, true_mixture_dist, col=3, lwd=2) # true mixture dist.
legend("topright",legend = c("Kernel Density Estimate", "Best-fitting Normal", "True Mixture"),
       col = c(1, 2, 3), lwd = 2, bty = "n")
```




## 1.6.2.Continuous mixture - t distribution (df=v)

Select one out of a continuous (possibly uncountable) family of distributions -aka. *compound* distribution.

- **Eg.1.Normal scale mixture:**

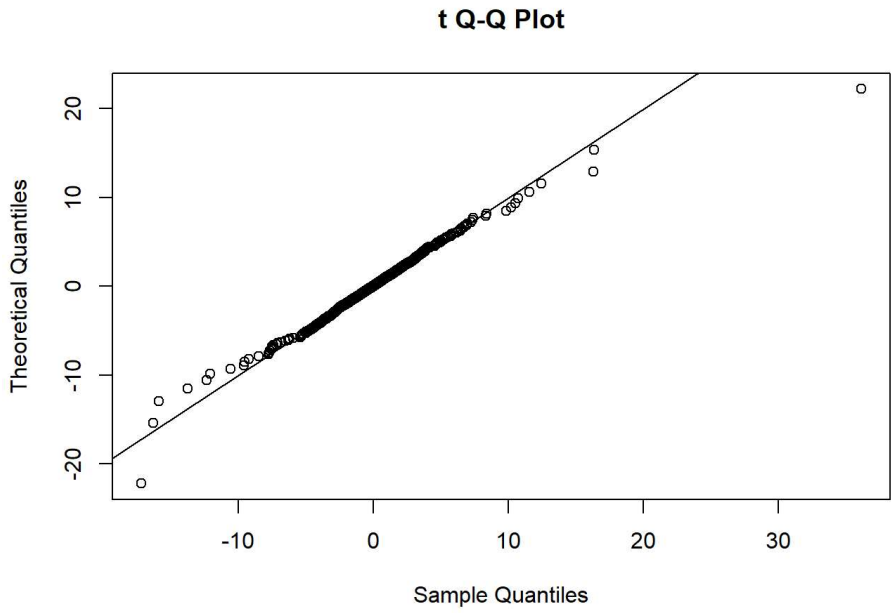
$$Y = \mu + \sqrt{V} \cdot Z, where\ V \text{ is RV with (non-negative) mixing distribution, representing 'random' StDev of} Y$$

- **Eg.2.t-distribution (with Code Example):**

$$t = Z\sqrt{\frac{\nu}{W}} \sim t(df = \nu),\ where\ Z \sim N(0,1), W \sim \chi^2(df = \nu) = Gamma(\frac{\nu}{2}, \frac{1}{2})\ => \frac{1}{W} \sim InvGamma,\ E(\frac{1}{W}) = \frac{1}{\nu - 2} => V(t) = \frac{\nu}{\nu - 2}$$

- `ppoints(n)` : generates `n` evenly spaced probabilities in the open interval (0,1), designed specifically for quantile plots like Q-Q plots.
- `qt(p, df)` : t-distribution quantile at prob `p`

```
# Continuous mixture
## INPUT
v=3; n=5000;

# Student's t (df=v) as a Normal scale mixture with inverse chi^2 mixing distr.
W=rchisq(n,v); Z=rnorm(n); Y=Z*sqrt(v/W)

qqplot( Y, qt( ppoints(n), v ), # Q-Q plot comparison with t(df=v)
        main="t Q-Q Plot", xlab="Sample Quantiles",
        ylab="Theoretical Quantiles"); abline(c(0,1))
```




- **Eg.3.GARCH(p,q) models:** $r_t = \mu + \sigma_t \cdot Z_t$
  - Where mixing process for $\sigma_t$ is:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i r_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

  - GARCH models also lead to heavy tails. (GARCH) Generalized AutoRegressive Conditional Heteroskedasticity.

# 2.Investment Returns Simulation (Comparing Normal, Student's t, and GARCH Models )

**Comparing Normal, Student's t, and GARCH Models** for Investment Returns Simulation

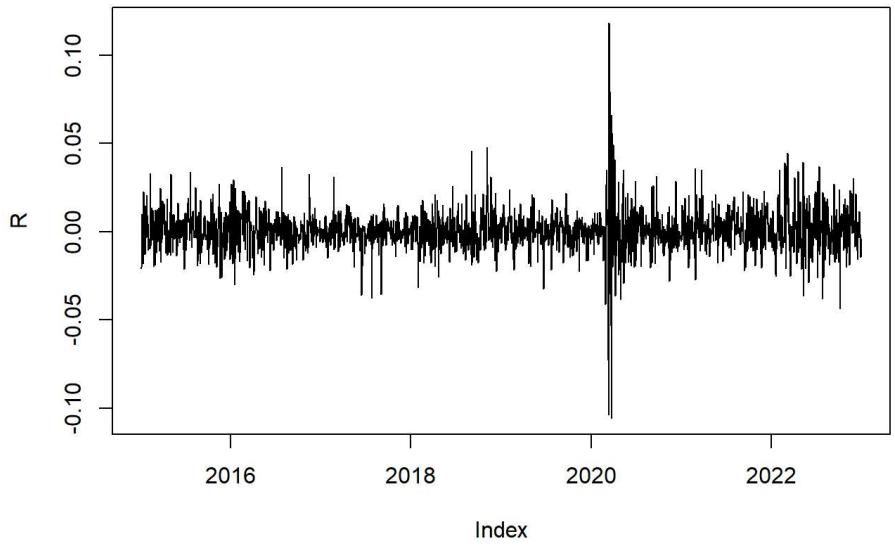
# 2.1.Data Prep

Download from Yahoo! Finance the daily adjusted closing prices of Loblaw Companies Limited (L.TO), from Jan-1-2015 to Dec-31-2022

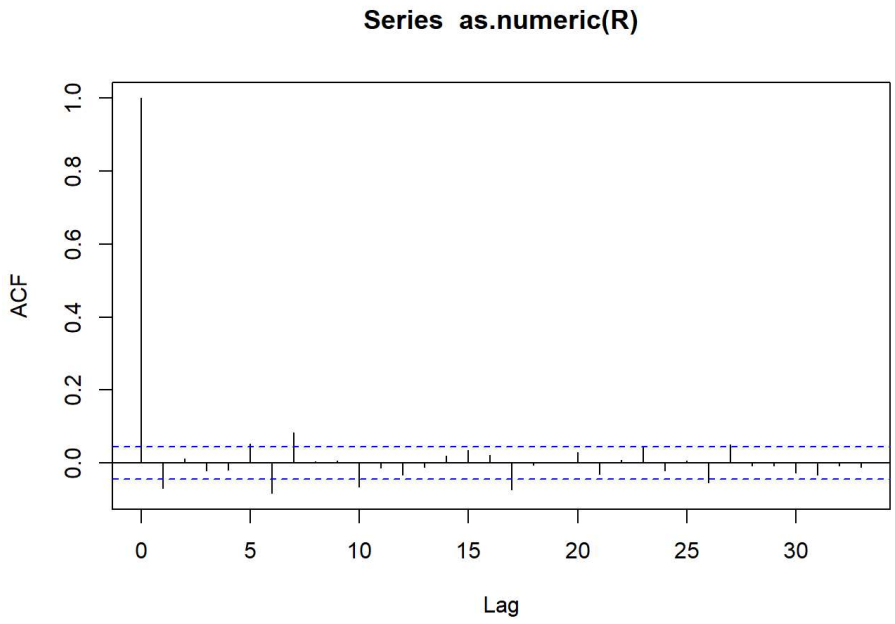
## 2.1.1.Returns & Descriptive Stats

Calculate the daily net returns, and plot their time-series plot, their sample auto-correlation plot, and their Normal QQ plot. Which of the return stylized facts can you observe?

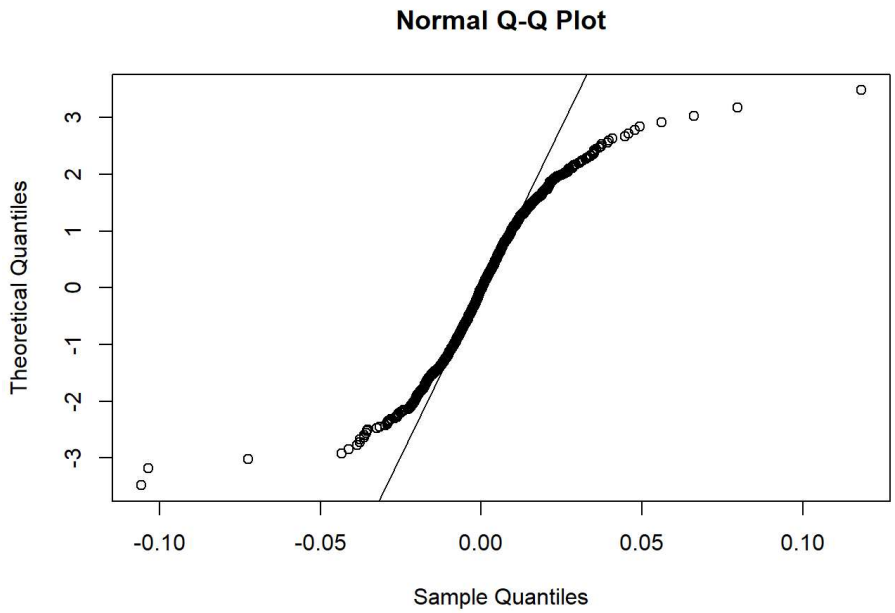
```
R = quantmod::Delt(S)[-1] # Remove first NA return
plot(R)
```




```
acf( as.numeric(R) )
```




```
qqnorm(R, datax = TRUE); qqline(R, datax = TRUE)
```




The returns exhibits volatility clustering (time series plot), are uncorrelated (ACF plot), and have much heavier tails than Normal (Normal QQ-plot).

# 2.2. Parameters

Fit a univariate t-distribution to the returns and report the resulting values (use the fitdistr function from the MASS library). Find the mean and variance of the fitted t-distribution, and compare them to the sample mean and variance of the returns

```
library(MASS)
(t_est = fitdistr(R, "t")$estimate)
```

```
##            m            s           df
## 0.0003224132 0.0076221453 3.3165545099
```

```
# estimated parameters are: (mu, sigma, v),
# from Normal-chi mixture model: N(mu, sigma) * sqrt( v / chi2(v) )
mu = t_est[1]; sigma = t_est[2]; v = t_est[3]

# t mean & variance
mean.t = mu; var.t = sigma^2 * ( v/(v-2) )

# the sample moments are
mean.norm = mean(R); var.norm = var(R)

print(paste("fitted t distribution mean:",mean.t,"variance:",var.t))
```

```
## [1] "fitted t distribution mean: 0.000322413153917257 variance: 0.000146353376582199"
```

```
print(paste("sample mean:",mean.norm,"variance:",var.norm))
```

```
## [1] "sample mean: 0.000566153188162708 variance: 0.000138272674515105"
```

The variance of the fitted t distribution (1.4635338^{-4}) is close to the sample variance (1.3827267^{-4}), while the mean of the t distribution (3.2241315^{-4}) is lower than the sample mean (5.6615319^{-4}). Differences can occur in the parameters because the t distribution can absorb extreme values.

# 2.3. Simulation

We will now compare 2 different return distribution approaches in an practical investment setting. Assume you invest all of your wealth in L.TO for 4 years (4 × 252 = 1008 days). Simulate 5000 iterations of 1008 daily returns from the following models:

```
n = 5000; m = 1008; set.seed(123)
```

- i. Returns are i.i.d. **Normal** with parameters equal to the sample mean and variance (i.e., the Normal MLE estimates).

```
### i
# create n x m matrix of simulated paths (where each row = a path)
R.norm = matrix( rnorm( n*m, mean.norm, sqrt(var.norm) ), nrow = n, ncol = m )

# apply cumulative product to each row, to get the cumulative net return path
Rcm.norm = t( apply( 1+R.norm, MARGIN = 1, FUN = cumprod) - 1 )

# final returns are the last column/value in each path
Rfn.norm = Rcm.norm[,m]

# calculate maximum drawdown for each path
mdd = function(R){ return(tseries::maxdrawdown(R)$maxdrawdown) }
MDD.norm = apply(Rcm.norm, MARGIN = 1, FUN = mdd )
```

- ii. Returns are i.i.d. **t** with the parameters (mean, var, df) of fitted t-distribution

```
### ii
# similarly for t distribution
R.t = matrix( rnorm( n*m, mu, sigma ) * sqrt( v / rchisq( n*m , df = v) ), nrow = n, ncol = m )

# apply cumulative product to each row, to get the cumulative net return path
Rcm.t = t( apply( 1+R.t, MARGIN = 1, FUN = cumprod) - 1 )

# final returns are the last column/value in each path
Rfn.t = Rcm.t[,m]

# calculate maximum drawdown for each path
mdd = function(R){ return(tseries::maxdrawdown(R)$maxdrawdown) }
MDD.t = apply(Rcm.t, MARGIN = 1, FUN = mdd )
```

- iii. (Optional) Returns follow a **GARCH(1,1)** model: use the `garchFit()` function to fit the model and the garchSim function to simulate from it (both in library `fgarch`).

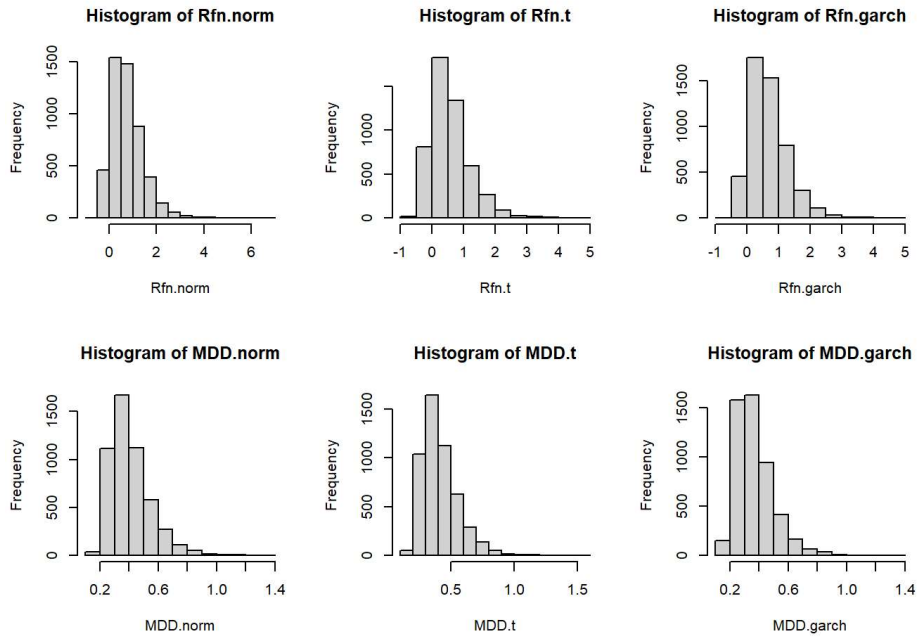
```
### iii
library(fGarch)
```

```
## NOTE: Packages 'fBasics', 'timeDate', and 'timeSeries' are no longer
## attached to the search() path when 'fGarch' is attached.
##
## If needed attach them yourself in your R script by e.g.,
##         require("timeSeries")
```

```
GARCH.fit=garchFit(~garch(1,1), data=R, trace = FALSE) # fit GARCH(1,1) model
GARCH.param=GARCH.fit@fit$coef # fitted coefficients
GARCH.spec=garchSpec(model=list(mu=GARCH.param['mu'], # define GARCH model specification for simulation
                                omega=GARCH.param['omega'],
                                alpha=GARCH.param['alpha1'],
                                beta=GARCH.param['beta1'] ))
R.garch = matrix( 0, nrow = n, ncol =m )
for(i in 1:n){ R.garch[i,]=as.numeric(garchSim(GARCH.spec,m))}

# apply cumulative product to each row, to get the cumulative net return path
Rcm.garch = t( apply( 1+R.garch, MARGIN = 1, FUN = cumprod) - 1 )

# final returns are the last column/value in each path
Rfn.garch = Rcm.garch[,m]

# calculate maximum drawdown for each path
mdd = function(R){ return(tseries::maxdrawdown(R)$maxdrawdown) }
MDD.garch = apply(Rcm.garch, MARGIN = 1, FUN = mdd )
```

Calculate the final wealth and the maximum drawdown of each iteration, and create a histograms of the two metrics from each model (2 × 3 = 6 histograms in total). What do you observe?

```
par(mfrow = c(2,3))
hist(Rfn.norm); hist(Rfn.t); hist(Rfn.garch)
hist(MDD.norm); hist(MDD.t); hist(MDD.garch)
```




We observe: The final returns for t-distr. and GARCH gave lower negative values, but are comparable to those of Normal. In terms of maximum drawdown, the GARCH values have a smaller range, but are otherwise also comparable. Not that for for both metrics we combine/compound many returns, so the effects of heavy tails is suppressed.

# 3. heavy tailed distributions (Cauchy, cumulative sample average)

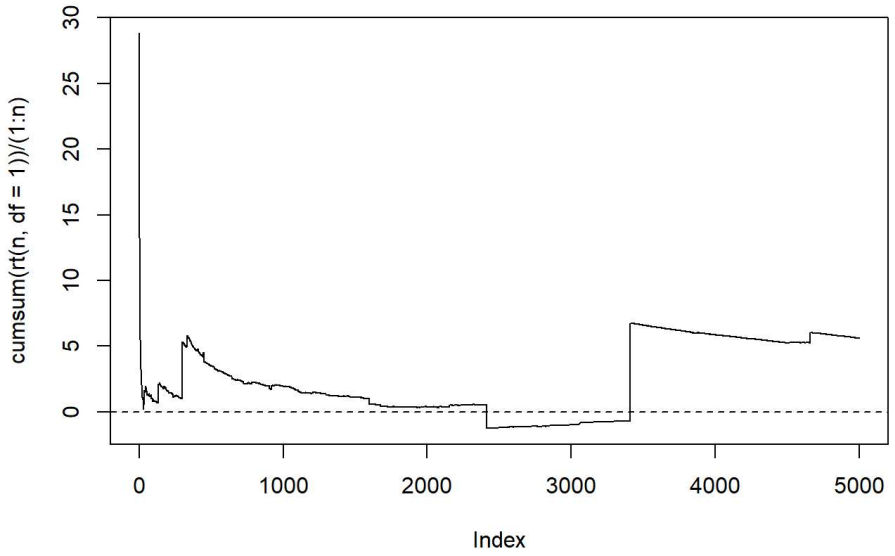
We will verify empirically (using simulation) some basic results from heavy tailed distributions.

## 3.1.WLLN not applied

Generate $n = 5000$ i.i.d. Cauchy variates, and calculate the cumulative sample average $\bar{X}_k = \frac{1}{k}\sum_{i=1}^{k} X_i$ versus $k$, for $k = 1, \ldots, n$. If the WLLN held for this distribution, you would expect to see the average converge to 0. Does the plot behave like that?

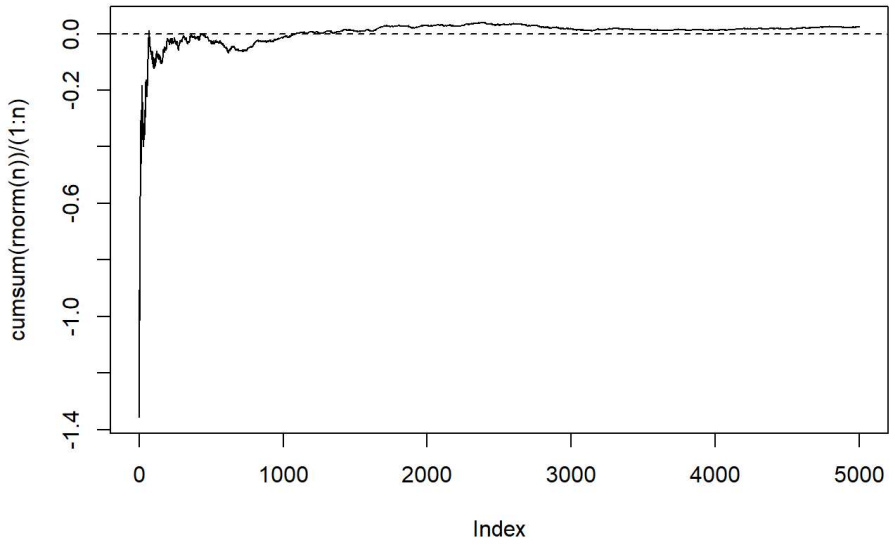
- Cauchy distribution is the same as a $t$ with 1 degree of freedom, use this fact to generate Cauchy variates using the function `rt(n,df=1)` .
- `cumsum(...)` : Takes the **cumulative sum** of that sequence. So, for example, if the output of `rt(n, 1)` is `[x₁, x₂, x₃, ...]` , it returns `[x₁, x₁+x₂, x₁+x₂+x₃, ...]`

```
set.seed(1234567890)
n=5000
plot( cumsum( rt(n , df = 1) ) / (1:n), type = "l"); abline(0,0, lty =2)
```




The sample average of the Cauchy distribution does not converge to a its mean (0), as there will always be large shifts caused by extreme values. You can compare this with the plot of the average of i.i.d. Normals, for which the WLLN applies, and which shows the expected convergence.

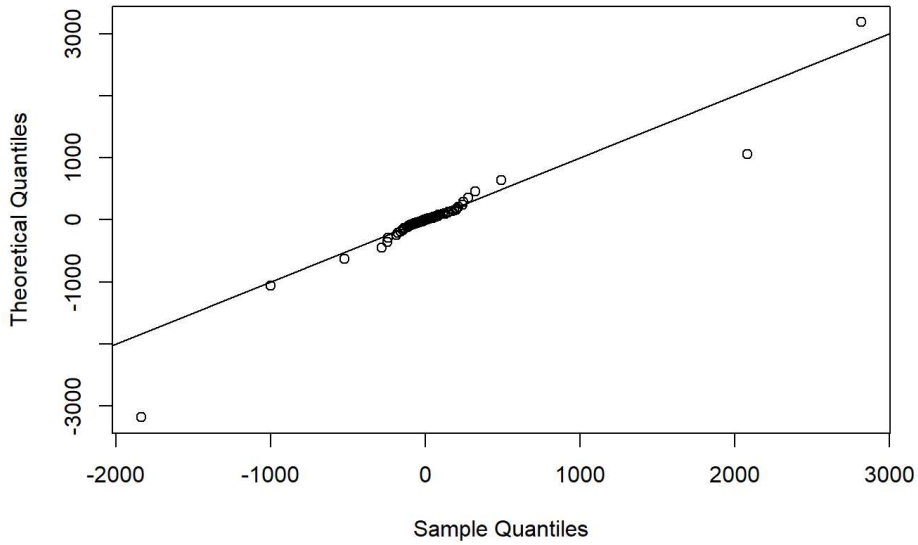
```
plot( cumsum( rnorm(n) )/(1:n), type = "l"); abline(0,0, lty =2)
```




# 3.2.Stable Distribution

- Generate $n = 5000$ pairs $(X_i, Y_i)$ of independent Cauchy variates, and calculate their mean $W_i = (X_i + Y_i)/2$. The mean should also be Cauchy distributed (Cauchy is a stable distribution). Create a QQ-plot comparing the sample quantiles from your simulation to the theoretical quantiles from a Cauchy distribution; does the plot confirm the theoretical result?
- Use function `qt(points(n), df = 1)` to find the theoretical quantiles of the Cauchy distribution.)

```
n=5000
X = rt(n, df=1); Y = rt(n, df=1); W = (X + Y)/2
x = sort(W); y = qt(ppoints(n), df = 1)
plot( x, y , xlab= "Sample Quantiles", ylab="Theoretical Quantiles"); abline(0,1)
```




The QQ plot shows good agreement of the Cauchy average quantiles vs the theoretical quantiles of the Cauchy distribution, as expected by the theoretical result. (The deviations at the very ends of the tails, were there are fewer values, are expected to some extent for extreme heavy-tail distributions like the Cauchy).

# 4.Extreme value theory (1st)

## 4.0.Thm and Settings

### 1st Extreme Value Theorem (Fisher–Tippett–Gnedenko)

- Let $X_1, X_2, \ldots$, be i.i.d. RVs and $M_n = \max(X_1, \ldots, X_n)$. In certain cases, there exist normalizing constants $a_n > 0$, $b_n$ such that:

$$P\left(\frac{M_n - b_n}{a_n} \leq x\right) = [F(a_n x + b_n)]^n \to H(x)$$

- Thm : If distribution H(x) exists, it must be one of three types. Type of EVT distribution depends on $X_1, X_2, \ldots$ tail behavior:

$$\text{Exponential Tails -> (Gumbel)} \;\; H(x) = \exp\{-e^{-x}\},\ x \in \mathbb{R}$$

$$\text{Heavy Tails -> (Frechet)} \;\; H(x) = \begin{cases} 0 & x < 0 \\ \exp\{-x^{-\alpha}\} & x > 0,\ \alpha > 0 \end{cases}$$

$$\text{Light / Finite Tails -> (Weibull)} \;\; H(x) = \begin{cases} \exp\{-|x|^{\alpha}\} & x < 0,\ \alpha > 0 \\ 1 & x > 0 \end{cases}$$

### Simulation experiment

The 1st Extreme Value Theorem (Fisher-Tippett-Gnedenko) states that the *rescaled* maximum $\frac{M_n - b_n}{a_n}$ of $n$ i.i.d. RVs can only converge to one of three types of distributions. Repeat the following simulation experiment for each of the following three distributions and associated scaling parameters.

- Generate $m = 1000$ values of the maximum of $n = 100$ i.i.d. RVs, i.e., generate $M_n(j) = \max\{X_i(j), \ldots, X_n(j)\}$ for each iteration $j = 1, \ldots, m$. Then rescale the simulated maxima according to appropriate $a_n, b_n$, i.e., calculate $Y_j = \frac{M_n(j) - b_n}{a_n}$ for $j = 1, \ldots, m$.
- Plot the histogram of the rescaled maxima $Y_j$, overlaid with the density of the corresponding theoretical limiting distribution (Weibull, Gumbel, or Fréchet).

```
n=100; m=1000
```

## 4.1.Light/Finite Tails -> Weibull

$X_i \sim^{i.i.d.}$ **Uniform(0,1)** with $b_n = 1$ and $a_n = 1/n$; which converges to Weibull (with $\alpha = 1$).

$$\text{Light / Finite Tails -> (Weibull)} \;\; H(x) = \begin{cases} \exp\{-|x|^{\alpha}\} & x < 0,\ \alpha > 0 \\ 1 & x > 0 \end{cases}$$

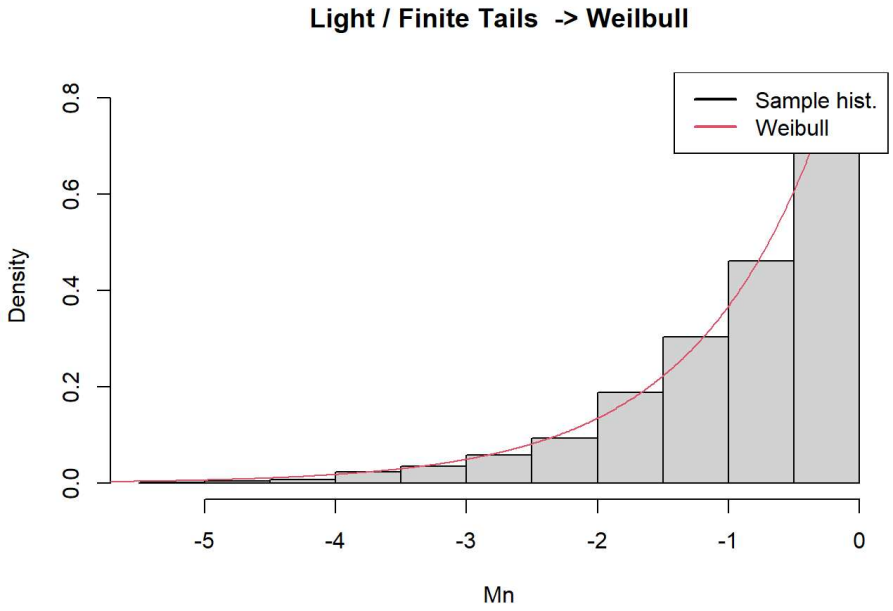
With shape parameter $k > 0$ and scale parameter $\theta > 0$, the PDF is:

$$f(x) = \begin{cases} \frac{k}{\theta}\left(\frac{x}{\theta}\right)^{k-1} \exp\left(-\left(\frac{x}{\theta}\right)^{k}\right), & x \geq 0 \\ 0, & x < 0 \end{cases} => if\ k = 1,\ f(x) = \frac{1}{\theta}e^{-x/\theta} <=> f(x) = \lambda e^{-\lambda x},\ \text{which is } exp(\lambda)$$

- `U = matrix(runif(m*n), m, n)` : Generates a matrix `U` with `m` rows and `n` columns filled with Uniform(0,1) values.; think as `m` independent experiments, each with `n` Uniform(0,1) samples.
- `apply(U, 1, max)` : For each row, take the maximum value. So you get a vector of `m` maxima, one from each row.
- `dexp(x)` :This gives the density of the Exponential(1) distribution at each value in $x$.

$$f(x) = \lambda e^{-\lambda x}, \quad \lambda = 1$$

```
### i
# Generate iid RVs in (m x n) matrix
U = matrix( runif(m*n), m, n)
# Calculate normalized maxima over rows
Mn = (apply(U, 1, max)-1)*n
# Plot histogram
hist(Mn, probability = T, main = "Light / Finite Tails  -> Weilbull")
# overlay theoretical distribution
x=seq(-10,0,.01); lines(x, dexp(-x), col = 2)
legend( "topright", lwd = 2, col = 1:2, c("Sample hist.","Weibull"))
```

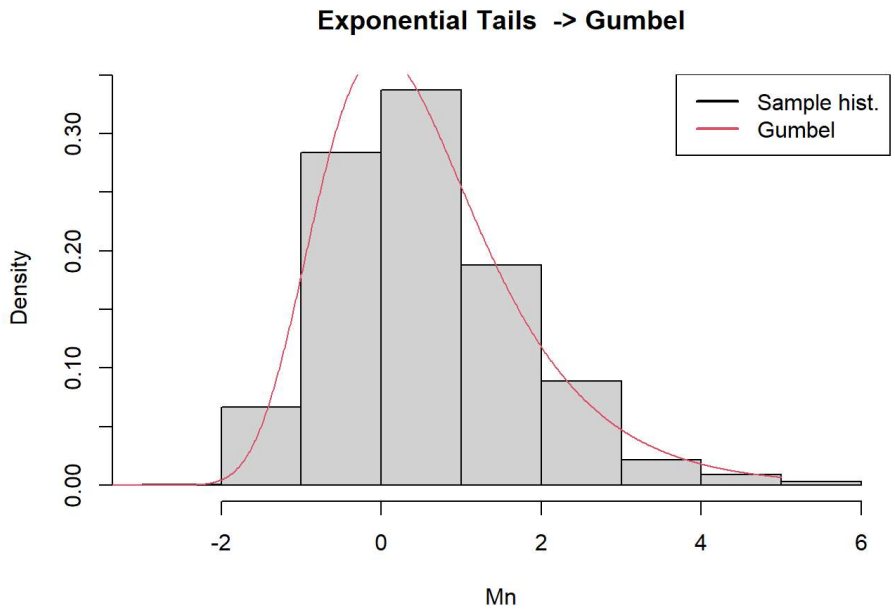



# 4.2.Exponential Tails -> Gumbel

$X_i \sim^{i.i.d.}$ **Normal(0,1)** with $b_n = \Phi^{-1}(1 - 1/n)$ and $a_n = \frac{1}{n\phi(b_n)}$; which converges to Gumbel. (Where $\Phi/\phi$ are the std Normal CDF/PDF.)

$$\text{Exponential Tails} \to (\text{Gumbel}) \quad H(x) = \exp\{-e^{-x}\},\ x \in \mathbb{R} => \text{pdf: } f(x) = e^{-x} \cdot \exp\left(-e^{-x}\right) = \exp\left(-x - e^{-x}\right), \quad x \in \mathbb{R}$$

```
### ii
Z = matrix( rnorm(m*n), m, n )
b_n = qnorm( 1 - 1/n)
a_n = 1 / (n * dnorm(b_n))
Mn = (apply(Z, 1, max) - b_n ) / (a_n)
hist(Mn, probability = T, main = "Exponential Tails  -> Gumbel")
x=seq(-5,5,.01); lines(x, exp( - x - exp(-x)), col = 2)
legend( "topright", lwd = 2, col = 1:2, c("Sample hist.","Gumbel"))
```




# 4.3.Heavy Tails -> Frechet

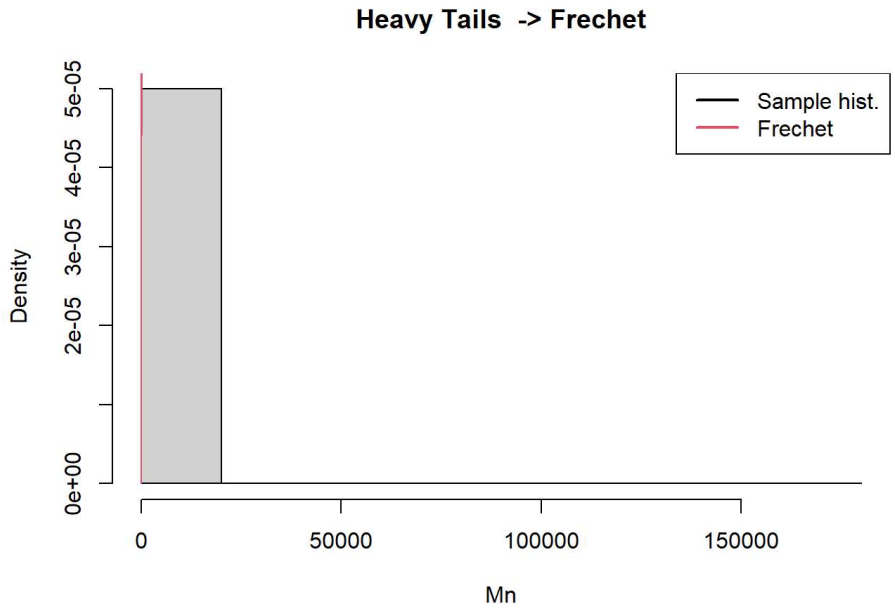
$X_i \sim^{i.i.d.}$ **Cauchy** with $b_n = 0$ and $a_n = n/\pi$ ; which converges to Fréchet (with $\alpha = 1$).

$$\text{Heavy Tails -> (Frechet) } H(x) = \begin{cases} 0 & x < 0 \\ \exp\{-x^{-\alpha}\} & x > 0,\ \alpha > 0 \end{cases}$$

The general pdf of the Fréchet distribution with shape parameter $\alpha > 0$ is:

$$f(x) = \begin{cases} \frac{\alpha}{x^{1+\alpha}} \exp\left(-\frac{1}{x^{\alpha}}\right), & x > 0 \\ 0, & x \leq 0 \end{cases} \Rightarrow if\ \alpha = 1\ ,\ f(x) = \frac{1}{x^2} \exp\left(-\frac{1}{x}\right)$$

```
### iii
X= matrix( rt(m*n,df=1), m, n )
Mn = (apply(X, 1, max) ) / (n/pi)
hist(Mn, probability = T, main = "Heavy Tails  -> Frechet" )
x=seq(0,150,.01); lines(x, 1/x^2 * exp(- 1/x), col = 2)
legend( "topright", lwd = 2, col = 1:2, c("Sample hist.","Frechet"))
```



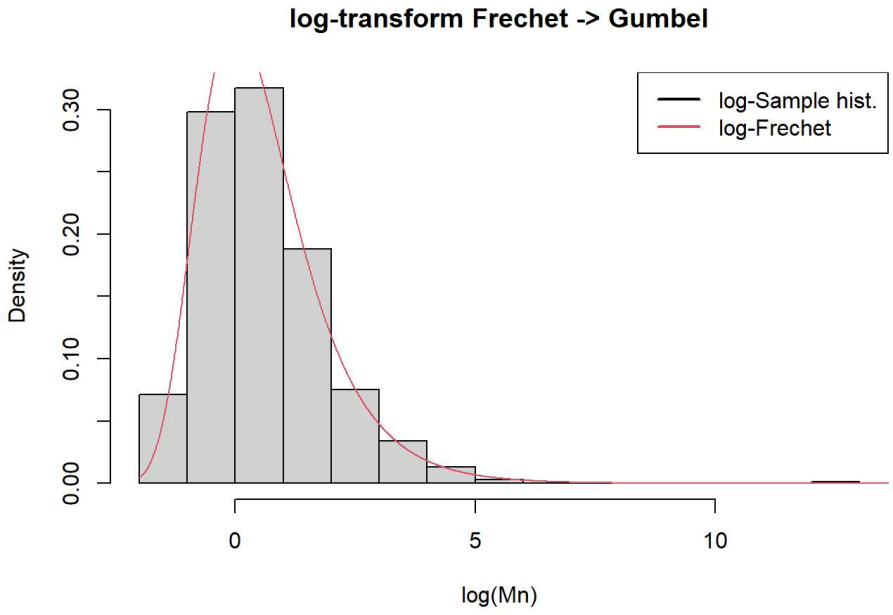
# 4.4.log-transformation of Frechet(a=1) is Gumbel

**Does the theorem seem to hold?**

In cases i. and ii. the convergence to the theoretical distribution is evident, but in iii. the histogram is not as informative because of the extreme values involved. For the last case, one can plot the histogram & density of the **log-transformed normalized maxima**, which actually show the convergence.

- Note: the **log-transformation of the Frechet(a=1) is the Gumbel distribution.**

```
hist( log(Mn), probability = T, main = "log-transform Frechet -> Gumbel" )
x=seq(-2,15,.01)
lines(x, exp( -x-exp(-x)), col = 2)
legend( "topright", lwd = 2, col = 1:2, c("log-Sample hist.","log-Frechet"))
```




**Why does this happen?**

The Fréchet distribution is one of the three types of Extreme Value Distributions for maxima, specifically for heavy-tailed variables. Recall Its CDF for shape parameter $\alpha > 0$ is:

$$F(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \exp(-x^{-\alpha}) & \text{if } x > 0 \end{cases} \; ; \; define \;\; Y = \log(X)$$

We want the distribution of $Y$. Use change of variables:

$$P(Y \leq y) = P(\log X \leq y) = P(X \leq e^y) = F(e^y) = \exp\left(-e^{-\alpha y}\right) => F_Y(y) = \exp\left(-e^{-\alpha y}\right)$$

Compare to the **Gumbel** CDF:

$$F(y) = \exp\left(-e^{-y}\right)$$

It is a **Gumbel distribution**, just scaled by $\alpha$. So:

- If $X \sim \text{Fréchet}(\alpha)$, then $\log X \sim \text{Gumbel}$ (scaled by $1/\alpha$)