Project: Financial Risk Analysis Based on Econometric Models

Tsung-Wei (Peter) Chen

12/02/2019

Introduction

Aims

The goal of project is to explore stock prices for Direxion Daily S&P500 Bull 3X Shares (SPXL) and Amazon.com, Inc. (AMZN) with specific techniques. Amazon.com, Inc., is an American multinational technology company based in Seattle that focuses on e-commerce, cloud computing, digital streaming, and artificial intelligence. According to recent significant holiday, Thanksgiving, in the US, Amazon promotes Black Friday deals for customers. Therefore, this is a great opportunity to explore the stock price of Amazon with daily S&P500.

Data

Use R to download the recent ten years stock prices for SPXL and AMZN from Yahoo Finance.

```
kable(head(SPXL, 5))
```

SPXL.Open	SPXL.High	SPXL.Low	SPXL.Close	SPXL.Volume	SPXL.Adjusted
3.09	3.38	3.02	3.33	87174000	2.69
3.29	3.43	3.22	3.33	111709200	2.69
3.42	3.51	3.32	3.41	119607600	2.75
3.23	3.28	3.05	3.12	116395200	2.52
3.05	3.15	3.01	3.15	104341200	2.55

```
kable(tail(SPXL, 5))
```

SPXL.Open	SPXL.High	SPXL.Low	SPXL.Close	SPXL.Volume	SPXL.Adjusted
59.9	60.7	59.9	60.7	2161900	60.7
60.8	61.2	60.5	61.1	1739900	61.1
61.5	61.9	61.3	61.9	1718800	61.9
61.6	61.7	61.1	61.2	1221600	61.2
61.4	61.4	59.4	59.6	4023600	59.6

```
kable(summary(SPXL))
```

Index	SPXL.Open	SPXL.High	SPXL.Low	SPXL.Close	SPXL.Volume	SPXL.Adjusted
Min. :2009-01-02	Min. : 1.2	Min. : 1.3	Min. : 1.2	Min. : 1.2	Min. : 386600	Min. : 1.0
1st Qu.:2011-09-22	1st Qu.: 6.2	1st Qu.: 6.3	1st Qu.: 6.1	1st Qu.: 6.2	1st Qu.: 3881750	1st Qu.: 5.9

Median :2014-06-18	Median :16.8	Median :17.2	Median :16.5	Median :16.8	Median : 6919800	Median :16.0
Mean :2014-06-17	Mean :20.2	Mean :20.4	Mean :19.9	Mean :20.2	Mean : 25193005	Mean :19.5
3rd Qu.:2017-03-10	3rd Qu.:31.2	3rd Qu.:31.6	3rd Qu.:30.7	3rd Qu.:31.1	3rd Qu.: 15466975	3rd Qu.:29.6
Max. :2019-12-02	Max. :61.6	Max. :61.9	Max. :61.3	Max. :61.9	Max. :447939600	Max. :61.9

```
kable(head(AMZN, 5))
```

AMZN.Open	AMZN.High	AMZN.Low	AMZN.Close	AMZN.Volume	AMZN.Adjusted
51.3	54.5	51.1	54.4	7296400	54.4
55.7	55.7	53.0	54.1	9509800	54.1
54.5	58.2	53.8	57.4	11080100	57.4
56.3	57.0	55.3	56.2	7942700	56.2
55.0	57.3	54.6	57.2	6577900	57.2

```
kable(tail(AMZN, 5))
```

AMZN.Open	AMZN.High	AMZN.Low	AMZN.Close	AMZN.Volume	AMZN.Adjusted
1753	1777	1753	1774	3486200	1774
1780	1797	1778	1797	3181200	1797
1801	1824	1797	1819	3025600	1819
1818	1825	1801	1801	1923400	1801
1804	1806	1763	1782	3925600	1782

```
kable(summary(AMZN))
```

Index	AMZN.Open	AMZN.High	AMZN.Low	AMZN.Close	AMZN.Volume	AMZN.Adjusted
Min. :2009-01-02	Min. : 49	Min. : 50	Min. : 48	Min. : 48	Min. : 984400	Min. : 48
1st Qu.:2011-09-22	1st Qu.: 192	1st Qu.: 195	1st Qu.: 190	1st Qu.: 192	1st Qu.: 2892475	1st Qu.: 192
Median :2014-06-18	Median : 331	Median : 334	Median : 326	Median : 331	Median : 4055150	Median : 331
Mean :2014-06-17	Mean : 614	Mean : 620	Mean : 607	Mean : 614	Mean : 4904139	Mean : 614
3rd Qu.:2017-03-10	3rd Qu.: 852	3rd Qu.: 855	3rd Qu.: 847	3rd Qu.: 852	3rd Qu.: 5862500	3rd Qu.: 852

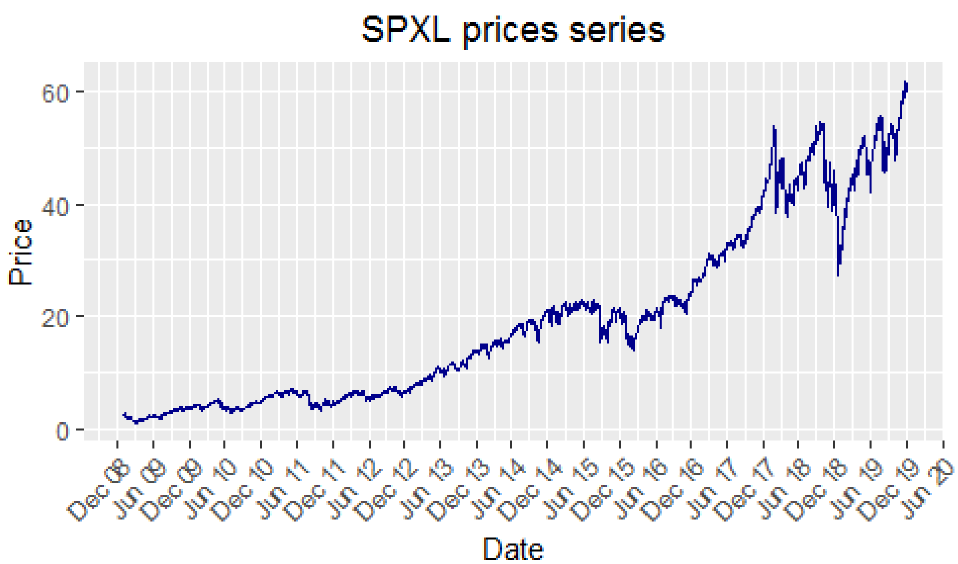
Max. :2019-12-02	Max. :2038	Max. :2050	Max. :2013	Max. :2040	Max. :58305800	Max. :2040

Explore Data Analysis

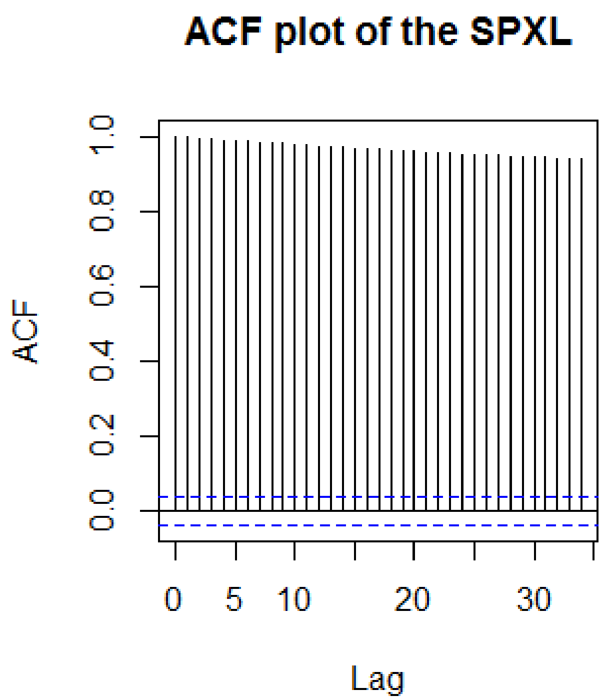
SPXL

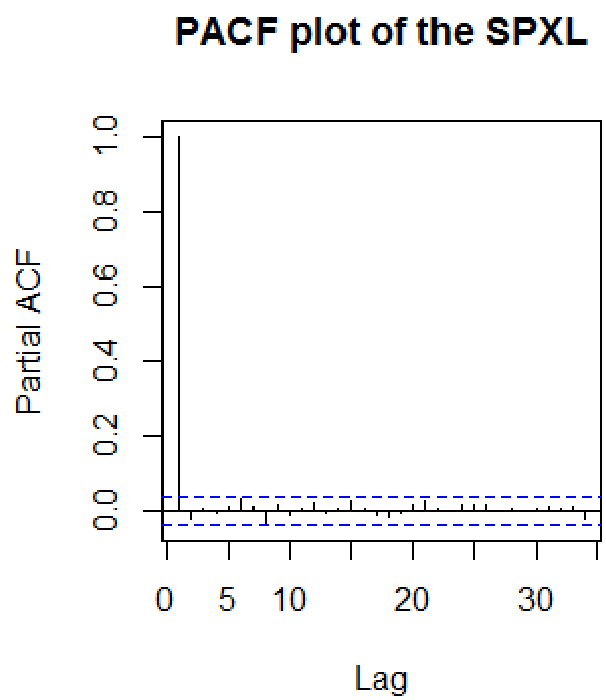
Plot daily prices, using the Adjusted Price column, since it incorporates events like splits and dividends distribution, which can affect the series. The time series plot appears in clusters, high in certain periods and low in certain periods.It evolves over time in a continuous manner and is thus, volatile. To attain stationarity, we find a fixed range in terms of log return of the stock prices. From the ACF plot, we observe that the plot decays to zero slowly. We can conclude that we need to perform time series analysis on the daily return (log return) of the stock prices.

```
ggplot(SPXL, aes(x = index(SPXL), y = SPXL[,6])) +
  geom_line(color = "darkblue") + ggtitle("SPXL prices series") +
  xlab("Date") + ylab("Price") + theme(plot.title = element_text(hjust = 0.5), axis.text.
x = element_text(angle = 45, hjust = 1)) +
  scale_x_date(date_labels = "%b %y", date_breaks = "6 months")
```



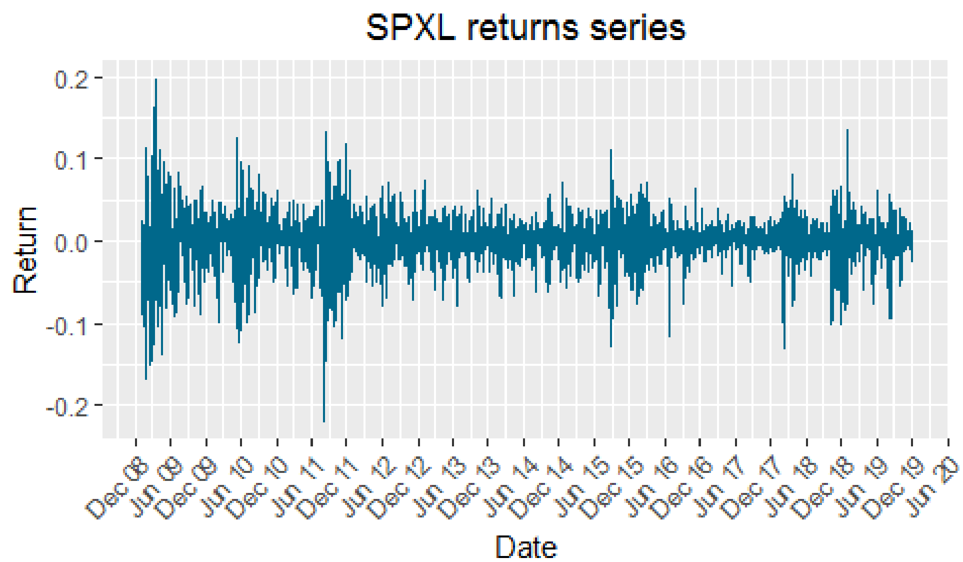

```
acf(SPXL$SPXL.Adjusted, main="ACF plot of the SPXL")
pacf(SPXL$SPXL.Adjusted, main="PACF plot of the SPXL")
```





From the basic statistics of the log return of the stock prices, we observe that the mean is approximately 0 and the distribution of log returns has large kurtosis(fat tails). We observe this further using histogram and Q-Q plot. The kurtosis is 5.16 which is larger than normal distribution, which kurtosis = 3. So the SPXL return has a heavier tail than normal. The skewness is -0.57, which means the distribution of return is asymmetric and the negative value implies that the distribution has a long left tail.

	SPXL.Adjusted
nobs	2747.000
NAs	0.000
Minimum	-0.220
Maximum	0.198
1. Quartile	-0.010
3. Quartile	0.016
Mean	0.001
Median	0.002
Sum	3.100
SE Mean	0.001
LCL Mean	0.000
UCL Mean	0.002
Variance	0.001
Stdev	0.031
Skewness	-0.568
Kurtosis	5.181

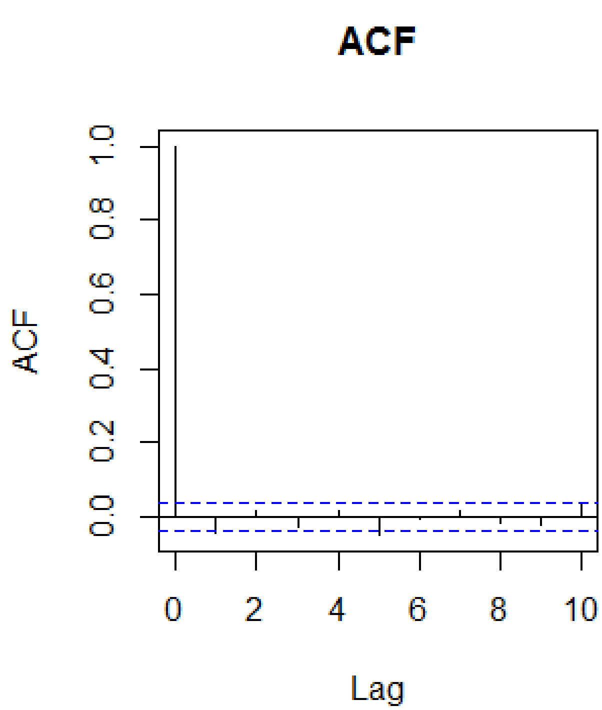


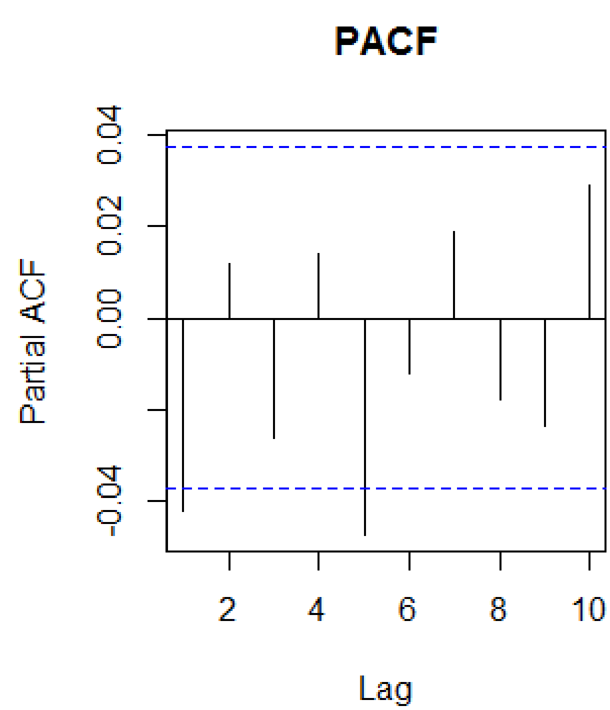
Test of independence. Compute the Ljung's Box Test on stock price returns.

```
Box.test(SPXL_rets^2, lag=2, type="Ljung")

##
##  Box-Ljung test
##
## data:  SPXL_rets^2
## X-squared = 300, df = 2, p-value <0.0000000000000002
```

ACF and PACF plot of the log return of the stock prices.

```
acf(SPXL_rets, lag=10, main="ACF", na.action = na.pass)
pacf(SPXL_rets, lag=10, main="PACF", na.action = na.pass)
```

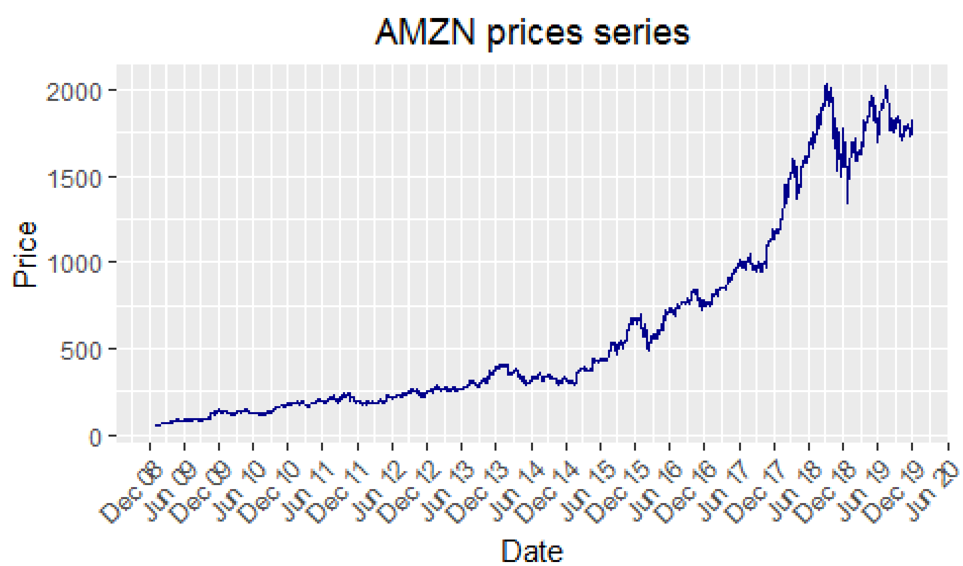




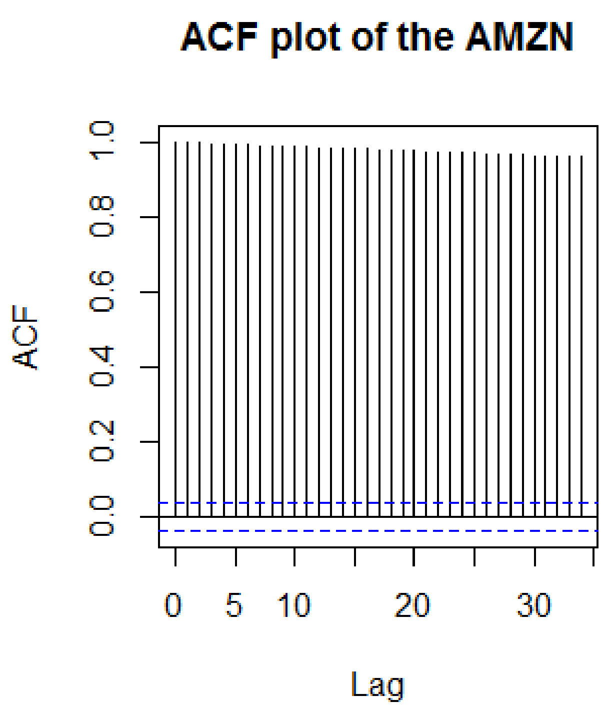
AMZN

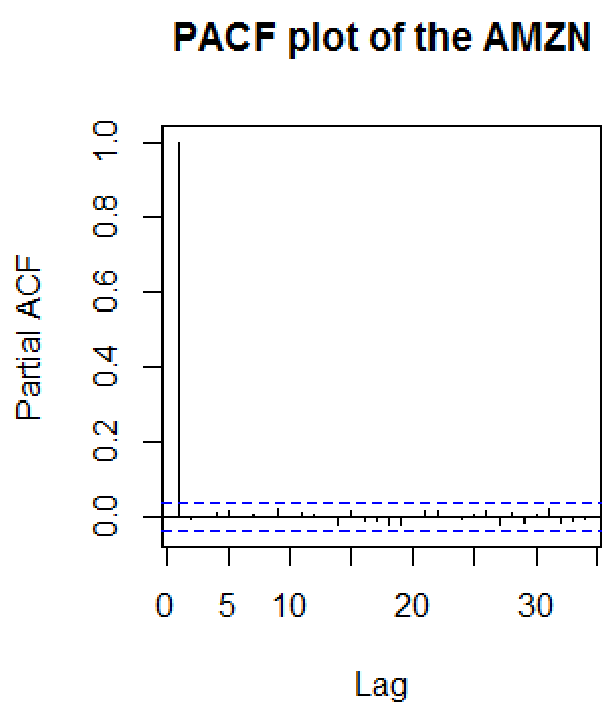
From the ACF plot, we observe that the plot decays to zero slowly. We can conclude that we need to perform time series analysis on the daily return (log return) of the stock prices.

```
ggplot(AMZN, aes(x = index(AMZN), y = AMZN[,6])) +
  geom_line(color = "darkblue") + ggtitle("AMZN prices series") +
  xlab("Date") + ylab("Price") + theme(plot.title = element_text(hjust = 0.5), axis.text.
x = element_text(angle = 45, hjust = 1)) +
  scale_x_date(date_labels = "%b %y", date_breaks = "6 months")
```

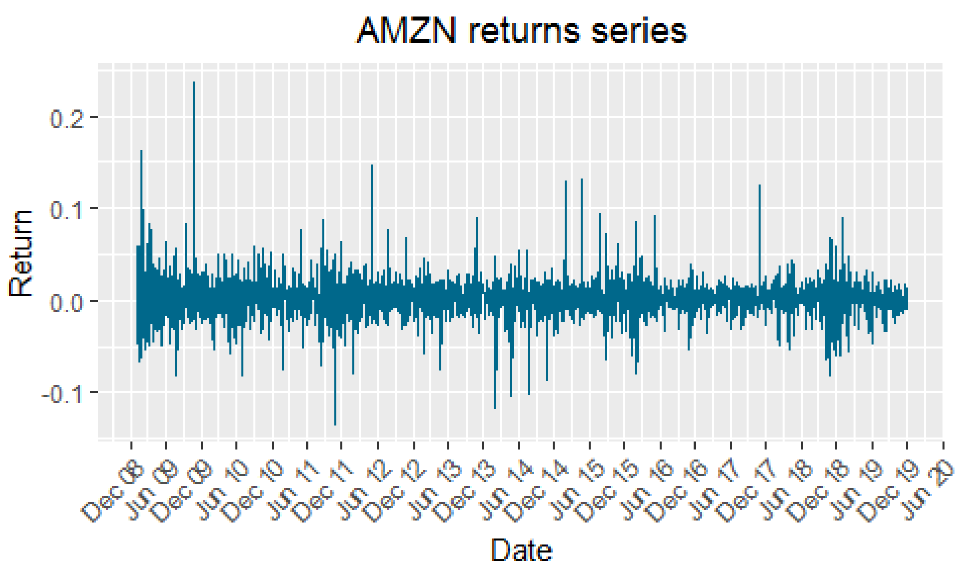



```
acf(AMZN$AMZN.Adjusted, main="ACF plot of the AMZN")
pacf(AMZN$AMZN.Adjusted, main="PACF plot of the AMZN")
```

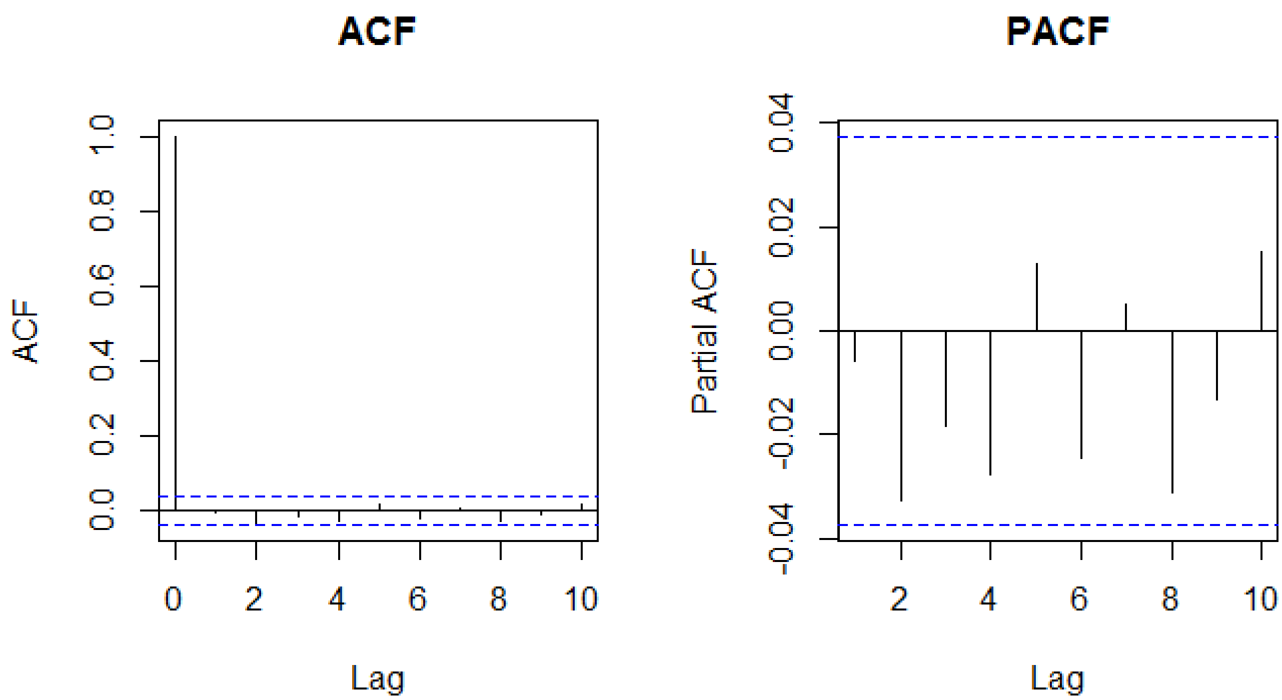




	AMZN.Adjusted
nobs	2747.000
NAs	0.000
Minimum	-0.135
Maximum	0.237
1. Quartile	-0.009
3. Quartile	0.012
Mean	0.001
Median	0.001
Sum	3.490
SE Mean	0.000
LCL Mean	0.000
UCL Mean	0.002
Variance	0.000
Stdev	0.021
Skewness	0.824
Kurtosis	12.122




```
Box.test(AMZN_rets^2, lag=2, type="Ljung")

##
##  Box-Ljung test
##
## data:  AMZN_rets^2
## X-squared = 10, df = 2, p-value = 0.003

acf(AMZN_rets, lag=10, main="ACF", na.action = na.pass)
pacf(AMZN_rets, lag=10, main="PACF", na.action = na.pass)
```



Build Time Series Models

Fit an AR(1)-GARCH(1,1) model to each series of log-returns:

SPXL

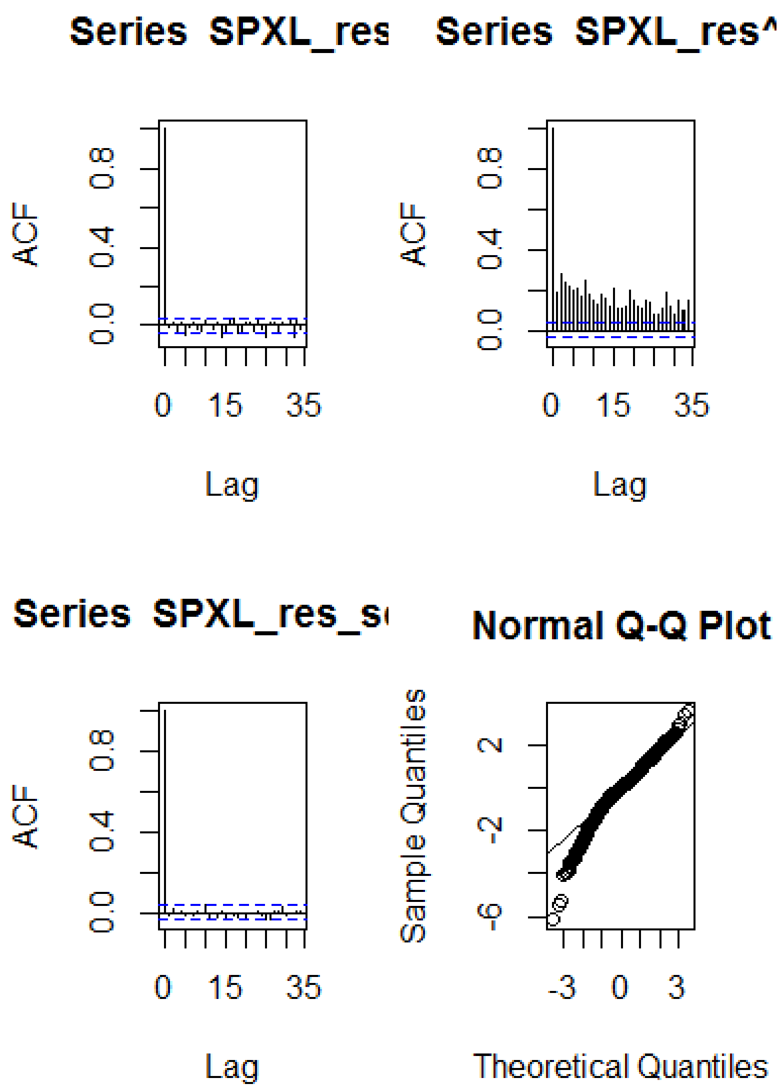
The residuas of SPXL ARMA model include significant correlation, which means not white noise. After getting residuals of ARMA-GARCH model, the acf plot shows white noise. Therefore, AR(1)-GARCH(1,1) IS GOOD. If not WN, it does not catch all of the dependence. Through qq-plot, it is normal distribution.

```
SPXLfit=garchFit(formula=~arma(1,0)+garch(1,1),data=SPXL_rets,cond.dist="norm")
```

```
##
## Series Initialization:
##   ARMA Model:                arma
##   Formula Mean:              ~ arma(1, 0)
##   GARCH Model:               garch
##   Formula Variance:          ~ garch(1, 1)
##   ARMA Order:                1 0
##   Max ARMA Order:            1
##   GARCH Order:               1 1
##   Max GARCH Order:           1
##   Maximum Order:             1
##   Conditional Dist:          norm
##   h.start:                   2
##   llh.start:                 1
##   Length of Series:          2747
##   Recursion Init:            mci
##   Series Scale:              0.0308
##
## Parameter Initialization:
```

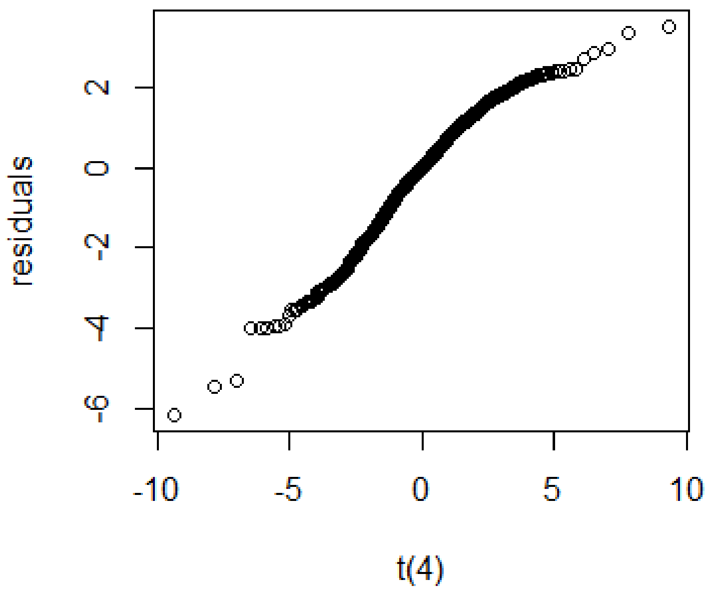
```
##  Initial Parameters:          $params
##  Limits of Transformations:   $U, $V
##  Which Parameters are Fixed?  $includes
##  Parameter Matrix:
##                      U       V  params includes
##     mu     -0.36688962   0.367  0.0367     TRUE
##     ar1    -0.99999999   1.000 -0.0422     TRUE
##     omega   0.00000100 100.000  0.1000     TRUE
##     alpha1  0.00000001   1.000  0.1000     TRUE
##     gamma1 -0.99999999   1.000  0.1000    FALSE
##     beta1   0.00000001   1.000  0.8000     TRUE
##     delta   0.00000000   2.000  2.0000    FALSE
##     skew    0.10000000  10.000  1.0000    FALSE
##     shape   1.00000000  10.000  4.0000    FALSE
##  Index List of Parameters to be Optimized:
##     mu    ar1  omega alpha1  beta1
##      1      2      3      4      6
##  Persistence:                  0.9
##
##
## --- START OF TRACE ---
## Selected Algorithm: nlminb
##
## R coded nlminb Solver:
##
##   0:     3492.1320: 0.0367106 -0.0422115 0.100000 0.100000 0.800000
##   1:     3443.7900: 0.0367125 -0.0417843 0.0746679 0.101228 0.787873
##   2:     3386.6988: 0.0367259 -0.0392697 0.0208096 0.171840 0.807797
##   3:     3380.2292: 0.0367272 -0.0391876 0.0285909 0.173141 0.810278
##   4:     3379.4794: 0.0368225 -0.0319473 0.0282894 0.171167 0.807988
##   5:     3379.3044: 0.0368349 -0.0316522 0.0302912 0.168230 0.806832
##   6:     3379.2224: 0.0368755 -0.0308688 0.0289561 0.165517 0.808619
##   7:     3379.1071: 0.0369233 -0.0295310 0.0292858 0.163440 0.811123
##   8:     3379.0541: 0.0370155 -0.0288609 0.0287081 0.161406 0.812835
##   9:     3379.0211: 0.0371494 -0.0296847 0.0289890 0.161233 0.813085
##  10:     3377.9816: 0.0459431 -0.0307067 0.0316395 0.178805 0.795908
##  11:     3377.0448: 0.0546310 -0.0606624 0.0260302 0.157453 0.820407
##  12:     3376.5837: 0.0568397 -0.0464255 0.0256673 0.153015 0.827358
##  13:     3376.3398: 0.0591266 -0.0450170 0.0241346 0.152161 0.825182
##  14:     3375.6132: 0.0614044 -0.0483759 0.0272860 0.154126 0.821240
##  15:     3374.7280: 0.0742584 -0.0434248 0.0274055 0.156176 0.817592
##  16:     3374.6972: 0.0747538 -0.0411332 0.0276252 0.156087 0.819357
##  17:     3374.6151: 0.0750962 -0.0429774 0.0294756 0.162186 0.811635
##  18:     3374.5565: 0.0755974 -0.0414119 0.0287622 0.163056 0.810971
##  19:     3374.5280: 0.0760996 -0.0398732 0.0289500 0.164017 0.810685
##  20:     3374.5145: 0.0766025 -0.0382395 0.0288157 0.164416 0.810491
##  21:     3374.5082: 0.0770369 -0.0361252 0.0286475 0.163250 0.811717
##  22:     3374.5078: 0.0770250 -0.0364174 0.0286406 0.163612 0.811437
##  23:     3374.5078: 0.0770159 -0.0364219 0.0286466 0.163595 0.811438
##  24:     3374.5078: 0.0770168 -0.0364208 0.0286460 0.163595 0.811439
##
## Final Estimate of the Negative LLH:
##  LLH:  -6190    norm LLH:  -2.25
##         mu        ar1      omega     alpha1      beta1
##  0.0023687 -0.0364208  0.0000271  0.1635952  0.8114394
```

```
##
## R-optimhess Difference Approximated Hessian Matrix:
##                mu      ar1         omega      alpha1       beta1
## mu       -6182006 -14975.5     -27243526     10314.4     -7577.4
## ar1        -14976  -2210.7        305533       -16.8        14.3
## omega   -27243526 305532.6 -228357703432 -52559514.0 -87868125.1
## alpha1      10314    -16.8     -52559514    -27484.4    -34366.4
## beta1       -7577     14.3     -87868125    -34366.4    -51138.7
## attr(,"time")
## Time difference of 0.064 secs
##
## --- END OF TRACE ---
##
##
## Time to Estimate Parameters:
##  Time difference of 0.207 secs

coef(SPXLfit)

##         mu        ar1      omega     alpha1      beta1
##  0.0023687 -0.0364208  0.0000271  0.1635952  0.8114394

predict(SPXLfit,n.ahead=10)

##    meanForecast meanError standardDeviation
## 1       0.00334    0.0193            0.0193
## 2       0.00225    0.0198            0.0198
## 3       0.00229    0.0202            0.0202
## 4       0.00229    0.0206            0.0206
## 5       0.00229    0.0210            0.0210
## 6       0.00229    0.0214            0.0214
## 7       0.00229    0.0218            0.0218
## 8       0.00229    0.0221            0.0221
## 9       0.00229    0.0225            0.0224
## 10      0.00229    0.0228            0.0228

SPXL_res=residuals(SPXLfit) # get residuals of ARMA model
SPXL_res_sd=residuals(SPXLfit,standardize=TRUE) # get residuals of ARMA-GARCH model

acf(SPXL_res) # white noise
acf(SPXL_res^2) # significant correlation => not white noise
acf(SPXL_res_sd) # white noise
acf(SPXL_res_sd^2) # white noise => AR(1)-GARCH(1,1) IS GOOD. If not WN, it does not catc
h all of the dependence.
qqnorm(SPXL_res_sd) # normal distribution
qqline(SPXL_res_sd)
```



Fit t distribution: not perform better than normal. Therefore, ignore fitting t distribution fo SPXL.

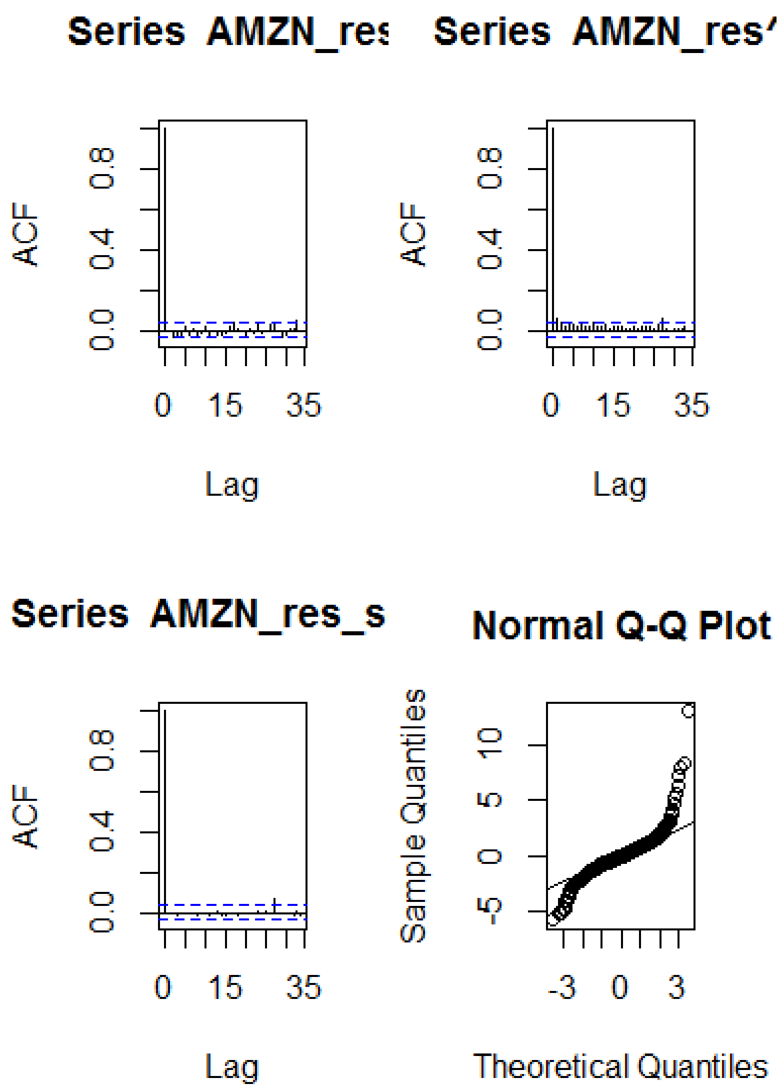
```
n=length(SPXL_res_sd)
x=qt((1:n)/(n+1),df=4)
qqplot(x,sort(SPXL_res_sd),xlab="t(4)",ylab="residuals") # worse than normal fitting
#SPXL_fit1=garchFit(formula=~arma(1,0)+garch(1,1),data=SPXL_rets,cond.dist="std")
#coef(SPXL_fit1)
```



AMZN

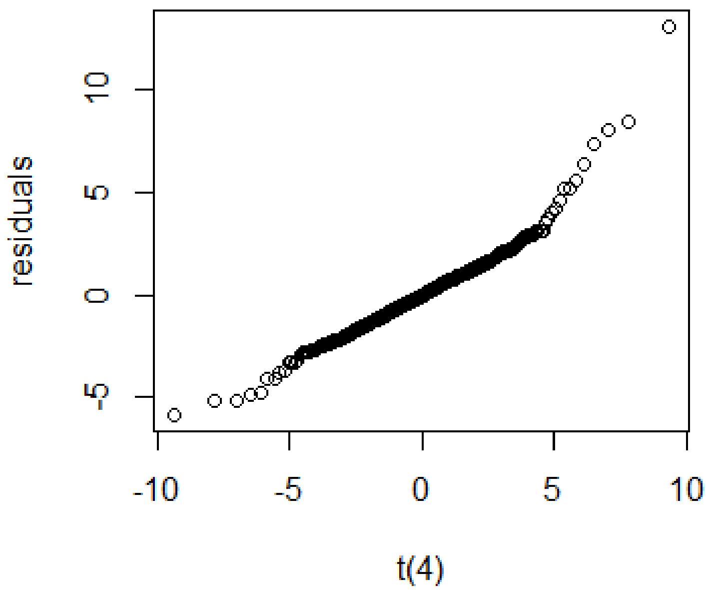
The residuas of AMZN ARMA model include significant correlation, which means not white noise. After getting residuals of ARMA-GARCH model, the acf plot shows white noise. Therefore, AR(1)-GARCH(1,1) IS GOOD. If not WN, it does not catch all of the dependence. Through qq-plot, it is not normal distribution. Fitting t distribution is needed.

```
acf(AMZN_res) # white noise
acf(AMZN_res^2) # significant correlation => not white noise
acf(AMZN_res_sd) # white noise
acf(AMZN_res_sd^2) # white noise => AR(1)-GARCH(1,1) IS GOOD. If not WN, it does not catc
h all of the dependence.
qqnorm(AMZN_res_sd) # not normal distribution
qqline(AMZN_res_sd)
```



Fit t distribution: perform better than normal. Therefore, build model with t distribution on AMZN.

```
n=length(AMZN_res_sd)
x=qt((1:n)/(n+1),df=4)
qqplot(x,sort(AMZN_res_sd),xlab="t(4)",ylab="residuals") # better than normal fitting
```




```
AMZN_fit1=garchFit(formula=~arma(1,0)+garch(1,1),data=AMZN_rets,cond.dist="std")

##
## Series Initialization:
##  ARMA Model:                arma
##  Formula Mean:              ~ arma(1, 0)
##  GARCH Model:               garch
##  Formula Variance:          ~ garch(1, 1)
##  ARMA Order:                1 0
##  Max ARMA Order:            1
##  GARCH Order:               1 1
##  Max GARCH Order:           1
##  Maximum Order:             1
##  Conditional Dist:          std
##  h.start:                   2
##  llh.start:                 1
##  Length of Series:          2747
##  Recursion Init:            mci
##  Series Scale:              0.0209
##
## Parameter Initialization:
##  Initial Parameters:          $params
##  Limits of Transformations:   $U, $V
##  Which Parameters are Fixed?  $includes
##  Parameter Matrix:
##                        U        V    params includes
##     mu     -0.60787295   0.608  0.06079     TRUE
##     ar1    -0.99999999   1.000 -0.00585     TRUE
##     omega   0.00000100 100.000  0.10000     TRUE
##     alpha1  0.00000001   1.000  0.10000     TRUE
##     gamma1 -0.99999999   1.000  0.10000    FALSE
##     beta1   0.00000001   1.000  0.80000     TRUE
```

```
##     delta   0.00000000   2.000  2.00000    FALSE
##     skew    0.10000000  10.000  1.00000    FALSE
##     shape   1.00000000  10.000  4.00000     TRUE
##  Index List of Parameters to be Optimized:
##     mu    ar1  omega alpha1  beta1  shape
##      1      2      3      4      6      9
##  Persistence:                  0.9
##
##
## --- START OF TRACE ---
## Selected Algorithm: nlminb
##
## R coded nlminb Solver:
##
##   0:     3470.3861: 0.0607892 -0.00584887 0.100000 0.100000 0.800000  4.00000
##   1:     3468.8932: 0.0607902 -0.00573933 0.0916490 0.100959 0.796782  3.99986
##   2:     3467.8295: 0.0607917 -0.00564393 0.0915111 0.108473 0.801737  3.99994
##   3:     3465.7969: 0.0607964 -0.00532428 0.0756345 0.116957 0.801742  3.99978
##   4:     3464.5029: 0.0608098 -0.00486378 0.0696308 0.127811 0.814783  3.99983
##   5:     3463.5115: 0.0608409 -0.00424631 0.0556217 0.123695 0.825287  3.99966
##   6:     3462.8691: 0.0608849 -0.00393724 0.0513523 0.115623 0.840778  3.99920
##   7:     3462.4855: 0.0609338 -0.00481728 0.0461719 0.103820 0.853289  3.99887
##   8:     3462.3689: 0.0610135 0.00391293 0.0415276 0.0971056 0.866594  3.99727
##   9:     3462.2046: 0.0610753 -0.0100113 0.0367233 0.0918827 0.875378  3.99594
##  10:     3462.1866: 0.0610759 -0.00977510 0.0366927 0.0913052 0.875090  3.99593
##  11:     3462.1700: 0.0610782 -0.00969241 0.0372787 0.0910834 0.875355  3.99588
##  12:     3462.1494: 0.0610930 -0.0100456 0.0374029 0.0898295 0.875315  3.99553
##  13:     3462.1179: 0.0611521 -0.00957568 0.0374080 0.0884442 0.877132  3.99444
##  14:     3462.0863: 0.0611998 -0.00792675 0.0362082 0.0879143 0.878509  3.99365
##  15:     3462.0516: 0.0613099 -0.00701137 0.0356485 0.0854727 0.881797  3.99064
##  16:     3462.0365: 0.0614030 -0.00811534 0.0353671 0.0819321 0.883746  3.98742
##  17:     3462.0074: 0.0615047 -0.00752622 0.0346249 0.0813497 0.885777  3.98272
##  18:     3462.0020: 0.0615556 -0.00605703 0.0330729 0.0812623 0.887333  3.97796
##  19:     3461.9830: 0.0615822 -0.00570613 0.0331012 0.0802903 0.888771  3.97277
##  20:     3461.9713: 0.0616499 -0.00549255 0.0326287 0.0788478 0.890223  3.96781
##  21:     3461.9620: 0.0616229 -0.00504397 0.0318654 0.0766856 0.893040  3.96372
##  22:     3461.9604: 0.0618333 -0.00449016 0.0320981 0.0770627 0.892466  3.95954
##  23:     3461.9586: 0.0617098 -0.00441095 0.0313076 0.0758865 0.894282  3.95572
##  24:     3461.9574: 0.0619903 -0.00411878 0.0310325 0.0752746 0.895135  3.95694
##  25:     3461.9571: 0.0617369 -0.00386630 0.0310093 0.0751124 0.895389  3.95439
##  26:     3461.9571: 0.0618400 -0.00400015 0.0309823 0.0749779 0.895591  3.94979
##  27:     3461.9570: 0.0618199 -0.00392403 0.0309355 0.0747863 0.895736  3.95119
##  28:     3461.9570: 0.0618300 -0.00388972 0.0309253 0.0747978 0.895729  3.95264
##  29:     3461.9570: 0.0618084 -0.00385879 0.0309167 0.0748223 0.895720  3.95280
##  30:     3461.9570: 0.0618165 -0.00386721 0.0309207 0.0748110 0.895724  3.95292
##  31:     3461.9570: 0.0618169 -0.00386805 0.0309202 0.0748113 0.895724  3.95288
##
## Final Estimate of the Negative LLH:
##  LLH:  -7164    norm LLH:  -2.61
##         mu        ar1      omega     alpha1      beta1      shape
##  0.0012919 -0.0038681  0.0000135  0.0748113  0.8957241  3.9528806
##
## R-optimhess Difference Approximated Hessian Matrix:
##                mu      ar1         omega     alpha1      beta1      shape
## mu     -12101926 -19932.1      -1275231      12408       -911      427.3
```

```
## ar1       -19932  -3044.0        699116        137        271        10.5
## omega   -1275231 699116.4 -754789924809 -167216441 -240030026 -3678168.6
## alpha1     12408    137.2    -167216441     -55538     -66154    -1030.0
## beta1       -911    270.8    -240030026     -66154     -86050    -1332.0
## shape        427     10.5      -3678169      -1030      -1332      -32.1
## attr(,"time")
## Time difference of 0.148 secs
##
## --- END OF TRACE ---
##
##
## Time to Estimate Parameters:
##  Time difference of 0.467 secs
```

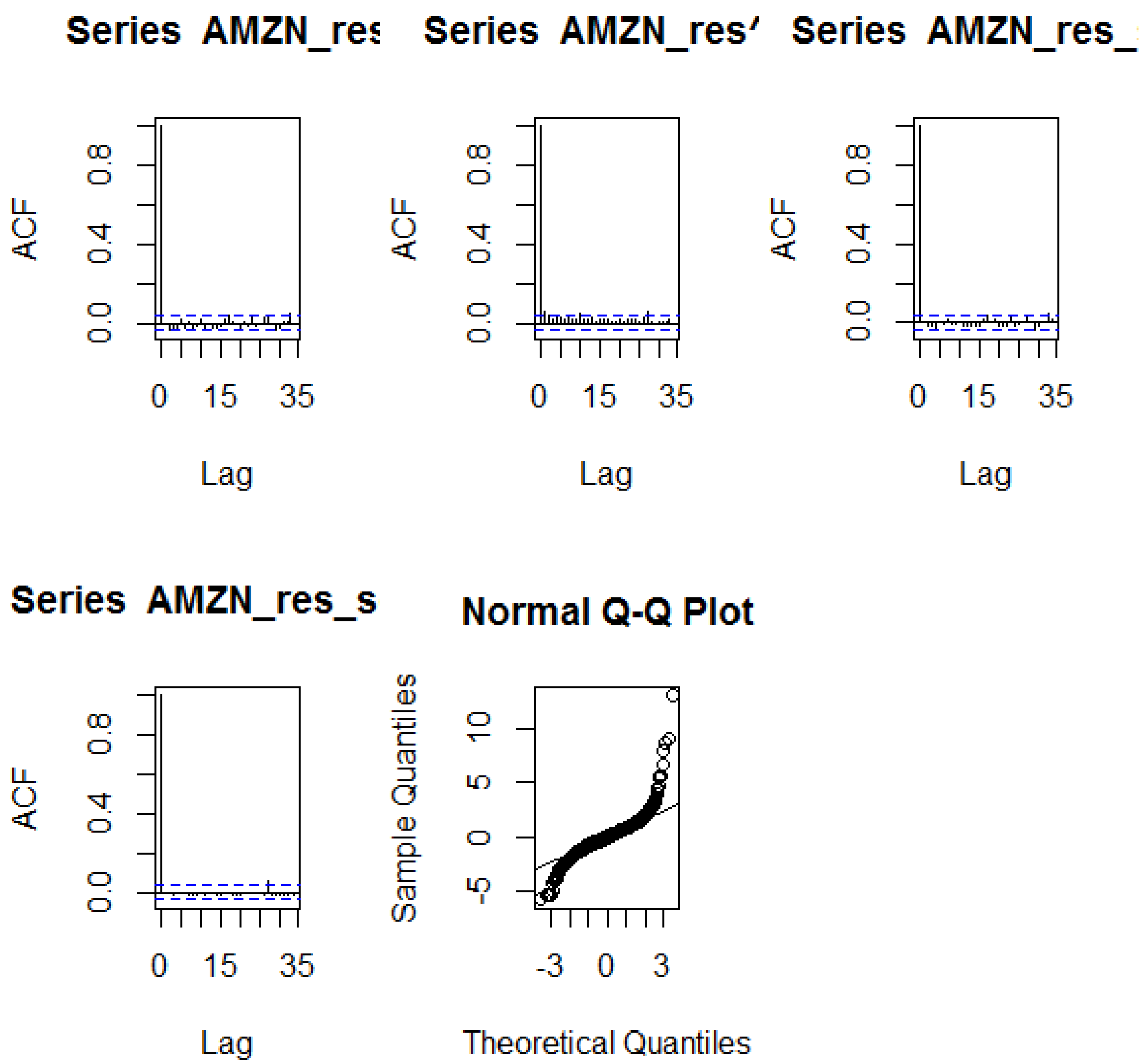
```
coef(AMZN_fit1)
```

```
##         mu        ar1      omega     alpha1      beta1      shape
##  0.0012919 -0.0038681  0.0000135  0.0748113  0.8957241  3.9528806
```

```
predict(AMZN_fit1,n.ahead=10)
```

```
##    meanForecast meanError standardDeviation
## 1       0.00133    0.0140            0.0140
## 2       0.00129    0.0143            0.0143
## 3       0.00129    0.0145            0.0145
## 4       0.00129    0.0148            0.0148
## 5       0.00129    0.0150            0.0150
## 6       0.00129    0.0152            0.0152
## 7       0.00129    0.0155            0.0155
## 8       0.00129    0.0157            0.0157
## 9       0.00129    0.0159            0.0159
## 10      0.00129    0.0161            0.0161
```

```
AMZN_res=residuals(AMZN_fit1) # get residuals of ARMA model
AMZN_res_sd=residuals(AMZN_fit1,standardize=TRUE) # get residuals of ARMA-GARCH model

acf(AMZN_res) # white noise
acf(AMZN_res^2) # significant correlation => not white noise
acf(AMZN_res_sd) # white noise
acf(AMZN_res_sd^2) # white noise => AR(1)-GARCH(1,1) IS GOOD. If not WN, it does not catc
h all of the dependence.
qqnorm(AMZN_res_sd)
qqline(AMZN_res_sd)
```



Fitting Copula Models to Bivariate Return Data

Univariate marginal t-distributions and t-copula

First, I fit a model with univariate marginal t-distributions and a t-copula. The model has three degrees-of-freedom (tail index) parameters, one for each of the two univariate models and a third for the copula. This means that the univariate distributions can have different tail indices and that their tail indices are independent of the tail dependence from the copula.

The univariate estimates will be used as starting values when the meta-t-distribution is fit by maximum likelihood. I also need an estimate of the correlation coefficient in the t-copula which is obtained using Kendall's tau.

```
est.SPXL_res = as.numeric(fitdistr(SPXL_res_sd,"t")$estimate)
est.AMZN_res = as.numeric(fitdistr(AMZN_res_sd,"t")$estimate)
est.SPXL_res[2] = est.SPXL_res[2] * sqrt(est.SPXL_res[3] / (est.SPXL_res[3]-2))
est.AMZN_res[2] = est.AMZN_res[2] * sqrt(est.AMZN_res[3] / (est.AMZN_res[3]-2))
cor_tau = cor(SPXL_res_sd, AMZN_res_sd, method = "kendall", use="pairwise.complete.obs")
```

```
print(paste0("Estimate of the correlation coefficient in the t-copula using Kendall's tau
:", round(cor_tau,2)))

## [1] "Estimate of the correlation coefficient in the t-copula using Kendall's tau:0.42"

omega = sin((pi/2)*cor_tau)
print(paste0("omega:", round(omega,2)))

## [1] "omega:0.62"
```

Define the t-copula using omega as the correlation parameter and 4 as the degrees-of-freedom (tail index) parameter.

```
print("Parametric Approach")

## [1] "Parametric Approach"

t_copula_param

## Call: fitCopula(copula, data = data, method = "ml", start = ..2)
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: tCopula
## rho.1    df
## 0.611 6.141
## The maximized loglikelihood is 649
## Optimization converged

print("Nonparametric Approach")

## [1] "Nonparametric Approach"

t_copula_nonparam

## Call: fitCopula(copula, data = data, method = "ml", start = ..2)
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: tCopula
## rho.1    df
##  0.61  6.01
## The maximized loglikelihood is 647
## Optimization converged
```

Both fits are by pseudo-likelihood. ft_param is the parametric approach because the univariate marginal distributions are estimated by fitting t-distributions, and ft_nonparam is the nonparametric approach because the univariate distributions are estimated by empirical CDFs. The two estimates of the correlation are 0.61 and 0.616, which are similar to 0.62 by using Kendall's tau.

Other copulas

Second, fit normal (Gaussian), Frank, Clayton, Gumbel and Joe copulas to the data.

```
Gaussian

## Call: fitCopula(copula, data = data, method = "ml")
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: normalCopula
## rho.1
## 0.588
```

```
## The maximized loglikelihood is 582
## Optimization converged
```

```
Frank
```

```
## Call: fitCopula(copula, data = data, method = "ml")
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: frankCopula
## alpha
##  4.57
## The maximized loglikelihood is 603
## Optimization converged
```

```
Clayton
```

```
## Call: fitCopula(copula, data = data, method = "ml")
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: claytonCopula
## alpha
## 0.937
## The maximized loglikelihood is 509
## Optimization converged
```

```
Gumbel
```

```
## Call: fitCopula(copula, data = data, method = "ml")
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: gumbelCopula
## alpha
##  1.66
## The maximized loglikelihood is 566
## Optimization converged
```

```
Joe
```

```
## Call: fitCopula(copula, data = data, method = "ml")
## Fit based on "maximum likelihood" and 2747 2-dimensional observations.
## Copula: joeCopula
## alpha
##  1.83
## The maximized loglikelihood is 407
## Optimization converged
```

Comparison Copulas

Assess the fit by AIC. With the table below, the AIC of t-copula is smaller than others. Therefore, I choose t-copula (parametric) as my copula model since it fits the data best.

Copula	loglik	AIC
t_copula_param	649.09	-1294.18
t_copula_nonparam	646.8	-1289.6
Frank	603.37	-1204.75
Gaussian	581.76	-1161.53
Gumbel	566.17	-1130.33
Clayton	509.25	-1016.5

Joe	406.6	-811.2

Risk Calculation: Parametric Estimation of VaR and ES

Parametric estimation allows the use of GARCH models to adapt the risk measures to the current estimate of volatility. Also, risk measures can be easily computed for a portfolio of stocks if we assume that their returns have a joint parametric distribution, such as a multivariate t-distribution. Nonparametric estimation using sample quantiles works best when the sample size and α are reasonably large. With smaller sample sizes or smaller values of α, it is preferable to use parametric estimation.

Compare VaR and ES parametric (unconditional) estimates with those from using ARMA+GARCH (conditional) models.

VaR and ES parametric (unconditional) estimates

First, assume that Investment (S) equals to 100000 and the returns are iid and follow a t-distribution. I fit t distribution to SPXL and AMZN.

```
SPXL_res = fitdistr(SPXL_rets,"t")
SPXL_mu = SPXL_res$estimate["m"]
SPXL_lambda = SPXL_res$estimate["s"]
SPXL_nu = SPXL_res$estimate["df"]
print(paste0("qt(alpha, df=SPXL_nu):", qt(alpha, df=SPXL_nu)))

## [1] "qt(alpha, df=SPXL_nu):-5.45580771895402"

print(paste0("dt(qt(alpha, df=SPXL_nu), df=SPXL_nu):", dt(qt(alpha, df=SPXL_nu), df=SPXL_
nu)))

## [1] "dt(qt(alpha, df=SPXL_nu), df=SPXL_nu):0.00423111517508101"

SPXL_Finv = SPXL_mu + SPXL_lambda * qt(alpha, df=SPXL_nu)
SPXL_uncond_VaR = -S * SPXL_Finv
SPXL_den = dt(qt(alpha, df=SPXL_nu), df=SPXL_nu)
SPXL_uncond_ES = S * (-SPXL_mu + SPXL_lambda*(SPXL_den/alpha)*(SPXL_nu+qt(alpha, df=SPXL_
nu)^2 )/(SPXL_nu-1))
print(paste0("SPXL unconditional VaR:", as.numeric(SPXL_uncond_VaR)))

## [1] "SPXL unconditional VaR:9333.56851724623"

print(paste0("SPXL unconditional ESL", as.numeric(SPXL_uncond_ES)))

## [1] "SPXL unconditional ESL16235.0869456025"

AMZN_res = fitdistr(AMZN_rets,"t")
AMZN_mu = AMZN_res$estimate["m"]
AMZN_lambda = AMZN_res$estimate["s"]
AMZN_nu = AMZN_res$estimate["df"]
print(paste0("qt(alpha, df=AMZN_nu):", qt(alpha, df=AMZN_nu)))

## [1] "qt(alpha, df=AMZN_nu):-4.33133602887618"

print(paste0("dt(qt(alpha, df=AMZN_nu), df=AMZN_nu):", dt(qt(alpha, df=AMZN_nu), df=AMZN_
nu)))

## [1] "dt(qt(alpha, df=AMZN_nu), df=AMZN_nu):0.00649362091463872"
```

```r
AMZN_Finv = AMZN_mu + AMZN_lambda * qt(alpha, df=AMZN_nu)
AMZN_uncond_VaR = -S * AMZN_Finv
AMZN_den = dt(qt(alpha, df=AMZN_nu), df=AMZN_nu)
AMZN_uncond_ES = S * (-AMZN_mu + AMZN_lambda*(AMZN_den/alpha)*(AMZN_nu+qt(alpha, df=AMZN_
nu)^2 )/(AMZN_nu-1))
print(paste0("AMZN unconditional VaR:", as.numeric(AMZN_uncond_VaR)))

## [1] "AMZN unconditional VaR:5630.80245925184"

print(paste0("AMZN unconditional ESL", as.numeric(AMZN_uncond_ES)))

## [1] "AMZN unconditional ESL8516.48162035553"

## [1] "Uncoditional estimates of VaR parametric:14964.3709764981"

## [1] "Uncoditional estimates of ES parametric:24751.5685659581"
```

The unconditional estimates are just a simple look for the data. Now, we need to focus on conditional one, which is what we actually need.

VaR and ES parametric estimates using ARMA+GARCH (conditional) models

Fit a ARMA(0,0)+GARCH(1,1) model to the returns and calculate one step forecasts.

```r
garch.SPXL = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                      variance.model=list(garchOrder=c(1,1)),
                      distribution.model = "norm")
SPXL.garch.fit = ugarchfit(data=SPXL_rets, spec=garch.SPXL)
show(SPXL.garch.fit)

##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.002285    0.000390   5.8608 0.000000
## ar1    -0.036418    0.021453  -1.6976 0.089579
## omega   0.000027    0.000004   7.0344 0.000000
## alpha1  0.163307    0.016177  10.0952 0.000000
## beta1   0.811566    0.015259  53.1847 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.002285    0.000380   6.0093 0.000000
## ar1    -0.036418    0.019763  -1.8428 0.065364
## omega   0.000027    0.000006   4.4448 0.000009
## alpha1  0.163307    0.024298   6.7209 0.000000
## beta1   0.811566    0.022585  35.9332 0.000000
```

```
##
## LogLikelihood : 6190
##
## Information Criteria
## ------------------------------------
##
## Akaike       -4.5028
## Bayes        -4.4921
## Shibata      -4.5029
## Hannan-Quinn -4.4990
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.7278  0.3936
## Lag[2*(p+q)+(p+q)-1][2]    0.7365  0.8817
## Lag[4*(p+q)+(p+q)-1][5]    1.3820  0.8744
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.3145  0.5749
## Lag[2*(p+q)+(p+q)-1][5]    1.1107  0.8341
## Lag[4*(p+q)+(p+q)-1][9]    1.6213  0.9451
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]   0.04132 0.500 2.000  0.8389
## ARCH Lag[5]   0.49092 1.440 1.667  0.8863
## ARCH Lag[7]   0.80152 2.315 1.543  0.9436
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  3.35
## Individual Statistics:
## mu     0.04898
## ar1    0.09626
## omega  0.38686
## alpha1 0.64989
## beta1  1.34546
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.28 1.47 1.88
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value        prob sig
## Sign Bias            3.311 0.000941730 ***
## Negative Sign Bias   1.222 0.221927607
## Positive Sign Bias   1.594 0.111068172
```

```
## Joint Effect        27.729 0.000004139 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic                     p-value(g-1)
## 1    20     164.8 0.00000000000000000000000297445
## 2    30     179.4 0.00000000000000000000012829701
## 3    40     215.9 0.00000000000000000000000023768
## 4    50     242.7 0.00000000000000000000000001831
##
##
## Elapsed time : 0.263
```

```
garch.AMZN = ugarchspec(mean.model=list(armaOrder=c(1,0)),
                     variance.model=list(garchOrder=c(1,1)),
                     distribution.model = "std")
AMZN.garch.fit = ugarchfit(data=AMZN_rets, spec=garch.AMZN)
show(AMZN.garch.fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001283    0.000287  4.47200 0.000008
## ar1    -0.003824    0.018216 -0.20992 0.833732
## omega   0.000013    0.000001 11.12756 0.000000
## alpha1  0.073735    0.002547 28.95222 0.000000
## beta1   0.897011    0.011411 78.61001 0.000000
## shape   3.959373    0.279165 14.18290 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001283    0.000262  4.90035 0.000001
## ar1    -0.003824    0.016679 -0.22926 0.818666
## omega   0.000013    0.000002  5.70892 0.000000
## alpha1  0.073735    0.010810  6.82128 0.000000
## beta1   0.897011    0.016537 54.24162 0.000000
## shape   3.959373    0.448728  8.82356 0.000000
##
## LogLikelihood : 7164
##
## Information Criteria
## ------------------------------------
##
## Akaike       -5.2112
```

```
## Bayes        -5.1983
## Shibata      -5.2112
## Hannan-Quinn -5.2065
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                   0.006584  0.9353
## Lag[2*(p+q)+(p+q)-1][2]  0.445443  0.9759
## Lag[4*(p+q)+(p+q)-1][5]  2.307726  0.6218
## d.o.f=1
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                    0.04149  0.8386
## Lag[2*(p+q)+(p+q)-1][5]   0.50464  0.9570
## Lag[4*(p+q)+(p+q)-1][9]   0.87527  0.9908
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.5349 0.500 2.000  0.4646
## ARCH Lag[5]    0.5853 1.440 1.667  0.8582
## ARCH Lag[7]    0.7398 2.315 1.543  0.9518
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  25.4
## Individual Statistics:
## mu     0.15078
## ar1    0.08887
## omega  8.74483
## alpha1 1.19190
## beta1  2.51922
## shape  1.32816
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value    prob sig
## Sign Bias           2.1342 0.03292  **
## Negative Sign Bias  1.1752 0.24002
## Positive Sign Bias  0.4444 0.65679
## Joint Effect        4.9221 0.17759
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
```

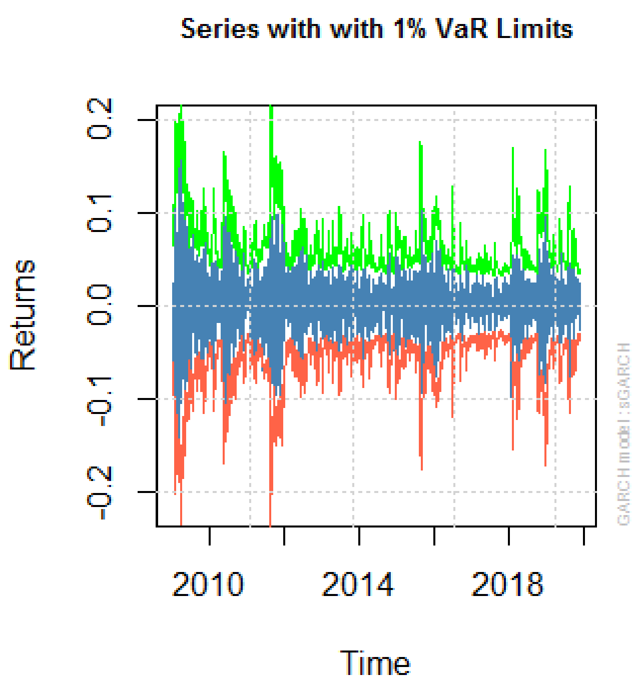
```
## 1     20      12.89       0.8441
## 2     30      20.72       0.8691
## 3     40      30.42       0.8357
## 4     50      46.14       0.5899
##
##
## Elapsed time : 0.589
```

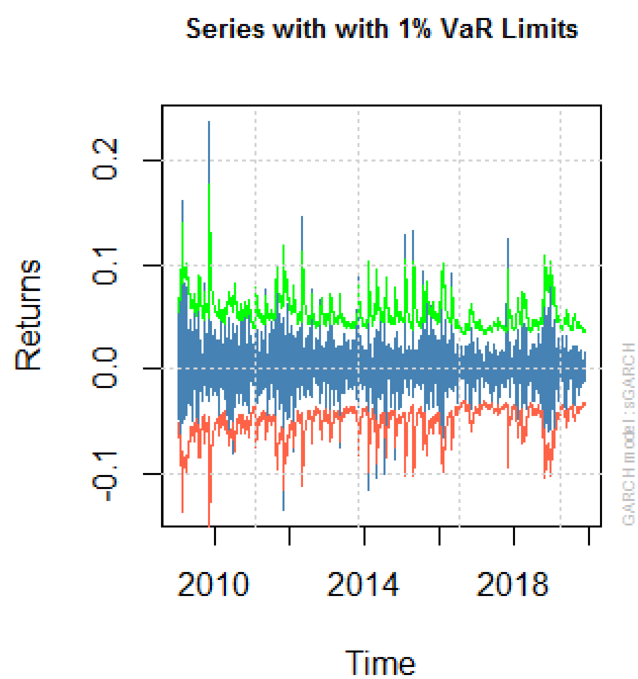
The left plot below is SPXL series with 1% VaR Limits and the right is AMZN series with 1% VaR Limits.

```r
plot(SPXL.garch.fit, which = 2, VaR.alpha=0.01)
```

please wait...calculating quantiles...

```r
plot(AMZN.garch.fit, which = 2, VaR.alpha=0.01)
```





Creating rolling forecasts of the conditional GARCH density, and calculating the Value at Risk at specified levels. The argument `refit.every` determines every how many periods the model is re-estimated.

Kupiec's unconditional coverage compares the number of expected versus actual exceedances given the tail probability of VaR, while the Christoffersen test is a joint test of the unconditional coverage and the independence of the exceedances.

SPXL GARCH VaR estimates are not much better than unconditional estimates based on statistical tests.

```r
SPXL_cond_roll <- ugarchroll(garch.SPXL, SPXL_rets, n.start = 2000, refit.every = 500, re
fit.window = "moving", solver = "hybrid", calculate.VaR = TRUE, VaR.alpha = 0.01, keep.co
ef = TRUE, solver.control = list(tol = 1e-7, delta = 1e-9), fit.control = list(scale = 1)
)
report(SPXL_cond_roll, type = "VaR", VaR.alpha = 0.01, conf.level = 0.99)
```

```
## VaR Backtest Report
## ===========================================
## Model:              sGARCH-norm
```

```
## Backtest Length: 747
## Data:
##
## ==========================================
## alpha:              1%
## Expected Exceed: 7.5
## Actual VaR Exceed:   18
## Actual %:           2.4%
##
## Unconditional Coverage (Kupiec)
## Null-Hypothesis: Correct Exceedances
## LR.uc Statistic: 10.752
## LR.uc Critical:      6.635
## LR.uc p-value:       0.001
## Reject Null:     YES
##
## Conditional Coverage (Christoffersen)
## Null-Hypothesis: Correct Exceedances and
##                  Independence of Failures
## LR.cc Statistic: 11.326
## LR.cc Critical:      9.21
## LR.cc p-value:       0.003
## Reject Null:     YES
```

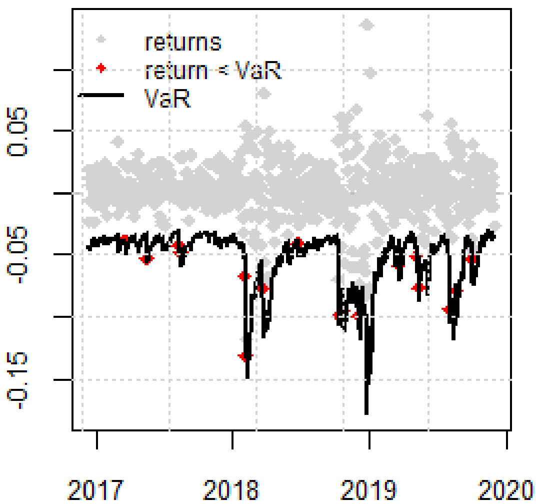
AMZN GARCH VaR estimates are much better than unconditional estimates based on statistical tests.

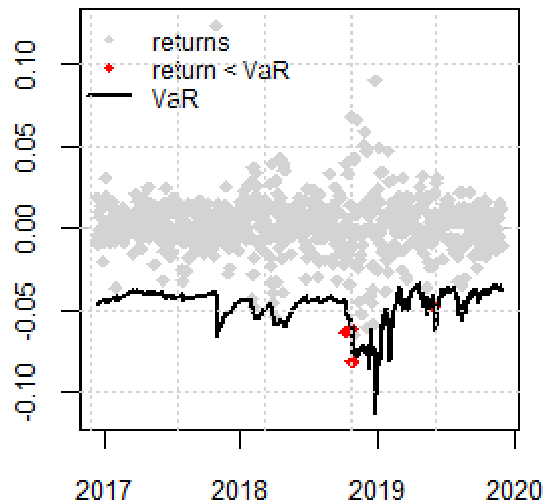
```
AMZN_cond_roll <- ugarchroll(garch.AMZN, AMZN_rets, n.start = 2000, refit.every = 500, re
fit.window = "moving", solver = "hybrid", calculate.VaR = TRUE, VaR.alpha = 0.01, keep.co
ef = TRUE, solver.control = list(tol = 1e-7, delta = 1e-9), fit.control = list(scale = 1)
)
report(AMZN_cond_roll, type = "VaR", VaR.alpha = 0.01, conf.level = 0.99)
```

```
## VaR Backtest Report
## ===========================================
## Model:              sGARCH-std
## Backtest Length: 747
## Data:
##
## ==========================================
## alpha:              1%
## Expected Exceed: 7.5
## Actual VaR Exceed:   4
## Actual %:           0.5%
##
## Unconditional Coverage (Kupiec)
## Null-Hypothesis: Correct Exceedances
## LR.uc Statistic: 1.959
## LR.uc Critical:      6.635
## LR.uc p-value:       0.162
## Reject Null:     NO
##
## Conditional Coverage (Christoffersen)
## Null-Hypothesis: Correct Exceedances and
##                  Independence of Failures
## LR.cc Statistic: 2.003
## LR.cc Critical:      9.21
```

```
## LR.cc p-value:       0.367
## Reject Null:      NO
```

```
plot(SPXL_cond_roll, which = 4)
plot(AMZN_cond_roll, which = 4)
```





Predict VaR based on fitted model

```
SPXL_cond_pred = ugarchforecast(SPXL.garch.fit, data=SPXL_rets, n.ahead=1)
SPXL_cond_pred

## 
## *------------------------------------*
## *       GARCH Model Forecast         *
## *------------------------------------*
## Model: sGARCH
## Horizon: 1
## Roll Steps: 0
## Out of Sample: 0
## 
## 0-roll forecast [T0=2019-12-02]:
##       Series  Sigma
## T+1 0.003339 0.0193

## Extract the resulting series
SPXL_mu.predict <- fitted(SPXL_cond_pred) # extract predicted X_t (= conditional mean mu_
t; note: E[Z] = 0)
SPXL_sig.predict <- sigma(SPXL_cond_pred) # extract predicted sigma_t
SPXL_VaR.predict <- as.numeric(SPXL_mu.predict + SPXL_sig.predict * qnorm(0.01)) # corres
ponding predicted VaR_alpha

AMZN_cond_pred = ugarchforecast(AMZN.garch.fit, data=AMZN_rets, n.ahead=1)
AMZN_cond_pred
```

```
##
## *------------------------------------*
## *       GARCH Model Forecast         *
## *------------------------------------*
## Model: sGARCH
## Horizon: 1
## Roll Steps: 0
## Out of Sample: 0
##
## 0-roll forecast [T0=2019-12-02]:
##       Series   Sigma
## T+1 0.001329 0.01396
```

```
AMZN_mu.predict <- fitted(AMZN_cond_pred)
AMZN_sig.predict <- sigma(AMZN_cond_pred)
AMZN_VaR.predict = AMZN_mu.predict + AMZN_sig.predict * qdist(distribution='std', shape=A
MZN_res$estimate['df'], p=0.01)

#print(paste0("SPXL VaR Prediction: ", round(SPXL_VaR.predict,3)))
#print(paste0("AMZN VaR Prediction: ", round(AMZN_VaR.predict,3)))
#print(paste0("SPXL ES Prediction: ", ESnorm(1-alpha, mu = SPXL_mu.predict, sd = SPXL_sig
.predict)))
#print(paste0("AMZN ES Prediction: ", ESst(1-alpha, mu = AMZN_mu.predict, sd = AMZN_sig.p
redict, df = AMZN_res$estimate["df"], scale = T)))
print(paste0("VaR Conditional Prediction: ", round(SPXL_VaR.predict + AMZN_VaR.predict,3)
))
```

```
## [1] "VaR Conditional Prediction: -0.077"
```

```
print(paste0("ES Conditional Prediction: ", round(ESnorm(1-alpha, mu = SPXL_mu.predict, s
d = SPXL_sig.predict) + ESst(1-alpha, mu = AMZN_mu.predict, sd = AMZN_sig.predict, df = A
MZN_res$estimate["df"], scale = T),3)))
```

```
## [1] "ES Conditional Prediction: 0.112"
```

Uncertainty Quantification

Using the bootstrap method to quantify the uncertainty of the Value-at-Risk estimations. With ugarchboot function, it is easy to conduct bootstrap method and calculate VaR.

```
SPXL_garch.boot = ugarchboot(SPXL.garch.fit, method="Partial", n.ahead=1, n.bootpred=2000
)
SPXL_garch.boot
```

```
##
## *-----------------------------------*
## *     GARCH Bootstrap Forecast      *
## *-----------------------------------*
## Model : sGARCH
## n.ahead : 1
## Bootstrap method:  partial
## Date (T[0]): 2019-12-02
##
## Series (summary):
##           min      q.25     mean     q.75      max forecast[analytic]
## t+1 -0.073766 -0.007107 0.002351 0.013793 0.071522           0.003339
```

```
## .....................
##
## Sigma (summary):
##           min    q0.25     mean    q0.75      max forecast[analytic]
## t+1 0.019303 0.019303 0.019303 0.019303 0.019303           0.019303
## .....................
```

```
AMZN_garch.boot = ugarchboot(AMZN.garch.fit, method="Partial", n.ahead=1, n.bootpred=2000
)
AMZN_garch.boot
```

```
##
## *-----------------------------------*
## *     GARCH Bootstrap Forecast      *
## *-----------------------------------*
## Model : sGARCH
## n.ahead : 1
## Bootstrap method:  partial
## Date (T[0]): 2019-12-02
##
## Series (summary):
##           min      q.25     mean     q.75     max forecast[analytic]
## t+1 -0.07486 -0.006805 0.000884 0.008671 0.12817           0.001329
## .....................
##
## Sigma (summary):
##           min    q0.25     mean    q0.75      max forecast[analytic]
## t+1 0.013961 0.013961 0.013961 0.013961 0.013961           0.013961
## .....................
```

```
SPXL_mu.boot <- fitted(SPXL_garch.boot@forc)
SPXL_sig.boot <- sigma(SPXL_garch.boot@forc)
SPXL_VaR.boot <- as.numeric(SPXL_mu.boot + SPXL_sig.boot * qnorm(0.01))
AMZN_mu.boot <- fitted(AMZN_garch.boot@forc)
AMZN_sig.boot <- sigma(AMZN_garch.boot@forc)
AMZN_VaR.boot = AMZN_mu.boot + AMZN_sig.boot * qdist(distribution='std', shape=AMZN_res$e
stimate['df'], p=0.01)
```

```
## [1] "VaR Bootstrap Prediction: -0.077"
```

```
## [1] "ES Bootstrap Prediction: 0.112"
```

VaR with bootstrap method is similar to the previous one.

Conclusion

In this project, SPXL is better with normal and AMZN is better with t distribution. After reviewing AIC, t-copula fits the data best. Moreover, SPXL GARCH VaR estimates are not much better than unconditional estimates based on statistical tests. AMZN GARCH VaR estimates are much better than unconditional estimates based on statistical tests. The VaR with bootstrap method is much similar to simulation method.

Reference and Appendix

Statistics and Data Analysis for Financial Engineering with R examples Second Edition – David Ruppert, David S. Matteson

Value at Risk estimation using GARCH model – Ionas Kelepouris & Dimos Kelepouris

Fitting and Predicting VaR based on an ARMA-GARCH Process – Marius Hofert

VaR with GARCH(1,1) – Dejan Prvulovic

Issues in estimating VaR with GARCH