FE 621

Final

Zhengkun Ye

12/16/2018

Problem A. Pricing basket options.

(a) Let the correlation matrix A be defined as:

$$A = \begin{bmatrix} 1.0 & 0.5 & 0.2 \\ 0.5 & 1.0 & -0.4 \\ 0.2 & -0.4 & 1.0 \end{bmatrix}.$$

Consider three assets, starting with $S(0) = [100, 101, 98]$. The assets are assumed to follow a standard geometric Brownian motion of the form

$$dS_i(t) = \mu_i S_i(t)dt + \sigma_i S_i(t)dW_i(t).$$

We assume $\mu = [0.03, 0.06, 0.02]$, the volatility $\sigma = [0.05, 0.2, 0.15]$, and the BM's have the correlation matrix A (e.g., $d < W_1, W_2 >= a_{12}dt = 0.5dt$).

```
temp <- c(1.0 , 0.5, 0.2, 0.5, 1.0, -0.4, 0.2, -0.4, 1.0)
A <- matrix(temp, nrow = 3, ncol = 3, byrow = T)
A
```

```
> A
     [,1] [,2] [,3]
[1,]  1.0  0.5  0.2
[2,]  0.5  1.0 -0.4
[3,]  0.2 -0.4  1.0
>
```

```
S_0 <- c(100, 101, 98)
mu <- c(0.03, 0.06, 0.02)
sigma <- c(0.05, 0.2, 0.15)

T <- 100
m <- 1000
dt <- 1/365

Three_dim_matrix <- array(NA, dim = c(T, m, length(S_0)))
```

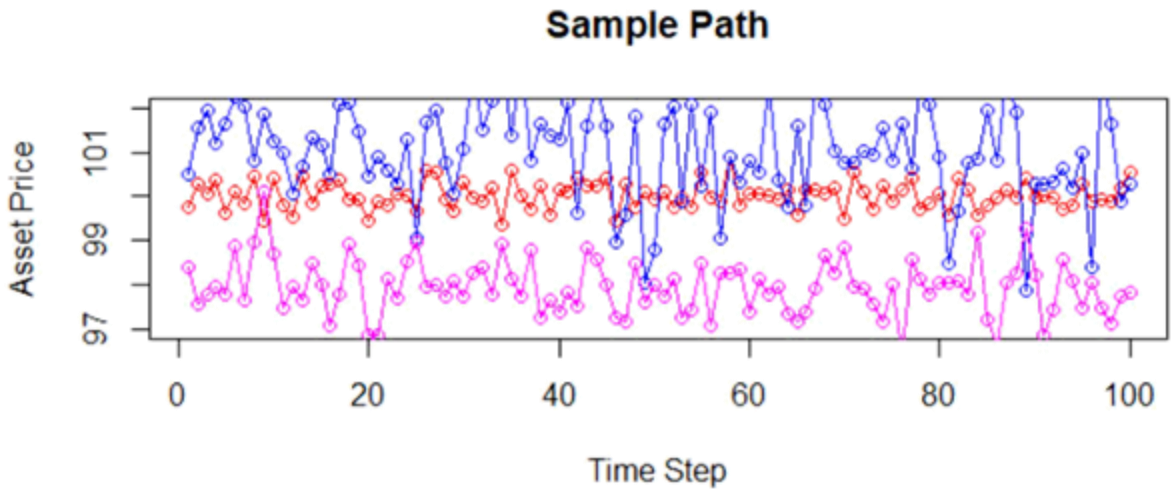
(b) We take maturity $T = 100$ days and the number of simulated paths is $m = 1000$, Consider one day sampling frequency, i.e. $\Delta t = 1/365$. Plot these 1000 sample paths.

```
for(i in 1 : length(S_0))
{
  nudt <- (mu[i] - 0.5 * sigma[i]^2) * dt
  sigsdt <- sigma[i] * sqrt(dt)
  lnS_0 <- log(S_0)

  for(j in 1 : m)
  {
    lnST <- lnS_0[i]
    for(k in 1 : T)
    {
      dw <- rnorm(3, 0, 1) %*% L
      lnST <- lnST + nudt + sigsdt * dw[i]
      Three_dim_matrix[k, j, i] <- exp(lnST)
    }
  }
}
Three_dim_matrix

plot(Three_dim_matrix[,1,1], ylim = c(97, 102), type = "o", col = "red", xlab = "Time Step",
     ylab = "Asset Price", main = "Sample Path")

lines(Three_dim_matrix[,1,2], type = "o", col = "blue")
lines(Three_dim_matrix[,1,3], type = "o", col = "magenta")
```



(c) Basket options are options on a basket of assets. A commonly traded basket option is a vanilla call/put option on a linear combination of assets. To clarify, suppose $S_i(t), i = 1, \ldots, N$ are the prices of N stocks at time t and let $a_i, i = 1, \ldots, N$ are real constants. Set

$$U(t) = \sum_{i=1}^{N} a_i S_i(t)$$

A vanilla basket option is simply a vanilla option on $U(T)$. Specifically, on the exercise date T, the payoff of the option is $\max\{\alpha(U(T) - K), 0\}$, where K is the exercise price and $\alpha = 1$ for a call and $\alpha = -1$ for a put. Price an European call option and an European put option with $K = 100$ on the 3 asset basket given in part (b), using Monte Carlo simulations. Consider a simple average basket $a_1 = a_2 = a_3 = 1/3$. Make sure that the option price is close to the true price. Report a 95% confidence interval of the option price.

```
K <- 100
Asset <- Three_dim_matrix
#since this problem doesn't supply the interest rate, so here
#I choose three interest rate randomly
r <- c(0, 0.03, 0.05)
MC_vanillla <- function(m, K, T, OptionType)
{
  sum_CT <- 0
  a <- c(1/3, 1/3, 1/3)
  for(i in 1 : m)
  {
    UT <- a[1] * Asset[T,i,1] + a[2] * Asset[T,i,2] + a[3] * Asset[T,i,3]
    if(OptionType == "Call")
      alpha <- 1
    else if(OptionType == "Put")
      alpha <- -1


    Payoff <- max(alpha * (UT - K), 0)
    sum_CT <- sum_CT + Payoff
  }
  OptionPrice <- c(1 : 3)
  for(j in 1 : 3)
  {
    OptionPrice[j] <- sum_CT/m * exp(-r[j] * T/365)
  }
  return(OptionPrice)
}

MC_Call <- MC_vanillla(m, K, T, "Call")
MC_Call
MC_Put <- MC_vanillla(m, K, T, "Put")
MC_Put

result_comb <- rbind(MC_Call, MC_Put)
rownames(result_comb) <- c("Call", "Put")
colnames(result_comb) <- c("r=0%", "r=3%", "r=5%")
as.data.frame(result_comb)
```

```
         r=0%     r=3%     r=5%
Call 2.023169 2.006609 1.995644
Put  1.456339 1.444418 1.436525
```

(d) Next price the following exotic options on the basket in part (b). Here we use $B = 104$, $K = 100$, and $T = 100$ days.

(i) If the asset 2 hits the barrier (i.e., $B < S_2(t)$) for some t then the payoff of the option is equal to an European Call option written on the basket $(U(T) - K)_+$ similar to part (c); If the barrier is not hit then the option is worthless

(ii) If $\max_{t\in[0,T]} S_2(t) > \max_{t\in[0,T]} S_3(t)$, then the payoff of the option is $(S_2^2(T) - K)_+$; otherwise, the option is a vanilla call option on the basket, similar to part (c) of this problem.

(iii) Take $A_i(0,T) := \sum_{t=1}^{T} S_i(t)$, the average of the daily values for stock i. If $A_2(0,T) > A_3(0,T)$, then the payoff is $(A_2(0,T) - K)_+$; otherwise the payoff is 0

```
Asset <- Three_dim_matrix
Exotic_Option <- function(m, B, K, T, r)
{
  sum_CT <- 0
  for(i in 1 : m)
  {
    max_s2 <- max(Asset[,i,2])
    max_s3 <- max(Asset[,i,3])
    #For the barrier condition
    if(B < max_s2)
    {
      Payoff <- max((Asset[T,i,2]-K), 0)
      sum_CT <- sum_CT + Payoff
    }
    else
    {
      #For the conditon (ii)
      if(max_s2 > max_s3)
      {
        #Since the result of max(Asset[T,i,2]*Asset[T,i,2] - K, 0)
        #is very large, so I guess here the should not be S2(T)^2-K
        #may be (S2(T)-K)^2
        Payoff <- max(Asset[T,i,2] - K, 0) * max(Asset[T,i,2] - K, 0)
        sum_CT <- sum_CT + Payoff
      }
      else
      {
        A2 <- mean(Asset[,i,2])
        A3 <- mean(Asset[,i,3])
        #For the condition (iii)
        if(A2 > A3)
        {
          Payoff <- max(A2-K, 0)
          sum_CT <- sum_CT + Payoff
        }


        else
        {
          #Vanilla call option
          UT <- (Asset[T,i,1] +  Asset[T,i,2] + Asset[T,i,3])/3
          Payoff <- max((UT - K), 0)
          sum_CT <- sum_CT + Payoff
        }
      }
    }
  }
  OptionPrice <- sum_CT/m * exp(-r * T/365)
}

m <- 1000
T <- 100
B <- 104
K <- 100
r <- 0.03
Price_exotic <- Exotic_Option(m, B, K, T, r)
```

```
> Price_exotic
[1] 5.58196
```

Thus, the price of the exotic option on the basket is 5.58196.

Problem B. Principal Component Analysis.

1. Download daily prices for the components of DJIA for the last 5 years. Construct the corresponding matrix of standardized returns.

$$Y_{it} = \frac{R_{it} - \overline{R}_i}{\overline{\sigma}_i}$$

$$i = 1, ..., N \text{ and } t = 1, ..., T$$

with

$$\overline{R}_i = \frac{\sum_{t=1}^{T} R_{it}}{T} \text{ and } \overline{\sigma}_i = \sqrt{\frac{\sum_{t=1}^{T} (R_{it} - \overline{R}_i)^2}{T}}$$

In [1]:
```
from quantopian.research import run_pipeline


# from quantopian.pipeline import Pipeline
# from quantopian.pipeline.data import morningstar, USEquityPricing
# from quantopian.pipeline.factors import Latest, CustomFactor, SimpleMovingAverage, AverageDollarVolume, Returns, RSI
# from quantopian.pipeline.classifiers.morningstar import Sector
# from quantopian.pipeline.filters import Q500US, Q1500US, QTradableStocksUS

from quantopian.pipeline import data,Pipeline
from quantopian.pipeline import filters
from quantopian.pipeline import factors
from quantopian.pipeline import classifiers



import pandas as pd
import seaborn as sns; sns.set()

import numpy as np
import scipy as sp
from time import time

import alphalens as al
from scipy import stats
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# from Ipython import display
from sklearn import linear_model, decomposition, ensemble, preprocessing, isotonic, metrics
```

In [2]:
```
from quantopian.research import local_csv
```

In [3]:
```
components_df = local_csv("DJIA_components.csv")
components_df.head()
```

Out[3]:

	Symbol	Company Name	Last Price	Change	% Change	Volume
0	PG	The Procter & Gamble Company	81.47	0.04	0.05%	3,025,104
1	INTC	Intel Corporation	48.82	-0.03	-0.06%	9,156,081
2	DIS	The Walt Disney Company	112.61	-0.07	-0.06%	3,615,567
3	BA	The Boeing Company	339.15	-0.26	-0.08%	1,237,816
4	CAT	Caterpillar Inc.	135.68	-0.24	-0.18%	2,514,744

In [4]:
```
components_list = components_df.iloc[:,0].values[:]
```

In [5]:
```
from quantopian.research import returns
```

In [6]:
```
start = pd.Timestamp("2013-05-01")
end = pd.Timestamp("2018-05-01")
```

In [7]:
```
returns_df = returns(components_list,start,end)
returns_df.shape
```

Out[7]: (1260, 30)

In [8]:
```
returns_df = returns_df.fillna(returns_df.mean(axis=0))
```

In [9]:
```
returns_df.head()
```

Out[9]:

	Equity (5938 [PG])	Equity (3951 [INTC])	Equity (2190 [DIS])	Equity (698 [BA])	Equity (1267 [CAT])	Equity (4151 [JNJ])	Equity (5923 [PFE])	Equity (21839 [VZ])	Equity (8089 [WBA])	Equity (23112 [CVX])	...
2013-05-01 00:00:00+00:00	0.003129	0.001660	0.004766	-0.002075	-0.018902	-0.011621	-0.005501	-0.028217	-0.011114	-0.012069	...
2013-05-02 00:00:00+00:00	0.009739	0.005020	0.011553	0.010744	0.014449	0.011171	0.012384	0.002494	-0.001622	0.013551	...
2013-05-03 00:00:00+00:00	0.005667	0.002910	0.014402	0.016912	0.032638	0.007163	-0.010561	0.002655	-0.017800	0.011724	...
2013-05-06 00:00:00+00:00	-0.005500	-0.002273	0.004160	0.004809	0.006789	-0.012362	-0.008077	-0.011951	-0.003124	-0.001627	...
2013-05-07 00:00:00+00:00	0.002441	0.010420	0.015306	0.006365	0.024997	0.009560	0.011300	0.016707	0.027995	-0.001541	...

5 rows × 30 columns

In [10]:
```
return_matrix = returns_df.values.T # (n_assets, n_days)
return_matrix.shape
```

Out[10]: (30, 1260)

In [11]:
```
return_mean = return_matrix.mean(axis=1)
return_std = return_matrix.std(axis=1)
```

In [12]:
```
return_zscore = (return_matrix - return_mean[:,np.newaxis])/return_std[:,np.newaxis]
```

In [13]:
```
return_zscore
```

Out[13]:
```
array([[  3.43535718e-01,   1.09626922e+00,   6.32478105e-01, ...,
          9.67363210e-02,  -7.79108617e-01,  -6.58346545e-01],
       [  5.48332176e-02,   2.86505086e-01,   1.41004563e-01, ...,
         -4.62481494e-01,  -1.51063669e+00,   2.21046811e+00],
       [  3.65467980e-01,   9.45726016e-01,   1.18928711e+00, ...,
         -5.81434635e-01,   9.31784361e-01,  -3.23138840e-01],
       ...,
       [ -2.33864021e+00,   5.72876952e-03,  -1.37914842e-01, ...,
          3.90009959e-02,  -8.18352476e-01,  -1.34254261e+00],
       [ -5.90974896e-01,   8.97976392e-01,   6.10342224e-01, ...,
         -8.61702700e-01,   1.18048021e+00,   1.49700269e+00],
       [ -1.55625436e-19,  -1.55625436e-19,  -1.55625436e-19, ...,
          7.44228324e-01,  -3.05225403e+00,  -6.08620890e-01]])
```

2. Calculate the sample correlation matrix

$$C_{ij} = \frac{1}{T}\sum_{t=1}^{T} Y_{it} Y_{jt}$$

In [14]:
```
n_assets,n_days = return_zscore.shape
```

In [15]:
```
n_assets
```

Out[15]: 30

In [16]:
```
n_days
```

Out[16]: 1260

In [17]:
```
correlation_matrix = 1/n_days * np.dot(return_zscore,return_zscore.T)
```

In [18]:
```
return_corr = np.dot(return_zscore,return_zscore.T)/n_days # our calculation
```

In [19]:
```
return_pearson = np.corrcoef(return_zscore) # Pearson's correlation coefficients from numpy library
```

In [20]:
```
return_corr
```

Out[20]:
```
array([[ 1.        ,  0.31651063,  0.34586084,  0.32744702,  0.26611145,
         0.48806812,  0.351679  ,  0.38917167,  0.30445375,  0.36184289,
         0.38448321,  0.57556357,  0.35145663,  0.38805311,  0.35939783,
         0.31597283,  0.29446122,  0.43464852,  0.37044373,  0.27587018,
         0.30131465,  0.3570394 ,  0.39275277,  0.30269319,  0.42965042,
         0.33259687,  0.31365427,  0.3344719 ,  0.26678955,  0.17959299],
       [ 0.31651063,  1.        ,  0.37164166,  0.36233481,  0.38965603,
         0.35826017,  0.34300929,  0.31761936,  0.25674025,  0.36440369,
         0.30666765,  0.310466  ,  0.37354248,  0.23531045,  0.51762388,
         0.48149235,  0.27267515,  0.36179328,  0.39445492,  0.3531561 ,
         0.30556945,  0.39742359,  0.3785358 ,  0.42148057,  0.45777421,
         0.4031579 ,  0.42345356,  0.29867029,  0.36177763,  0.25273558],
       [ 0.34586084,  0.37164166,  1.        ,  0.42182501,  0.34194208,
         0.38052498,  0.3696546 ,  0.35670697,  0.33555422,  0.36063894,
         0.32349592,  0.34547603,  0.43419338,  0.28016715,  0.35309899,
         0.4048259 ,  0.40284256,  0.4272286 ,  0.42160909,  0.39414143,
         0.34881572,  0.42628305,  0.39742775,  0.47147455,  0.43548365,
         0.36594066,  0.47779378,  0.30275101,  0.30078365,  0.19512212],
       [ 0.32744702,  0.36233481,  0.42182501,  1.        ,  0.44265077,
         0.40209163,  0.32521566,  0.31125823,  0.30986919,  0.37516523,
         0.3192577 ,  0.32580116,  0.39575503,  0.26227786,  0.34218842,
         0.36523094,  0.34089476,  0.39781751,  0.41981637,  0.40459689,
         0.36578006,  0.55618914,  0.38502304,  0.48919391,  0.50715062,
         0.37990204,  0.49120299,  0.2974873 ,  0.33389128,  0.26115068],
       [ 0.26611145,  0.38965603,  0.34194208,  0.44265077,  1.        ,
         0.33108464,  0.29309764,  0.27677751,  0.23254404,  0.52925509,
         0.31834238,  0.24261807,  0.36213576,  0.21001521,  0.3827435 ,
         0.39053225,  0.27442281,  0.37824121,  0.36375301,  0.41518866,
         0.28792626,  0.49307674,  0.51614484,  0.52625397,  0.49809669,
         0.38207636,  0.51620536,  0.26947165,  0.34356317,  0.27440697],
       ...

       [ 1.        ,  0.54579478,  0.41497892,  0.37487872,  0.39277393,
         0.42614924,  0.42841632,  0.42291569,  0.35499207,  0.36841227,
         0.39305097,  0.3453336 ,  0.46098057,  0.4499695 ,  0.40246175,
         0.43304304,  0.46700136,  0.43543052,  0.41230514,  0.55586231,
         0.39758497,  0.43967644,  0.49552351,  0.27502901,  0.21356111],
       [ 0.351679  ,  0.34300929,  0.3696546 ,  0.32521566,  0.29309764,
         0.54579478,  1.        ,  0.34322491,  0.35317967,  0.32988074,
         0.31122401,  0.3245418 ,  0.40002508,  0.31053382,  0.32861033,
         0.3381421 ,  0.31098803,  0.40553287,  0.4138451 ,  0.38778881,
         0.44038238,  0.39910233,  0.37255933,  0.424557  ,  0.43304162,
         0.32764647,  0.46466393,  0.53564703,  0.23967718,  0.19367639],
       [ 0.38917167,  0.31761936,  0.35670697,  0.31125823,  0.27677751,
         0.41497892,  0.34322491,  1.        ,  0.29965357,  0.34431261,
         0.28419762,  0.4124737 ,  0.34655554,  0.31510742,  0.32917139,
         0.30220019,  0.27790168,  0.38982356,  0.28609442,  0.28430247,
         0.22759324,  0.34783498,  0.35860466,  0.31557578,  0.39650615,
         0.3445919 ,  0.34398069,  0.35885892,  0.18951081,  0.18193012],
       ...
```

```
[ 0.30445375,  0.25674025,  0.33555422,  0.30986919,  0.23254404,
  0.37487872,  0.35317967,  0.29965357,  1.        ,  0.26144123,
  0.26445521,  0.26028059,  0.35977744,  0.28715378,  0.31469984,
  0.28362227,  0.28755663,  0.32158276,  0.32611778,  0.29883167,
  0.32887445,  0.34361865,  0.24553689,  0.35567907,  0.33537083,
  0.24574882,  0.33596027,  0.3189372 ,  0.26096852,  0.1033203 ],
[ 0.36184289,  0.36440369,  0.36063894,  0.37516523,  0.52925509,
  0.39277393,  0.32988074,  0.34431261,  0.26144123,  1.        ,
  0.30233938,  0.33916305,  0.3496116 ,  0.20743997,  0.34299995,
  0.38143307,  0.24660333,  0.39282228,  0.36174794,  0.33568831,
  0.31048994,  0.41110346,  0.77178262,  0.46615808,  0.43311119,
  0.38638158,  0.49538752,  0.32771816,  0.26932871,  0.22926211],
[ 0.38448321,  0.30666765,  0.32349592,  0.3192577 ,  0.31834238,
  0.42614924,  0.31122401,  0.28419762,  0.26445521,  0.30233938,
  1.        ,  0.43634912,  0.40585453,  0.30231   ,  0.37636101,
  0.31983771,  0.3293421 ,  0.40046431,  0.38708705,  0.27282943,
  0.3051452 ,  0.32501183,  0.33020011,  0.33568914,  0.40497773,
  0.30601109,  0.36169124,  0.31037648,  0.27641311,  0.14139388],
[ 0.57556357,  0.310466  ,  0.34547603,  0.32580116,  0.24261807,
  0.42841632,  0.3245418 ,  0.4124737 ,  0.26028059,  0.33916305,
  0.43634912,  1.        ,  0.34812826,  0.35575428,  0.33198986,
  0.3208922 ,  0.30689949,  0.46981546,  0.33853726,  0.2909043 ,
  0.28071213,  0.36966819,  0.35039151,  0.28570681,  0.42356493,
  0.32527396,  0.30102966,  0.31800277,  0.21973939,  0.1529338 ],
[ 0.35145663,  0.37354248,  0.43419338,  0.39575503,  0.36213576,
  0.42291569,  0.40002508,  0.34655554,  0.35977744,  0.3496116 ,
  0.40585453,  0.34812826,  1.        ,  0.37946553,  0.39589668,
  0.38063995,  0.4645759 ,  0.4530742 ,  0.45620943,  0.41248481,
  0.38809792,  0.43732741,  0.35508502,  0.452369  ,  0.47995126,
  0.35617215,  0.46582369,  0.31271199,  0.3257026 ,  0.26478169],
[ 0.38805311,  0.23531045,  0.28016715,  0.26227786,  0.21001521,
  0.35499207,  0.31053382,  0.31510742,  0.28715378,  0.20743997,
  0.30231   ,  0.35575428,  0.37946553,  1.        ,  0.25062165,
  0.30470564,  0.27598856,  0.31156949,  0.27711501,  0.23807702,
  0.29225296,  0.33087211,  0.24194557,  0.23360348,  0.32571233,
  0.267631  ,  0.2694766 ,  0.2840055 ,  0.2101603 ,  0.17969822],
```

```
  [ 0.35939783,  0.51762388,  0.35309899,  0.34218842,  0.3827435 ,
    0.36841227,  0.32861033,  0.32917139,  0.31469984,  0.34299995,
    0.37636101,  0.33198986,  0.39589668,  0.25062165,  1.        ,
    0.46259065,  0.32533754,  0.38718334,  0.47190319,  0.36268388,
    0.34644877,  0.39685857,  0.32902337,  0.43029936,  0.44057855,
    0.41073585,  0.4446318 ,  0.27744631,  0.40671951,  0.2069403 ],
  [ 0.31597283,  0.48149235,  0.4048259 ,  0.36523094,  0.39053225,
    0.39305097,  0.3381421 ,  0.30220019,  0.28362227,  0.38143307,
    0.31983771,  0.3208922 ,  0.38063995,  0.30470564,  0.46259065,
    1.        ,  0.31194508,  0.36956844,  0.41835137,  0.33870599,
    0.32261621,  0.43223705,  0.37398943,  0.41838437,  0.46387084,
    0.43063501,  0.45785466,  0.32057727,  0.36834778,  0.24830144],
  [ 0.29446122,  0.27267515,  0.40284256,  0.34089476,  0.27442281,
    0.3453336 ,  0.31098803,  0.27790168,  0.28755663,  0.24660333,
    0.3293421 ,  0.30689949,  0.4645759 ,  0.27598856,  0.32533754,
    0.31194508,  1.        ,  0.35419103,  0.39685037,  0.36121458,
    0.32494416,  0.35722988,  0.25557219,  0.36950274,  0.36525071,
    0.27513075,  0.38542462,  0.25981646,  0.26181453,  0.16255843],
  [ 0.43464852,  0.36179328,  0.4272286 ,  0.39781751,  0.37824121,
    0.46098057,  0.40553287,  0.38982356,  0.32158276,  0.39282228,
    0.40046431,  0.46981546,  0.4530742 ,  0.31156949,  0.38718334,
    0.36956844,  0.35419103,  1.        ,  0.4246406 ,  0.39452036,
    0.40700957,  0.49315398,  0.42128101,  0.52370184,  0.54237477,
    0.41352416,  0.56406729,  0.34134668,  0.27512204,  0.18894422],
  [ 0.37044373,  0.39445492,  0.42160909,  0.41981637,  0.36375301,
    0.4499695 ,  0.4138451 ,  0.28609442,  0.32611778,  0.36174794,
    0.38708705,  0.33853726,  0.45620943,  0.27711501,  0.47190319,
    0.41835137,  0.39685037,  0.4246406 ,  1.        ,  0.47995061,
    0.36461226,  0.46336691,  0.36056496,  0.500325  ,  0.47440569,
    0.39326332,  0.48763863,  0.33739154,  0.35576926,  0.20323833],
  [ 0.27587018,  0.3531561 ,  0.39414143,  0.40459689,  0.41518866,
    0.40246175,  0.38778881,  0.28430247,  0.29883167,  0.33568831,
    0.27282943,  0.2909043 ,  0.41248481,  0.23807702,  0.36268388,
    0.33870599,  0.36121458,  0.39452036,  0.47995061,  1.        ,
    0.37203914,  0.4618218 ,  0.32628137,  0.55882402,  0.44132485,
```

In [21]:
```
return_pearson
```

Out[21]:
```
array([[ 1.        ,  0.31651063,  0.34586084,  0.32744702,  0.26611145,
         0.48806812,  0.351679  ,  0.38917167,  0.30445375,  0.36184289,
         0.38448321,  0.57556357,  0.35145663,  0.38805311,  0.35939783,
         0.31597283,  0.29446122,  0.43464852,  0.37044373,  0.27587018,
         0.30131465,  0.3570394 ,  0.39275277,  0.30269319,  0.42965042,
         0.33259687,  0.31365427,  0.3344719 ,  0.26678955,  0.17959299],
       [ 0.31651063,  1.        ,  0.37164166,  0.36233481,  0.38965603,
         0.35826017,  0.34300929,  0.31761936,  0.25674025,  0.36440369,
         0.30666765,  0.310466  ,  0.37354248,  0.23531045,  0.51762388,
         0.48149235,  0.27267515,  0.36179328,  0.39445492,  0.3531561 ,
         0.30556945,  0.39742359,  0.3785358 ,  0.42148057,  0.45777421,
         0.4031579 ,  0.42345356,  0.29867029,  0.36177763,  0.25273558],
       [ 0.34586084,  0.37164166,  1.        ,  0.42182501,  0.34194208,
         0.38052498,  0.3696546 ,  0.35670697,  0.33555422,  0.36063894,
         0.32349592,  0.34547603,  0.43419338,  0.28016715,  0.35309899,
         0.4048259 ,  0.40284256,  0.4272286 ,  0.42160909,  0.39414143,
         0.34881572,  0.42628305,  0.39742775,  0.47147455,  0.43548365,
         0.36594066,  0.47779378,  0.30275101,  0.30078365,  0.19512212],
       [ 0.32744702,  0.36233481,  0.42182501,  1.        ,  0.44265077,
         0.40209163,  0.32521566,  0.31125823,  0.30986919,  0.37516523,
         0.3192577 ,  0.32580116,  0.39575503,  0.26227786,  0.34218842,
         0.36523094,  0.34089476,  0.39781751,  0.41981637,  0.40459689,
         0.36578006,  0.55618914,  0.38502304,  0.48919391,  0.50715062,
         0.37990204,  0.49120299,  0.2974873 ,  0.33389128,  0.26115068],
       [ 0.26611145,  0.38965603,  0.34194208,  0.44265077,  1.        ,
         0.33108464,  0.29309764,  0.27677751,  0.23254404,  0.52925509,
         0.31834238,  0.24261807,  0.36213576,  0.21001521,  0.3827435 ,
         0.39053225,  0.27442281,  0.37824121,  0.36375301,  0.41518866,
         0.28792626,  0.49307674,  0.51614484,  0.52625397,  0.49809669,
         0.38207636,  0.51620536,  0.26947165,  0.34356317,  0.27440697],
       [ 0.48806812,  0.35826017,  0.38052498,  0.40209163,  0.33108464,
         1.        ,  0.54579478,  0.41497892,  0.37487872,  0.39277393,
         0.42614924,  0.42841632,  0.42291569,  0.35499207,  0.36841227,
         0.39305097,  0.3453336 ,  0.46098057,  0.4499695 ,  0.40246175,
```

The sanity check validates our computation.

3. Calculate the eigenvalues and eigenvectors of the matrix C_{ij} and graph the eigenvalues $\lambda_1 > \lambda_2 > \ldots$. What percent of the trace is explained by summing the first 5 eigenvalues?

In [22]:
```
eigenvalues,eigenvectors = np.linalg.eig(return_corr)
```

Check the sum of trace equal to n_assets

In [23]:
```
np.abs(eigenvalues.sum()-30)<1e-6
```

Out[23]:
```
True
```

In [24]:
```
variance_explained = eigenvalues[:5].sum()/eigenvalues.sum()
```

In [25]:
```
print("variance explained by the first 5 components: {}".format(variance_explained))
```

```
variance explained by the first 5 components: 0.549830564097
```

4. Consider the first eigenvector and denote it by $(V_1, V_2, ..., V_N)$. Define the factor

$$F_t = \frac{1}{\sqrt{\lambda}} \sum_{i=1}^{N} \frac{V_i}{\sigma_i} R_{it} \text{ with } t = 1, 2, ..., T$$

Note: F_t represents the returns of a portfolio which invests $\frac{1}{\sqrt{\lambda_1}} \frac{V_i}{\sigma_i}$ dollars in the i^{th} stock. Calculate the sample mean and sample standard deviation of the factor F.

In [26]:
```
allocation = eigenvectors[0]/return_std/np.sqrt(eigenvalues[0])
```

In [27]:
```
factor_F = np.dot(allocation,return_matrix)
```

In [28]:
```
factor_F
```

Out[28]:
```
array([ 0.30896123, -0.17995745,  0.16354552, ..., -0.00240359,
       -0.24592785,  0.2000278 ])
```

In [29]:
```
factor_F.mean()
```

Out[29]:
```
0.012690239585865702
```

In [30]:
```
factor_F.std()
```

Out[30]:
```
0.26953743954344622
```

5. Consider a series of daily returns of the DIA ETF for the same time period. Perform a linear regression of the returns of F with the standardized returns of DIA. Calculate the R-squared of the regression and discuss whether F and the capitalization-weighted market portfolio are good proxies for each other. Discuss the result and argue why F and the particular market index might be related.

linear regression

In [31]:
```
from quantopian.research import returns,prices
```

In [32]:
```
DIA_return_slice = returns("DIA",start,end) # spider DIA
DIA_price_slice = prices("DIA",start,end)
```

In [33]:
```
DIA_return_array = DIA_return_slice.values
DIA_price_array = DIA_price_slice.values
```

In [34]:
```
DIA_return_array.shape,factor_F.shape
```

Out[34]:
```
((1260,), (1260,))
```

In [35]:
```
from sklearn import linear_model
from sklearn import metrics
```

In [36]:
```
lr = linear_model.LinearRegression()
```

In [37]:
```
lr.fit(X=factor_F[:,np.newaxis],y=DIA_return_array)
```

Out[37]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [38]:
```
DIA_return_predict = lr.predict(X=factor_F[:,np.newaxis])
```

R2

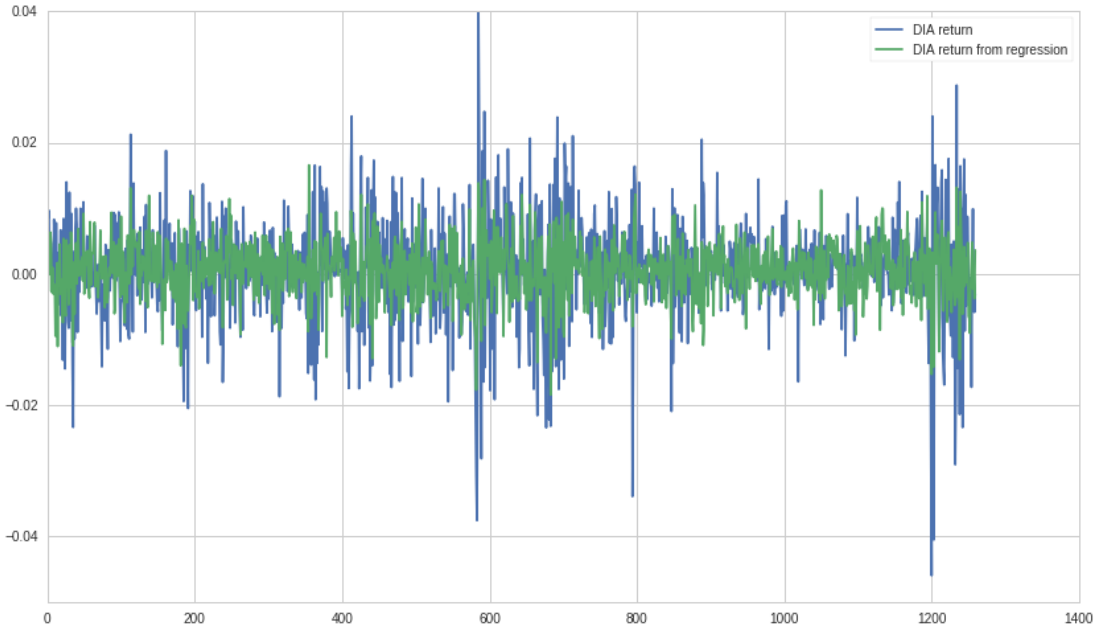
```
In [39]: metrics.r2_score(y_pred=DIA_return_predict,y_true=DIA_return_array)
Out[39]: 0.33059296345675937
```

Visulization

```
In [40]: import matplotlib.pyplot as plt
```

```
In [41]: plt.plot(DIA_return_array)
         plt.plot(DIA_return_predict)
         plt.legend(["DIA return","DIA return from regression"])
Out[41]: <matplotlib.legend.Legend at 0x7f2063093d90>
```

Out[41]: <matplotlib.legend.Legend at 0x7f2063093d90>



Variance explained by the main component

```
In [42]: eigenvalues[0]/eigenvalues.sum()
Out[42]: 0.38706301384096137
```

Theoretically, if DIA is constructed by exactly the 30 assets we used, and never change its component list as well as its allocation, the variance explained using factor F should be 30.6%. And R2 should be much higher than what we computed. However, all the assumptions collapse since the components of DJIA is dynamic as well as DIA.

Even though, DIA as a DJIA ETF successfully shows its validity by a good R2 score to the main component of DJIA components. This is since DIA is constructed to mimic DJIA, and the main component explained 38.7% variance of DJIA, so DIA should be related to this main component.

6. Consider the 5 eigenportfolios (factors) corresponding to the top 5 eigenvalues in the PCA conducted previously. That is, each of the 5 factors are constructed in a similar way with the construction in part4. Let F_{kt} denote the daily return of the eigenportfolio k on date t. For each of the 30 equity in the DJIA do the following:

   - Estimate the mean μ_s and standard deviation σ_s for return of each equity R^s. Standardize the returns $r_k^s = (R_k^s - \mu_s)/\sigma_s$, where k denote the time.
   - Run a regression with the 5 factors and obtain the parameters β_{sk} in:

     $$r_k^s = \sum_{i=1}^{5} \beta_{sk} F_k + \varepsilon_k^s.$$

     Please note that since the returns are standardized the regression intercept should be zero.

   Next we will be using the following factor model for stock returns:

$$R^s = \mu_s + \sigma_s\Big(\sum_{k=1}^{5} \beta_{sk} F_k\Big) + \sigma_s\sqrt{\Big(1 - \sum_{k=1}^{5} \beta_{sk}^2\Big) G_s}$$

In [43]:
```
eigenvalues[:5].sum()/eigenvalues.sum()
```

Out[43]:
```
0.54983056409691755
```

factor F matrix

In [44]:
```
allocations = []
allocations[:] = [eigenvectors[i]/return_std/np.sqrt(eigenvalues[i]) for i in range(5)]
```

In [45]:
```
allocations = np.array(allocations)
factor_F_matrix = np.dot(allocations,return_matrix)
```

In [46]:
```
factor_F_matrix = factor_F_matrix.T # [n_samples, n_features]
factor_F_matrix.shape
```

Out[46]:
```
(1260, 5)
```

cross-sectional zscore of returns

In [47]:
```
mus = return_matrix.mean(axis=1)
sigmas = return_matrix.std(axis=1)

return_zscore = (return_matrix-mus[:,np.newaxis])/sigmas[:,np.newaxis]
```

In [49]:
```
return_zscore = return_zscore.T # [n_samples, n_targets]
return_zscore.shape
```

Out[49]:
```
(1260, 30)
```

linear regression without fitting the intercept

In [50]:
```
lr = linear_model.LinearRegression(fit_intercept=False)
lr.fit(X=factor_F_matrix, y=return_zscore)
```

Out[50]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
```

In [57]:
```
beta_matrix = lr.coef_ # beta in problem
beta_matrix
```

Out[57]:
```
array([[ 1.56985572,  0.25368053,  0.3925496 ,  0.08232734,  0.44158824],
       [ 1.36109333, -0.28945669, -0.06468587, -0.35824321, -0.2301076 ],
       [ 1.19878756,  0.01891622,  0.07692076, -0.21622567,  0.32766502],
       [ 1.10182079,  0.24588231,  0.19445316, -0.2469598 ,  0.22058351],
       [ 0.35246397,  0.22485324,  0.02582279, -0.50181785,  0.06194982],
       [ 0.56029829, -0.23009169,  0.34339788, -0.12557866,  0.2150055 ],
       [ 0.20677281, -0.24907175,  0.26370521, -0.14449374,  0.23408149],
       [ 0.57931571, -0.0123194 ,  0.07469673, -0.19860754,  0.27708611],
       [ 0.62265921,  0.06558351,  0.16660004, -0.20220961, -0.01075293],
       [ 0.7898533 ,  0.01671887,  0.00472377, -0.20123155,  0.30643163],
       [ 0.03398733,  0.19830815,  0.26069007, -0.47882555,  0.01413778],
       [ 1.43970523,  0.12030461,  0.30543673,  0.00189531,  0.22592013],
       [-0.13063955,  0.05399763,  0.25594416, -0.30789539,  0.39091451],
       [ 1.09613329,  0.10957761,  0.03162746, -0.36291078, -0.05773072],
       [ 0.34395249, -0.27893228, -0.34297894, -0.44313225,  0.44595214],
       [ 0.03913523, -0.16311638, -0.03363578, -0.52669871,  0.03271775],
       [ 1.11614211,  0.12969477,  0.13453344, -0.40070803, -0.19528123],
       [ 1.2741176 , -0.2245349 ,  0.30120976, -0.09502557,  0.11521614],
       [-0.21925635, -0.20130342,  0.28806045, -0.22018512,  0.24476284],
       [ 0.07804731, -0.01694555,  0.74882675, -0.06147783, -0.03503514],
       [ 0.59028017, -0.68830173,  0.1332342 , -0.09400626, -0.13236548],
       [ 0.93054688, -0.08629018, -0.07883992, -0.34671935,  0.30312433],
       [ 0.81550984,  0.11212761,  0.24064302, -0.06145311,  0.35562896],
       [-0.13745344, -0.34470531,  0.42140593, -0.25023126, -0.09345176],
       [ 0.25207553, -0.46555389,  0.07929122, -0.32145945,  0.1816907 ],
       [-0.03483441,  0.1646313 ,  0.21463218, -0.43241803,  0.41209004],
       [ 0.14355039, -0.209275  ,  0.4359626 , -0.25447394,  0.16873041],
       [-0.60106963, -0.40233747,  0.2266469 , -0.16184728,  0.38616018],
       [ 1.77627993, -0.39418229, -0.02470173,  0.00382899,  0.25322973],
       [ 0.15769719,  0.09259879,  0.18875661, -0.27696005, -0.09393466]])
```

In [58]:
```
lr.intercept_
```

Out[58]:
```
0.0
```

r2 score

In [59]:
```
lr.score(X=factor_F_matrix, y=return_zscore)
```

Out[59]:
```
0.42016451976274194
```

In [60]:
```
metrics.r2_score(y_true=return_zscore,y_pred=lr.predict(factor_F_matrix)) # sanity check
```

Out[60]:
```
0.42016451976274194
```

Monte Carlo

In [61]:
```
mus.shape,sigmas.shape,beta_matrix.shape
```

Out[61]:
```
((30,), (30,), (30, 5))
```

In [62]:
```
n_trials = 100000
```

In [63]:
```
Fki = sp.stats.t(df=3.5).rvs((5,n_trials))
Gsi = sp.stats.t(df=3.5).rvs((n_assets,n_trials))
```

The following expression under the square root operator is not always positive:

$$1-\sum_{k=1}^{5}\beta_{sk}^2$$

To make this model valid, one can use absolute value of the above expression

In [64]:
```
(1-(beta_matrix**2).sum(axis=1))
```

Out[64]:
```
array([-1.88467394, -1.1218322 , -0.59748415, -0.42192542,  0.56888441,
        0.45320417,  0.74999525,  0.54244028,  0.53923439,  0.24143545,
        0.66208564, -1.23155943,  0.66689643, -0.34955281,  0.29101942,
        0.69224814, -0.47939487, -0.78682352,  0.72003443,  0.42787298,
        0.1337009 , -0.09167788,  0.1342136 ,  0.61335293,  0.57708272,
        0.56881257,  0.65230693,  0.25015683, -2.37530021,  0.84539742])
```

In [65]:
```
Rsi = mus[:,np.newaxis] + sigmas[:,np.newaxis]*np.dot(beta_matrix,Fki) + sigmas[:,np.newaxis]*\
    np.sqrt(np.abs(1-(beta_matrix**2).sum(axis=1)))[:,np.newaxis]*Gsi
```

In [66]:
```
Ris = Rsi.T
```

In [67]:
```
Ris[0]
```

Out[67]:
```
array([-0.03273784, -0.03039759, -0.02801586, -0.02722335, -0.0038028 ,
       -0.01867527, -0.01293161, -0.00471716, -0.01916345, -0.00344741,
        0.00130724, -0.0183349 , -0.00042342, -0.02335875, -0.040015  ,
       -0.0209574 , -0.01777809, -0.03227339, -0.00805495, -0.03082729,
       -0.0327339 , -0.01984014, -0.02129233, -0.01396883, -0.0168634 ,
       -0.03193198, -0.01277147,  0.00867572, -0.05525569, -0.00304265])
```

In [68]:
```
def calculate_return_tensor(n_trials,n_days):
    R_tensor = list() # [n_days,n_trials,n_assets]
    for i in range(n_days):
        Fki = sp.stats.t(df=3.5).rvs((5,n_trials))
        Gsi = sp.stats.t(df=3.5).rvs((n_assets,n_trials))
        Rsi = mus[:,np.newaxis] + sigmas[:,np.newaxis]*np.dot(beta_matrix,Fki) + sigmas[:,np.newaxis]*\
            np.sqrt(np.abs(1-(beta_matrix**2).sum(axis=1)))[:,np.newaxis]*Gsi
        Ris = Rsi.T # [n_trials, n_assets]
        R_tensor.append(Ris)
    R_tensor = np.array(R_tensor)
    return R_tensor
```

In [69]:
```
return_tensor = calculate_return_tensor(n_trials=10000,n_days=10)
```

In [70]:
```
return_tensor.shape
```

Out[70]:
```
(10, 10000, 30)
```

10 days return

In [71]:
```
return_matrix_10d = (return_tensor+1).prod(axis=0)-1
```

In [72]:
```
return_matrix_10d.shape # [n_samples, n_assets]
```

Out[72]:
```
(10000, 30)
```

10 days return for the portfolio equally weighted in its components

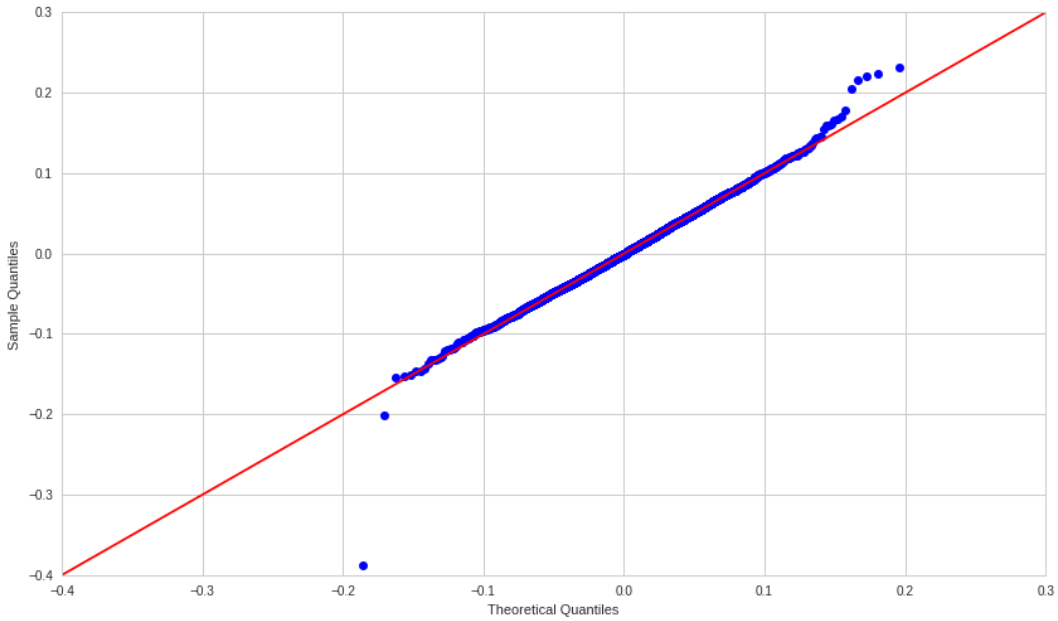
In [73]:
```
return_array_10d = return_matrix_10d.mean(axis=1)
```

QQ plot corresponding to a fitted student-t distribution

In [74]:
```
import statsmodels.api as sm
```

In [75]:
```
df, loc, scale = sp.stats.t.fit(return_array_10d)
```

In [76]:
```
sm.qqplot(return_array_10d, sp.stats.t,distargs=(df,),loc=loc,scale=scale,line='45')
plt.show()
```



From the fitted qq plot, we find that the return distribution is still a student-t distribution

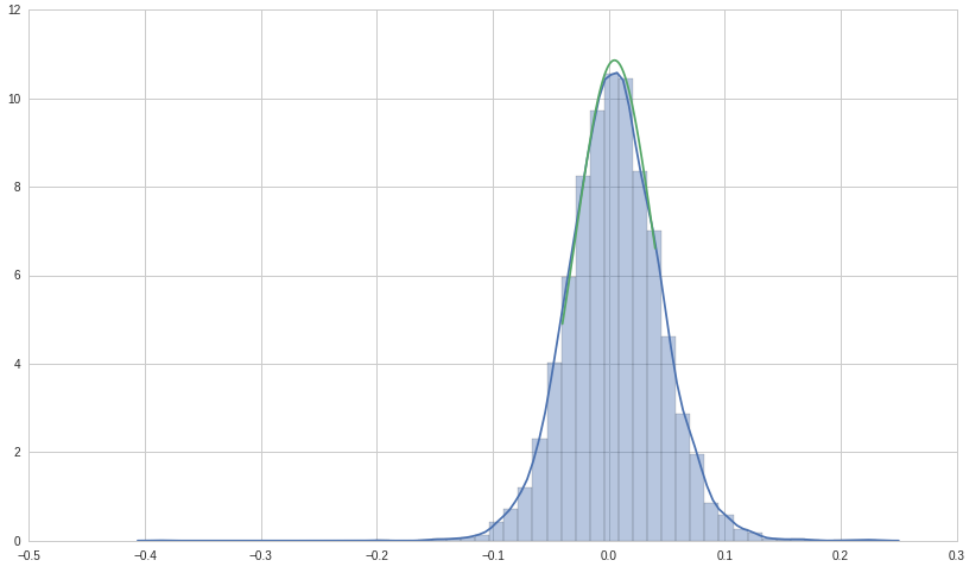
distribution plot

In [77]:
```
import seaborn as sns
```

In [78]:
```
df,loc,scale
```

Out[78]:
```
(11.672261241179042, 0.0049268710816779207, 0.035950094483224407)
```

In [79]:
```
sns.distplot(return_array_10d)
x=np.linspace(-0.04,0.04,100)
plt.plot(x,sp.stats.t(df=df,scale=scale,loc=loc).pdf(x))
plt.show()
```



Based on both qq plot and the distribution plot, the student-t fits very well. In the next part, we use this fitted t distrunution to calculate the value at risk

Calculate the 10-day VaR using fitted student-t distribution

```
In [80]: def VaR_t(confidence_level,df,loc,scale):
             VaR = sp.stats.t(df=df,loc=loc,scale=scale).ppf(1-confidence_level)
             return VaR

In [81]: VaR_t(0.99,df=df,loc=loc,scale=scale)

Out[81]: -0.091863586064685782
```

Calculate the 5-day (one-week) VaR using fitted student-t distribution

```
In [82]: return_array_5d = return_array_10d/2 # one-week return transformed from 10d returns

In [83]: df2, loc2, scale2 = sp.stats.t.fit(return_array_5d)

In [84]: VaR_t(0.99,df=df2,loc=loc2,scale=scale2)

Out[84]: -0.045931763693248473
```

Bonus:

Bonus. Research paper For this problem refer to the paper at: https://www.sciencedirect.com/science/article/pii/S0378437108010479. The paper may be accesed from the Stevens network and a preprint may be found at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2066888. In this paper the authors apply a Re-scaled Range (R/S) method and a De-trended Fluctuation (DFA) method to estimate the degree of long memory effects in high frequency financial data. The study uses High frequency data during a particular day. Repeat the two studied with any day data. Use a Bloomberg terminal to download high frequency data.

The procedure used to calculate R/S is as follows:

1. Let N be the length of time series (y_1, y_2, y_3…, y_N). The logarithmic ratio of the time series is obtained. The length of the new time series M(t) will be N – 1.

$$M(t) = \log\left(\frac{y_{t+1}}{y_t}\right), t = 1,2,....,(N-1)$$

2. The time series is then divided into m sub-series of length n. n represents the number of elements in the series and m represents the number of sub-series. Thus m * n = N -1. Each sub-series can be labeled as Q_a where a = 1, 2,…., m and each element in Q_a can be labeled as $L_{k,a}$ for k = 1,2,….,n.

3. For each Q_a, the average value is calculated:

$$Z_a = \frac{1}{n}\sum_{k=1}^{n} L_{k,a}$$

4. The cumulative deviation in each Q_a is calculated:

$$C_{k,a} = \sum_{j=1}^{k} \left(L_{j,a} - Z_a \right) \quad \text{k} = 1,2,\ldots,\text{n}$$

5. Thus the range of each sub-series Q_a is given as:

$$R(Q_a) = \max\left(C_{k,a}\right) - \min\left(C_{k,a}\right)$$

6. The standard deviation of each sub-series Q_a is calculated:

$$S(Q_a) = \sqrt{\left(\frac{1}{n}\right)\sum_{j=1}^{n} \left(L_{j,a} - Z_a \right)^2}$$

7. Each sub-series is normalized by dividing the range, $R(Q_a)$ by the standard deviation, $S(Q_a)$. The average value of R/S for sub-series of length n is obtained by:

$$\left(R/S\right)_n = \frac{1}{m}\sum_{a=1}^{m} \frac{R(Q_a)}{S(Q_a)}$$

8. Steps 2 through 7 are repeated for all possible values of n, thus obtaining the corresponding R/S values for each n.

The relationship between length of the sub-series, n and the rescaled range R/S is:

$$\frac{R}{S} = (c * n)^H$$

where R/S is the rescaled range, n is the length of the sub-series of the time series and H is the Hurst exponent. Taking logarithms yields:

$$\log\left(R/S\right) = H * (\log n + \log c)$$

9. An ordinary least squares regression is performed using log(R/S) as a dependent variable and log(n) as an independent variable. The slope of the equation is the estimate of the Hurst exponent, H.

Detrended Fluctuation Analysis (DFA)

First, the absolute value of M(t), i.e. the logarithmic returns of the indices calculated in the R/S analysis, is integrated:

$$y(t) = \sum_{i}^{t} \left| M(i) \right|$$

Then the integrated time series of length N is divided into m boxes of equal length n with no intersection between them. As the data is divided into equal-length intervals, there may be some left over at the end. In order to take account of these leftover values, the same procedure is repeated but beginning from the end, obtaining $2N$ / n boxes. Then, a least squares line is fitted to each box, representing the trend in each box, thus obtaining (y_n (t)). Finally the root mean square fluctuation is calculated using the formula:

$$F(n) = \sqrt{\frac{1}{2N} \sum_{t=1}^{2N} \left[y(t) - y_n(t) \right]^2}$$

This computation is repeated over all box sizes to characterize a relationship between the box size n and F (n). A linear relationship between the F(n) and n (i.e. box size) in a log-log plot reveals that the fluctuations can be characterized by a scaling exponent (α),

I only went through the method processes whereas I did not know how to get free High Frequency Data via R or Python, and I did not have access to enter Hanlon Labs during the weekends. So, I did not get the enough time to finish the testing part by those data.