Stocks' Crisis Estimation by Hidden Markov Model

Xinning Liu 110629272
Bo Feng 111132621
Ziheng Wang 111156445

1. Instruction

In order to estimate rare event happens in the stock market, we need to build a model, which follows the same patterns with the real daily return sequence. So, we use mixed Gaussian model and estimate Gaussian mixture density via EM Algorithm. Then we introduce the hidden Markov model, and use an algorithm called Baum Welch algorithm to estimate HMM's parameters. After we have the finite component distribution, we use importance sampling to solve the problem that the possibility of a crisis is too small to be applied to Monte Carlo Method.

2. Feature of stocks daily return

We have learned several portfolio theories, like Markowitz portfolio theory. Typically, we assume the daily return of stocks will have an i.i.d. Gaussian distribution. However, this assumption cannot be set up in the real market.

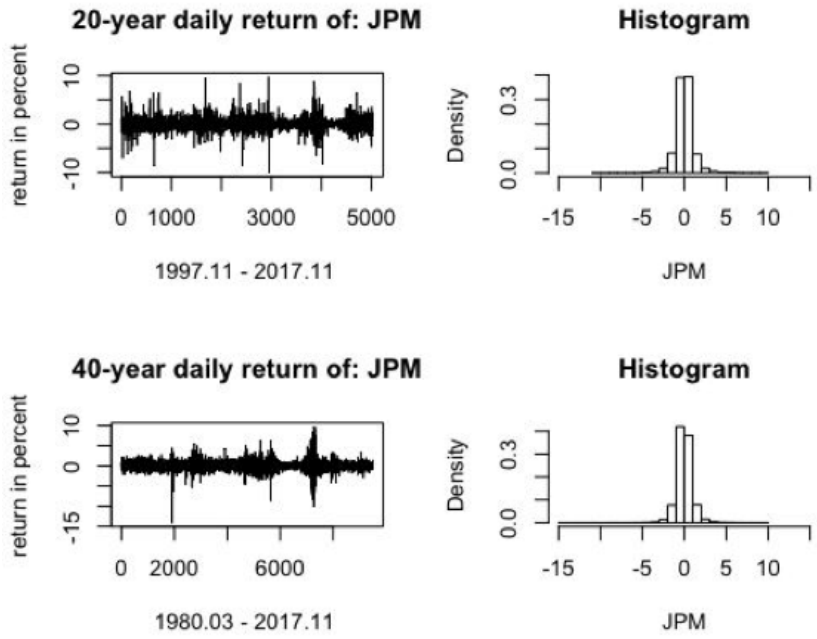

Figure 1 Daily return and histogram of JPM

Figure 1 shows the daily return and histogram of JPM in 20-year and 40-year time spread. We can find that the daily returns tend to be clustered, which means they may not be independent. Form the histogram, we can see the daily returns are not symmetric, and have high kurtosis.

Table 1 Kurtosis and Skewness of stocks daily return (in percentage)

	JPM	USB	WFC	CMA	C	BAC
Kurtosis	12.02	13.57	25.46	11.41	38.22	24.95
Skewness	0.25	-0.06	0.83	-0.22	-0.47	-0.32

Table 1 shows the kurtosis and the skewness of 6 stocks. We can see all of them have very high kurtosis, which means they have fat tail. The skewness also shows that the daily returns are a little asymmetry.

The fat tail property means if we want to find a good fit distribution, Gaussian should not be a good choice. Consider the not very asymmetric property, a t distribution with 4-5 degrees of freedom might be a good choice. However, we want to keep the asymmetric property of the model, and make the calculation not very complicated. So, before we consider the cluster property, Gaussian mixture distribution becomes our choice.

Table 2 shows the p-value of Wald-Wolfowitz runs test shows for most stocks. Even though the observations are not totally random, we can assume the daily returns are independent in the first step.

Table 2 p-value of Wald-Wolfowitz Runs Test

	JPM	USB	WFC	CMA	C	BAC
20 years	0.8657	0.284	0.7932	0.02098	0.7983	0.8988
40 years	0.6638	0.001625	7.532e-06	0.04757	0.2969	0.8215

3. Gaussian Mixture Model (GM)

A Gaussian mixture model has following p.d.f.

$$f(x|\Theta) = \sum_{i=1}^{M} w_i f_i(x|\theta_i)$$

where the parameters $\Theta = (w_1, w_2, \dots, w_M, \theta_1, \theta_2, \dots, \theta_M)$, such that $\sum_{i=1}^{M} w_i = 1$, and each f_i is a Gaussian density function parameterized by θ_i. In other words, a Gaussian mixture model will have M component Gaussian density mixed together with weights w_i.

a. Estimate Gaussian Mixture density via EM Algorithm

A general MLE method to estimate parameters Θ given observation X may have difficulty to compute because of it will contain the log of the sum.

$$\log \mathcal{L}(\Theta|\mathrm{X}) = \log \prod_{j=1}^{N} f(x_j|\Theta) = \sum_{j=1}^{N} \log \sum_{i=1}^{M} w_i f_i(x_j|\theta_i)$$

An idea to solve this problem is using EM algorithm. In EM algorithm, besides X which we can observe, the component number Υ is a hidden variable of X which we want to estimate.

For $(x_j \in \mathrm{X}, y_j \in \Upsilon)$, we have

$$P(y_j = i) = w_i$$
$$f(x_j|y_j = i, \Theta) = f_i(x_j|\theta_i)$$

If we know the value of Υ, the log likelihood will become

$$\log \mathcal{L}(\Theta|\mathrm{X}, \Upsilon) = \sum_{j=1}^{N} \log f_i(x_j|\theta_i) P(y_j = i)$$

According to Bayes' rule, we can estimate $P(y_j = i|x_j, \Theta)$ by

$$P(y_j = i|x_j, \Theta) = \frac{P(y_j = i, x_j|\Theta)}{f(x_j|\Theta)} = \frac{w_i f_i(x_j|\theta_i)}{\sum_{i=1}^{M} w_i f_i(x_j|\theta_i)}$$

The p.m.f of Υ will be

$$p(\Upsilon|\mathrm{X}, \Theta) = \prod_{j=1}^{N} P(y_j|x_j, \Theta)$$

So, the E-step and M-step will be

E-step

$$Q(\Theta|\Theta^g) = \mathrm{E}^{\Upsilon}[\log \mathcal{L}(\Theta|\mathrm{X}, \Upsilon)|\mathrm{X}, \Theta^g]$$

M-step

$$\Theta^{g+1} = \arg\max_{\Theta} Q(\Theta|\Theta^g)$$

b. Result Analysis

For the 6 stocks we analyzed, most of them (except USB) has 3 components. The component which has extreme expectation (positive and negative) will have large variance (>8%), and the component which has relative small positive expectation will have small variance (>0.2%). The large variance components have very small weight (<0.1), which means they are rarely happened.

USB is a little different. It has 3 components in recent 20 years, but 4 components in 40 years.

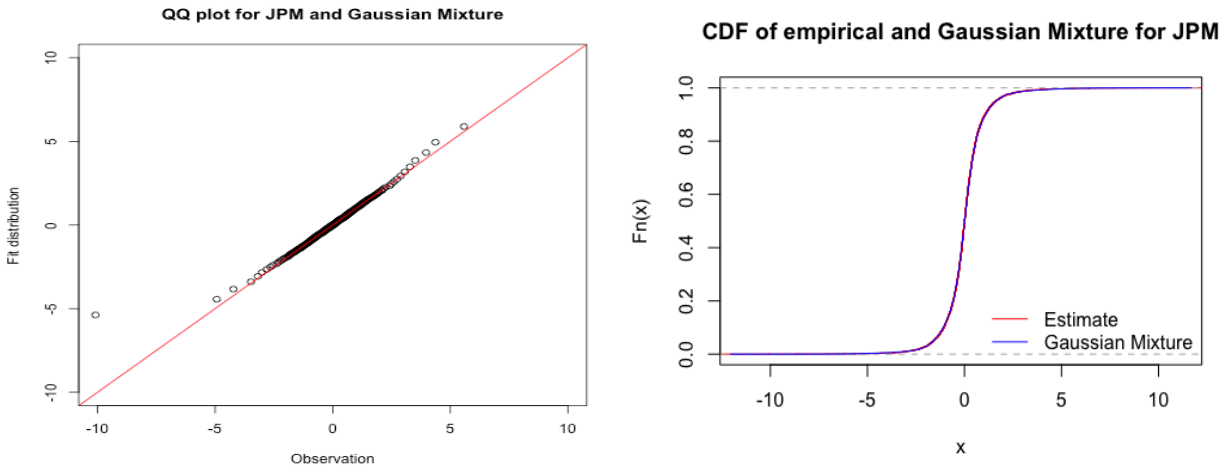

Figure 2 QQ-Plot and CDF Comparison for JPM 20-year Observations

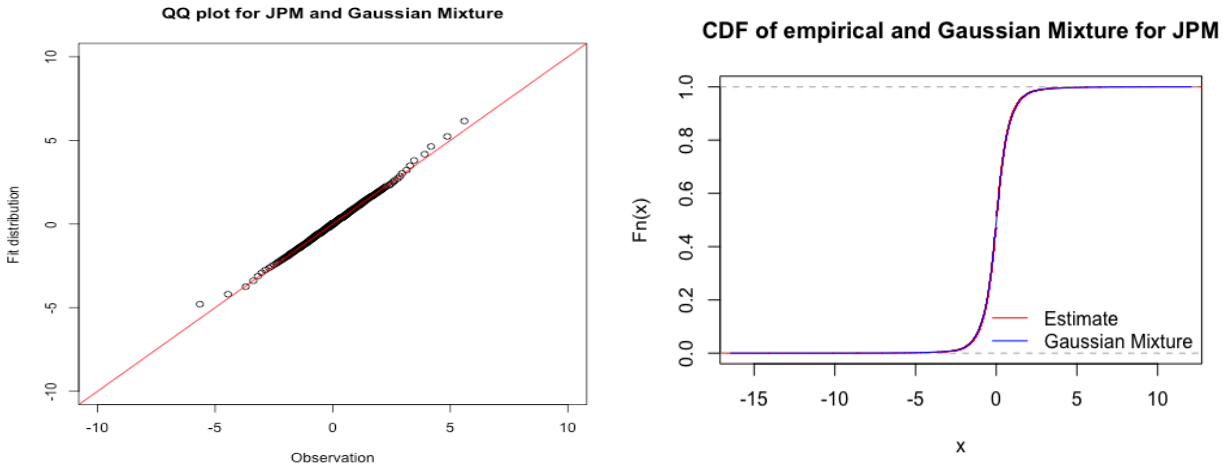

Figure 3 QQ-Plot and CDF Comparison for JPM 40-year Observations

Table 3 p-value of Goodness of fit χ^2 test

	JPM	USB	WFC	CMA	C	BAC
20 years	0.5422	0.2436	0.6962	0.9868	0.01834	0.9539
40 years	5.669e-12	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16	< 2.2e-16

Figure 2 and Figure 3 are QQ-plot and empirical and Gaussian Mixture c.d.f comparison. Table 3 shows the p-value of goodness of fit test. We can say Gaussian Mixture modal can catch the feature of the market. However, for a very long-range observation like 40 years, the model does not work well. It is possible because the market does not enter the relative steady stage.

4. Dependency of daily returns and Hidden Markov Model

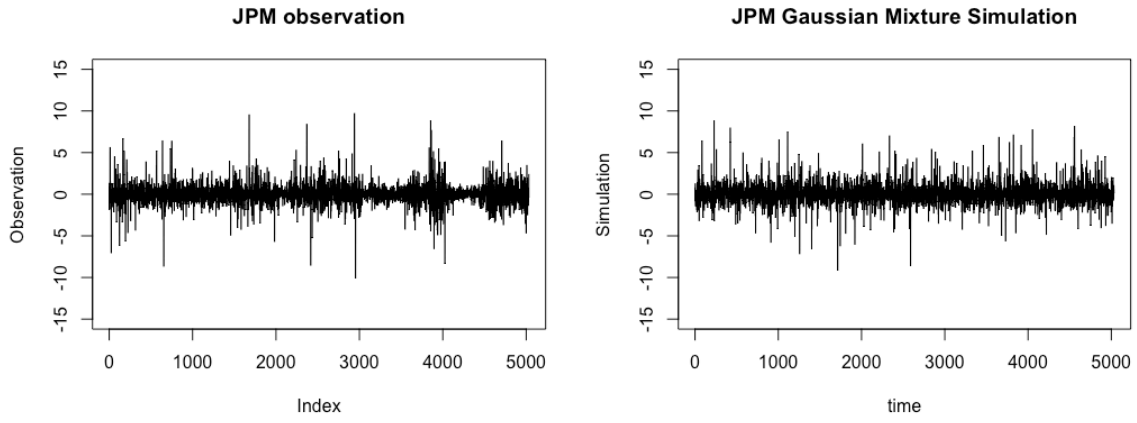

Figure 4 Real data and Gaussian Mixture simulation for JPM 20-year Observations

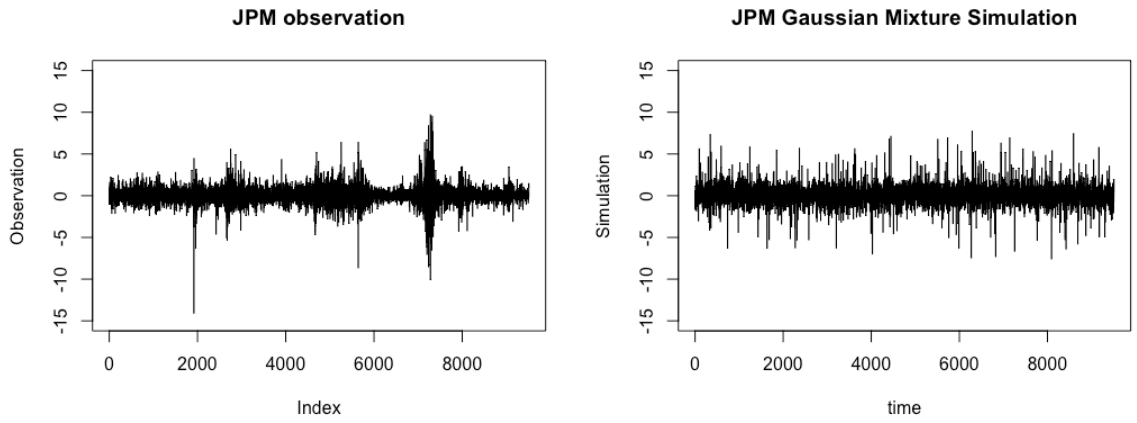

Figure 5 Real data and Gaussian Mixture simulation for JPM 40-year Observations

Even though the Gaussian Mixture model may have a good goodness of fit test result, we still cannot say the model have caught all of the features the daily returns have.

Figure 4 and Figure 5 are the comparison of the real observation and Gaussian mixture simulation. We can see the real observations tend to be clustered. Some days may have a relative large volatility, and following days may have a relative small volatility. However, in Gaussian mixture model, the simulated sequences are more random. There is no obvious cluster property in Gaussian mixture model.

Because of the cluster property, now, we consider possibly daily return not only depends on its own randomness, it also depends on the information of the day before. We induce Hidden Markov Model into Gaussian Mixture.

a. Hidden Markov Model (HMM)

In a hidden Markov model, we assume for each observation x which we can observe, it has a hidden state s. The observation's distribution depends on hidden state. The sequence of hidden states has Markov property, which makes the sequence a Markov chain, i.e.

$$P(s_j | s_{j-1}, s_{j-2}, \dots, s_1) = P(s_j | s_{j-1})$$

A hidden Markov model will have a hidden state transition matrix A, an initial state π at time $t = 1$, and an emission parameter matrix which contains the parameters of the observation's density conditioned on hidden states.

In a hidden Markov Gaussian mixture (HMMGM) model, the observation's distribution conditioned on hidden state is still a Gaussian mixture.

b. Algorithm to estimate HMM

An algorithm called Baum–Welch algorithm can be used to estimate HMM's parameters. It is still an EM algorithm. The basic idea is to find the expected number of each state for observation sequence and the expected number of transitions between each pair of states. For details, see J. A. Bilmes ,"A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixed and Hidden Markov Models", U.C. Berkely, TR-97-021, April 1998.

The Gaussian mixture model can be seen as the steady distribution of HMM, and we can use the parameter of the Gaussian mixture to initialize the parameter of HMM to make the algorithm converge faster.

After estimation of HMM parameters, we can use Viterbi algorithm (a dynamic programming algorithm) to decode the observations for state sequence (which called Viterbi path).

c. Result Analysis

We simulate a general HMM sequence, and also a sequence under Viterbi path.

Figure 6 shows the general HMM simulation, we can find the sequence has very weak cluster property. The 40-years version has relative clear cluster property. Figure 7 and **Error! Reference source not found.** show the comparison of the real data and the Viterbi path simulation. We can see the same volatility pattern in the simulation and the real data.

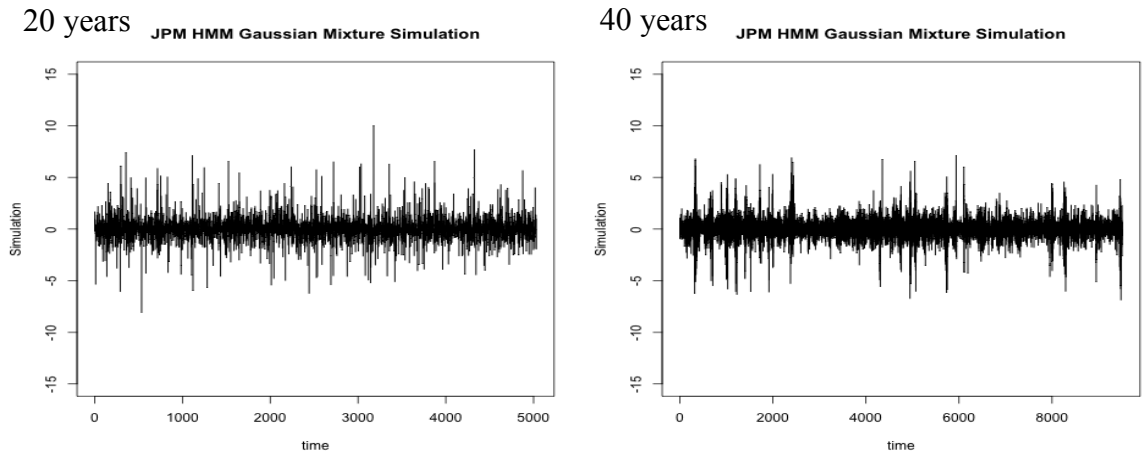

Figure 6 General HMM simulation for JPM

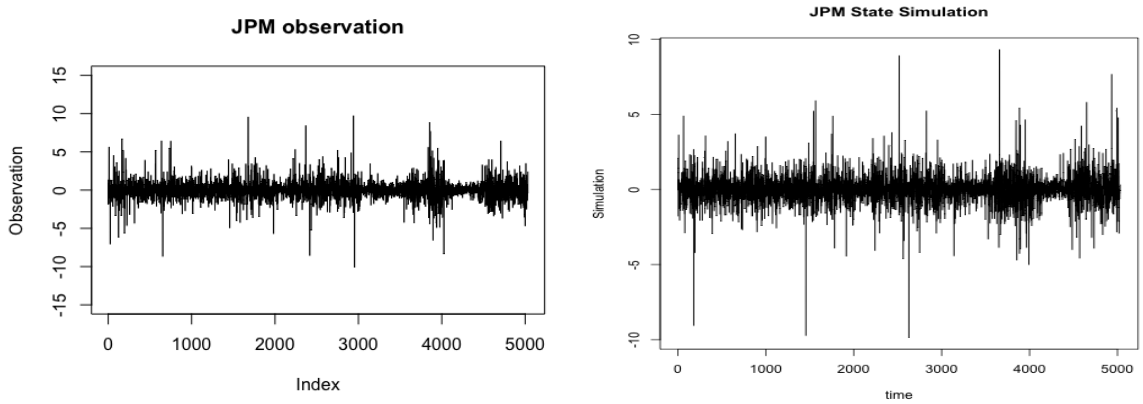

Figure 7 Comparison of real data and Viterbi Path Simulation for JPM 20-year observation

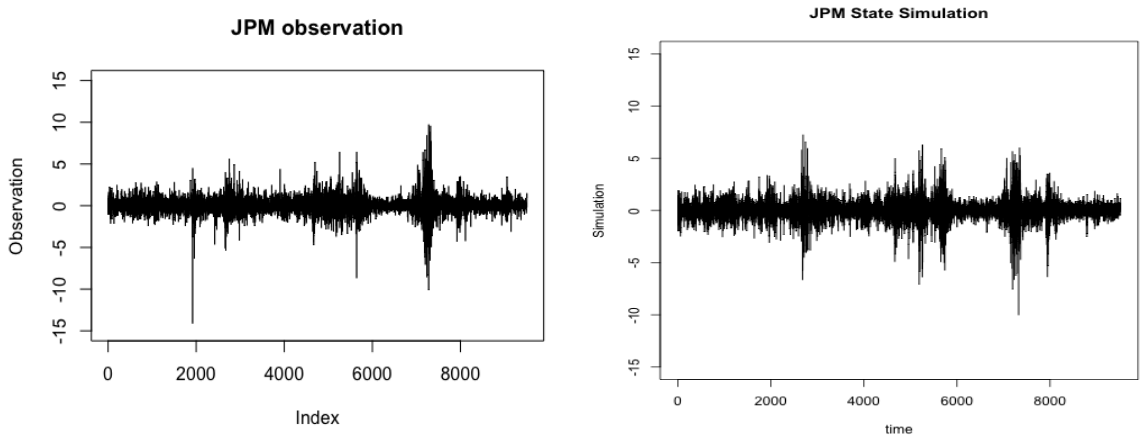

Figure 8 Comparison of real data and Viterbi Path Simulation for JPM 40-year observation

Because the general simulation does not show a clear cluster property but a special path does catch the patter of the real data, we think more information to be considered may improve the model.

5. Two-day dependency HMM

In a two-day dependency HMM, the transaction of the hidden state will depend on the information of two days before, i.e.,

$$P(s_j|s_{j-1}, s_{j-2}, \dots, s_1) = P(s_j|s_{j-1}, s_{j-2})$$

Following is an example for 2 component Gaussian mixture:

Transition Matrix A:

$$\begin{bmatrix} p_1 & p_2 \\ p_1 & p_2 \end{bmatrix} \Longrightarrow \begin{bmatrix} p_1 & p_2 & 0 & 0 \\ 0 & 0 & p_1 & p_2 \\ p_1 & p_2 & 0 & 0 \\ 0 & 0 & p_1 & p_2 \end{bmatrix}$$

Component Distribution:

$$(\mathrm{f}_1 \quad \mathrm{f}_2) \Longrightarrow (\mathrm{f}_1 \quad \mathrm{f}_2 \quad \mathrm{f}_1 \quad \mathrm{f}_2)$$

Initial State π:

$$(\mathrm{p}_1 \quad \mathrm{p}_2) \Longrightarrow (\mathrm{p}_1/2 \quad \mathrm{p}_2/2 \quad \mathrm{p}_1/2 \quad \mathrm{p}_2/2)$$

The dimension of HMM parameters squared.

a. Result Analysis

Figure 9 shows the two-day dependency general HMM simulation. We can see very strong cluster property in both 20-year and 40-year version. The 40-year one is more convincing.

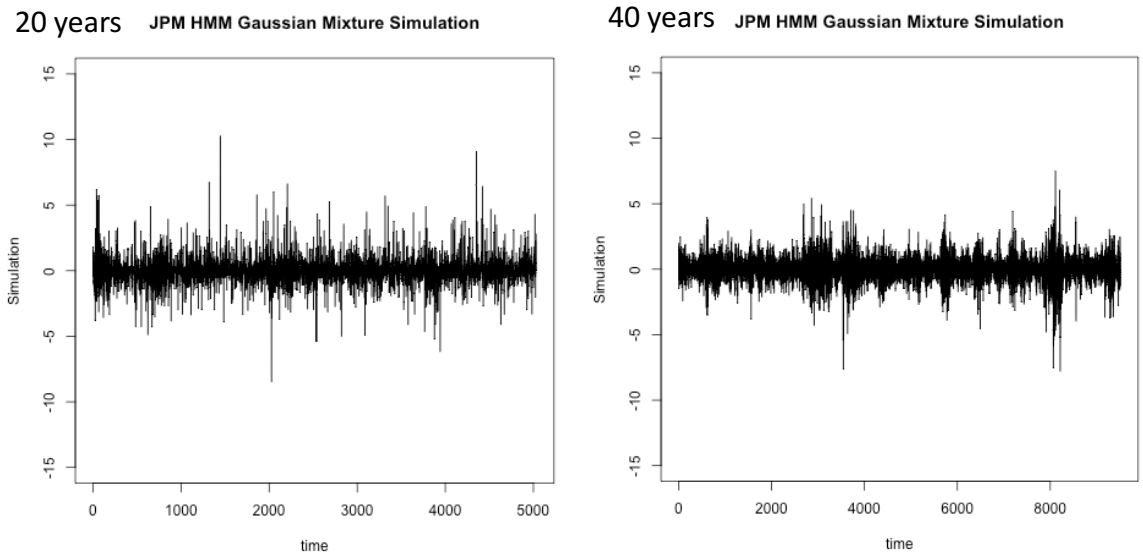

Figure 9 Two-day dependency general HMM simulation

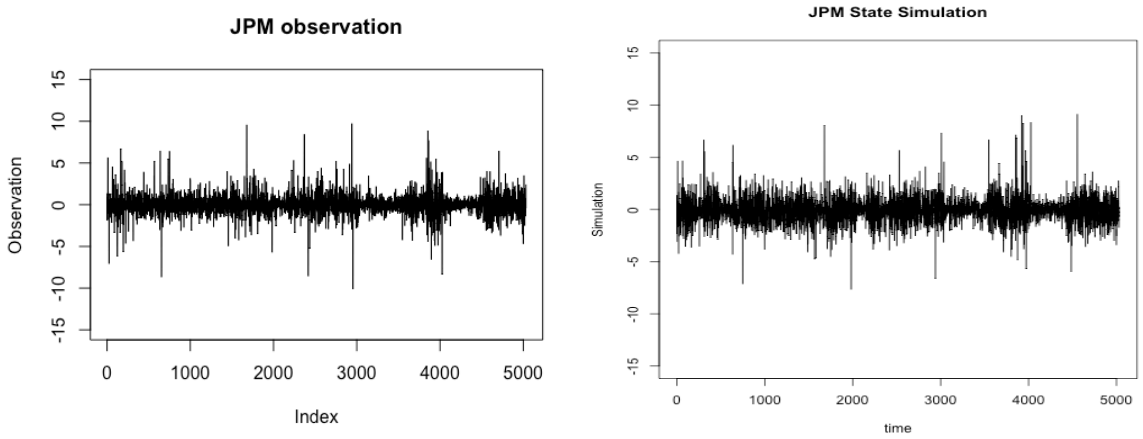

Figure 10 Two-day dependency Viterbi Path 20-year

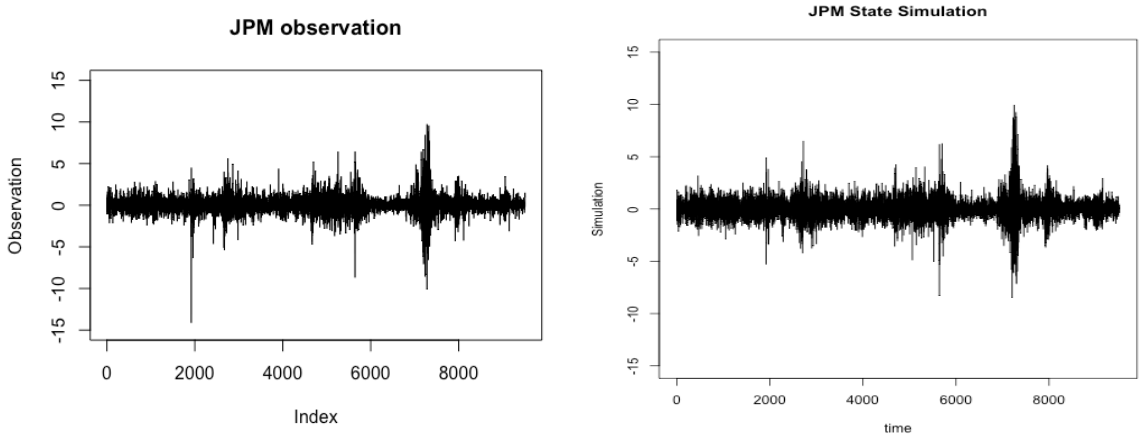

Figure 11 Two-day dependency Viterbi Path 40-year

Both two-day dependency Viterbi path simulation work better than their one-day dependent version. We can see a similar volatility pattern from the simulation path.

6. Other Information of HMM Simulation

We did GM, HMM one-day dependency, HMM two-day dependency for all 6 stocks. Most of 40-year data have better result except UBS. The 40-year data of UBS does not converge. The minimum error we can get is 0.1.

7. Find the Component Distribution

Based on the work above, we can get the transit matrix for each stock.

e.g. The Transit Matrix of JPM using 40-year data:

0.6364286	0.3635714	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.9651091	0.0181278	0.0167631	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.9203139	0.0632259	0.0164602
0.3118312	0.6855170	0.0026518	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0195438	0.9804562	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0213778	0.9770244	0.0015977
0.0000000	0.0000000	1.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0262417	0.0000000	0.9737583	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0524511	0.0000000	0.9475489

The first day hidden state is “State 12”. The last day hidden state according to Viterbi algorithm is “State 11”.

Since we have three different states each day, based on the last two days, we may have nine distinct situations. For each situation, we multiply with the transit matrix and get the component distribution of the following day, which is used for predicting the returns.

8. Importance sampling

Since important sampling is a general technique for estimating properties of a particular distribution, we use importance sampling technique to solve the problem which the probability of a crisis is too small to be applied to Monte Carlo Method.

We use Gaussian distribution as the importance sampling distribution and set the number of iterations to be 10 000 00. By estimating the confidence interval, we calculate an accuracy, which is 10^-174.

9. Conclusion

Table 4 shows on the probability of JPM return below -20% under HMM model (columns represent state number of two days before; rows represent state number of one day before):

Table 4 The probability of JPM has a return below 20%

	1	**2**	**3**

1	0	1.14902446e-080	1.23015758e-003
2	4.58409077e-014	0	1.48792469e-011
3	7.66936819e-012	2.58231679e-078	6.56932562e-002

Since the state of 11.28 and 11.29 are all number 1, based on the method we used, we can deduce that the probability of 11.30 having crisis tends to be 0. The real return of JPM on 2017 .11.20 is 0.76%. Our model makes sense.

Based on the results we get, HMM model appears to be relatively effective to be used fitting the daily return data. For further study, we are considering other situations such as -10% and -15% daily return. Also, we are looking for a way that can prove the model is effective or not technically.

10. Reference

J. A. Bilmes ,"A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixed and Hidden Markov Models", U.C. Berkely, TR-97-021, April 1998.

11. Project Code

a. R code of HMM and GM model

```
library(mclust)
library(RcppHMM)
library(e1071)
library(randtests)
MclustAnalysis<-function( Observation, g, name){
 starttime=proc.time()

 Dens <- densityMclust ( Observation , G = c(2:4))

 ng=10
 Qt   <- quantileMclust( Dens, seq(0,1,by=1/ng)[-1])

 a    <- sort ( c( Qt, Observation), index.return=TRUE)
 o    <- match( c(1:10), a$ix)
 ob   <- o - c(0,o[-10]) - 1
 GoodnessFit <- chisq.test(ob,p=rep(0.1,10))
 print(GoodnessFit)

 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"qqplot.png"))

 ng=length(Observation)%/%10
 Qt   <- quantileMclust( Dens, seq(0,1,by=1/ng)[-1])

 qqplot( Observation, Qt,
       main = paste( "QQ plot for", name, "and Gaussian Mixture"),
       ylab="Fit distribution",
       xlim=c(-10,10),
       ylim=c(-10,10)
 )
 dev.off()

 fn<-ecdf(Observation)
 cdf<-cdfMclust(Dens)

 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"cdfplot.png"))

 plot(fn,col=rgb(1,0,0), main=paste("CDF of empirical and Gaussian Mixture for",name))
 lines(cdf,col=rgb(0,0,1))
 legend("bottomright",
      legend = c("Estimate", "Gaussian Mixture"),
      col = c(rgb(1,0,0),rgb(0,0,1)),
      lty=c(1,1),
      bty='n')
 dev.off()

 print(paste("Fitting Gaussian Mixture use",(proc.time()-starttime)[3],"sec."))
```

```
 dens<-c()
 dens$pro = Dens$parameters$pro
 dens$mean = as.vector(Dens$parameters$mean)
 if (Dens$parameters$variance$modelName=="E"){
  dens$variance = rep(Dens$parameters$variance$sigmasq,3)
 }else{
  dens$variance = Dens$parameters$variance$sigmasq
 }
 dens$G=Dens$G
 dens
}

plotModelSimulation.GM <- function(Observation, parameters, g, name){
 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"Observation.png"))
 plot(Observation,type='l', main = paste(name,"observation"),ylim=c(-15,15))
 dev.off()

 starttime=proc.time()

 n = length(Observation)
 a=c()
 for(i in 1:g){
  mean=parameters$mean[i]
  sd=sqrt(parameters$variance[i])
  a[(n*(i-1)+1):(n*i)]=rnorm(n,mean,sd)
 }
 a=matrix(a,ncol = g)
 s=sample(g,size=n,replace = TRUE, prob = parameters$pro)
 ob=c()
 for(j in 1:n){
  ob[j]=a[j,s[j]]
 }

 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"GMSimulation.png"))

 plot(ob,type='l',main=paste(name,"Gaussian Mixture Simulation"),xlab="time",ylab="Simulation",ylim=c(-15,15))
 dev.off()

 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"GMHistogram.png"))

 hist(ob,breaks=20,freq=FALSE,main=paste(name,"GM Simulation Histogram"),xlab=name,xlim=c(-15,15))
 dev.off()

 print(paste("GM Simulation, Kurtosis=",kurtosis(ob),"; Skewness=", skewness(ob),sep=""))

 rt<-runs.test(ob, plot=TRUE)
 print(paste(name,"GM Simulation Wald-Wolfowitz Runs Test:"))
 print(rt)


 print(paste("Simulating Gaussian Mixture use",(proc.time()-starttime)[3],"sec."))

}
```

```
HMMGMAnalysis<-function(Observation, parameters, g){
 starttime=proc.time()

 N=c()
 for (i in 1:(g*g)){
  N[i]=paste("State ",(i+g-1)%/%g,(i-1)%%g+1,sep="")
 }
 A <- matrix(rep(0,g^4),ncol=g*g)
 for (i in 1:g){
  for(j in 0:(g-1)){
   A[i+j*g,(i*g-g+1):(i*g)]=parameters$pro
  }
 }
 Mu<- matrix(rep(parameters$mean,g),ncol=g*g)
 Sigma <- array(rep(parameters$variance,g), dim = c(1,1,length(N)))
 Pi <- rep(parameters$pro, g)/g
 HMM.cont.univariate <- verifyModel(list( "Model"="GHMM",
                                "StateNames" = N,
                                "A" = A,
                                "Mu" = Mu,
                                "Sigma" = Sigma,
                                "Pi" = Pi))
 observationSequences = array(Observation,c(1,length(Observation),1))
 HMM.cont.univariate <- learnEM(HMM.cont.univariate,
                        observationSequences,
                        iter= 5000,
                        delta = 1E-10,
                        print = FALSE)
 #  print(HMM.cont.univariate)
 print(paste("Fitting Two days HMM Gaussian Mixture use",(proc.time()-starttime)[3],"sec."))


 HMM.cont.univariate
}

plotModelSimulation.HMM.GM <- function(Observation, hmm, g, name){
 #  plot(Observation,type='l', main = paste(name,"observation"))

 starttime=proc.time()

 n = length(Observation)

 observationSequence <- generateObservations(hmm, n)

 ob=observationSequence$Y[1,]
 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"HMMSimulation.png"))
 plot(ob,type='l',main=paste(name,"HMM Gaussian Mixture Simulation"),xlab="time",ylab="Simulation",ylim=c(-
15,15))
 dev.off()


 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"HMMHistogram.png"))
```

```
 hist(ob,breaks=20,freq=FALSE,main=paste(name,"HMM Simulation Histogram"),xlab=name,xlim=c(-15,15))
 dev.off()

 print(paste("HMM Simulation, Kurtosis=",kurtosis(ob),"; Skewness=", skewness(ob),sep=""))
 rt<-runs.test(ob, plot=TRUE)
 print(paste(name,"HMM Simulation Wald-Wolfowitz Runs Test:"))
 print(rt)

 print(paste("Simulating HMM Gaussian Mixture use",(proc.time()-starttime)[3],"sec."))

}

plotStateSimulation<-function(Observation,hmm,g,name){
 starttime=proc.time()
 hiddenStates <- viterbi(hmm, matrix(Observation,ncol=length(Observation)) )
 print(paste(name,"last state is",hiddenStates[length(hiddenStates)]))
 mean=hmm$Mu
 sd=sqrt(hmm$Sigma)
 n=length(Observation)
 a<-c()
 for(i in 1:(g*g)){
  a[(n*(i-1)+1):(n*i)]=rnorm(n,mean[i],sd[i])
 }
 a=matrix(a,ncol = g*g)
 ob=c()
 for(j in 1:n){
  s=as.numeric(strsplit(hiddenStates[j],' ')[[1]][2])
  s1=s  %% 10
  s2=s %/% 10
  ob[j]=a[j,s1+(s2-1)*g]
 }

 png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"StateSimulation.png"))

 plot(ob,type='l',main=paste(name,"State Simulation"),xlab="time",ylab="Simulation",ylim=c(-15,15))
 dev.off()


png(filename=paste("/Users/liuxinning/Documents/R/TwoDayDependent20y/",name,"StateSimulationHistogram.
png"))

 hist(ob,breaks=20,freq=FALSE,main=paste(name,"State Simulation Histogram"),xlab=name,xlim=c(-15,15))
 dev.off()

 print(paste("State Simulation, Kurtosis=",kurtosis(ob),"; Skewness=", skewness(ob),sep=""))
 rt<-runs.test(ob, plot=TRUE)
 print(paste(name,"State Simulation Wald-Wolfowitz Runs Test:"))
 print(rt)

 print(paste("State Simulation, Kurtosis=",kurtosis(ob),"; Skewness=", skewness(ob),sep=""))
 print(paste("State Simulation use",(proc.time()-starttime)[3],"sec."))
```

```
  hiddenStates
}


library(readr)
DailyReturn <- read_csv("~/Documents/R/AMS 553/DailyReturnPercent.csv")[2:7]
#source("/Users/liuxinning/Documents/R/HMMGaussianMixturegg.R")
parameter<-c()
name<-names(DailyReturn)
hmm<-c()
path<-c()
g=3

sink("/Users/liuxinning/Documents/R/TwoDayDependent20y/TwoDayDependent20y.output.txt")
for (i in 1:6) {

  print(paste("Analysing ",name[i]))

  # estimate Gaussian Mixture parameter
  t=MclustAnalysis(DailyReturn[[i]], g, name[i])
  if(g!=t$G){
    g=t$G
  }

  parameter[[i]]<-t

  print("Gaussian Mixture parameter:")
  print(parameter[[i]])

  # Simulate GM sequence
  plotModelSimulation.GM(DailyReturn[[i]], parameter[[i]], g, name[i])

  # estimat Hidden Markov model two day dependency model
  hmm[[i]]<-HMMGMAnalysis(DailyReturn[[i]],parameter[[i]],g)

  hmmp<-c()
  hmmp$StateNames=hmm[[i]]$StateNames
  hmmp$A=round(hmm[[i]]$A,digits = 7)
  hmmp$Mu=as.vector(hmm[[i]]$Mu)
  hmmp$Sigma=as.vector(hmm[[i]]$Sigma)
  hmmp$Pi=round(as.vector(hmm[[i]]$Pi),digits = 7)

  print("HMM parameter:")
  print(hmmp)

  # simulate HMM sequence
  plotModelSimulation.HMM.GM(DailyReturn[[i]], hmm[[i]], g, name[i])

  # simulate HMM sequence by Viterbi path
  path[[i]]<-plotStateSimulation(DailyReturn[[i]], hmm[[i]], g, name[i])

}
```

```
sink()
```

b. Python code of importance sampling

```
import numpy as np
import scipy.stats as stats

A_JPM = np.matrix('0.9657182 0.0342818 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
0.0000000;\
        0.0000000 0.0000000 0.0000000 0.7501299 0.2454169 0.0044532 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.8434548 0.1385754 0.0179699;\
        0.0589979 0.9352595 0.0057426 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.4063575 0.5936425 0.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0863012 0.91369881 0;\
        0.0389484 0.0004351 0.9606165 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0398372 0.0000000 0.9601628')


mu = np.array([0.031575586, 0.028980357, 0.041507030, 0.056480484, 0.005618569, -0.068715217,
0.159396639, -0.089861975, -0.073997683])
sigma = np.array([0.68730315, 0.20264853, 2.97403890, 0.35474083, 0.04872352, 1.06597064, 3.21149282,
0.97471365, 13.39634333])
pi = np.eye(9)

n = 10000000

p = pi*(A_JPM)

p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt

print("JPM: "+str(p_is.sum(axis=1)))




A_USB = np.matrix('0.2939406 0.0003574 0.1928870 0.0002229 0.0091818 0.0437886 0.1797716 0.0191264
0.2607237;\
        0.0000367 0.9197353 0.0001424 0.0753120 0.0000327 0.0000867 0.0001418 0.0000241 0.0044882;\
        0.2698988 0.0000648 0.1094652 0.0023074 0.0512594 0.0826403 0.0452535 0.0128595 0.4262510;\
        0.0003566 0.0093053 0.0004843 0.9588932 0.0013987 0.0003923 0.0009083 0.0282583 0.0000031;\
        0.2782404 0.0006043 0.2191391 0.0099917 0.0857565 0.0400363 0.1435159 0.0061969 0.2165188;\
```

```
        0.2692504 0.0009514 0.3143754 0.0058223 0.0492275 0.0732002 0.0615581 0.0195037 0.2061110;\
        0.2006581 0.0019845 0.11541223 0.1942252 0.1675143 0.0502872 0.0429915 0.2133859 0.0135411;\
        0.1701886 0.0108061 0.0365144 0.4938982 0.0120236 0.0287503 0.1361841 0.0581136 0.0535211;\
        0.0682694 0.0000011 0.1700355 0.0000436 0.0616259 0.0626160 0.0543085 0.0009159 0.5821842')



mu = np.array([0.077900187, -0.022377402, 0.003256698, 0.005190826, 0.050693413, 0.005980560,
0.053217457, 0.578735921, 0.015151893])
sigma = np.array([0.27134266, 6.93704024, 0.17078143, 0.74594339, 0.32949119, 0.27884051, 0.32524088,
1.25959727, 0.04825162])
pi = np.eye(9)

n = 10000000

p = pi*(A_USB)

p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt

print("USB: "+str(p_is.sum(axis=1)))



A_WFC = np.matrix('0.9600380 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0399620
0.0000000;\
        0.0000000 0.9375777 0.0144577 0.0327731 0.0151916 0.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.9370599 0.0000000 0.0000000 0.0095318 0.0534083 0.0000000 0.0000000;\
        0.0000000 0.9047865 0.0000000 0.0158120 0.0000000 0.0000000 0.0282349 0.0000000 0.0511667;\
        0.0000000 0.0000000 0.0000000 0.0261207 0.4548316 0.5190477 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0610830 0.9334754 0.0054416 0.0000000 0.0000000 0.0000000;\
        0.0015467 0.0000000 0.1244242 0.0000000 0.0000000 0.0000000 0.8690841 0.0049450 0.0000000;\
        0.0075333 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0118735 0.9805933 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0038521 0.0000000 0.0012291 0.0000000 0.9949188')


mu = np.array([0.023569313,  0.006107293,  0.023782419,  1.053819973, -0.172232188,  0.274614703,
0.031158377, -0.007053175, 0.021166087])
sigma = np.array([15.8403492,  0.1534832,  0.4645961,  0.1568297,  0.3621988,  0.1366332 , 1.2680424,
1.6278268,  0.1063219])
pi = np.eye(9)

n = 10000000

p = pi*(A_WFC)
```

```
p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt

print("WFC: "+str(p_is.sum(axis=1)))




A_CMA = np.matrix('0.4312561 0.0854925 0.0000698 0.3918164 0.0001367 0.0425006 0.0000386 0.0201950
0.0284942;\
            0.3161369 0.1570421 0.0000594 0.3212551 0.0059859 0.0725317 0.0001102 0.0668124 0.0600662;\
            0.1689626 0.1014868 0.0025054 0.2451069 0.1720345 0.0790196 0.0050435 0.1442913 0.0815492;\
            0.5875349 0.1194199 0.0000402 0.1434014 0.0110836 0.0493741 0.0000279 0.0229669 0.0661510;\
            0.4470892 0.1773326 0.0001076 0.1490812 0.0525756 0.0741301 0.0001342 0.0519151 0.0476344;\
            0.4872252 0.0845524 0.0001190 0.2193071 0.0931333 0.0278275 0.0001088 0.0542585 0.0334683;\
            0.2147179 0.0592593 0.0174562 0.3497221 0.1710520 0.0323956 0.0008491 0.1364981 0.0180497;\
            0.6284352 0.0083542 0.0001759 0.0767045 0.0439892 0.0731218 0.0000845 0.0528383 0.1162964;\
            0.4250688 0.0641847 0.0000495 0.2866729 0.0996912 0.0311377 0.0000348 0.0348730 0.0582875')


mu = np.array([0.0159257347,  0.0292217002, -1.5549132082,  0.0271045731,  0.0353345102,  0.0003988441, -
3.4051045519,  0.0235616800,0.0173382703])
sigma = np.array([0.6452458,  0.5110855, 21.7062039,  0.4214157,  0.9050641,  1.8290470 , 5.8750329,
2.3770424 , 1.5001398])
pi = np.eye(9)

n = 10000000

p = pi*(A_CMA)

p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt

print("CMA: "+str(p_is.sum(axis=1)))
```

```
A_C = np.matrix('0.6437583 0.0617071 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
0.2945346;\
        0.0061624 0.8524366 0.0000000 0.0000000 0.0007212 0.1406797 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.9639721 0.0251845 0.0108434 0.0000000 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0054253 0.9815974 0.0005839 0.0000000 0.0123933 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0609446 0.0000000 0.9302989 0.0000000 0.0000000 0.0087565 0.0000000;\
        0.0000000 0.1090015 0.0000000 0.0000000 0.0000000 0.8909985 0.0000000 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0005508 0.2470749 0.6857157 0.0666586;\
        0.0044458 0.0015402 0.0000000 0.0000000 0.0000000 0.0000000 0.0995540 0.8225333 0.0719266;\
        0.0078504 0.0000000 0.0000000 0.0118492 0.0018659 0.0000000 0.1717383 0.0000000 0.8066961')


mu = np.array([0.460823585,  0.032787912,  0.004582427, -0.025534228, -0.834778874,  0.005380753,
0.861822396, -0.077684587, -0.162082915])
sigma = np.array([4.96551471,  0.44603655,  4.15447248,  1.43701856, 36.81525156,  0.09817246,  0.48395034,
0.29145849,  0.83967158])
pi = np.eye(9)

n = 10000000

p = pi*(A_C)

p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt

print("C: "+str(p_is.sum(axis=1)))




A_BAC = np.matrix('0.9670135 0.0000000 0.0329865 0.0000000 0.0000000 0.0000000    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.9665112 0.0000000 0.0334888    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000    1 0.0000000 0.0000000;\
        0.0022293 0.9557165 0.0420542 0.0000000 0.0000000 0.0000000    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.9940477 0.0059523    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000    0 0.8605537 0.1394463;\
        0.0111351 0.0956026 0.8932622 0.0000000 0.0000000 0.0000000    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0115149 0.0023040 0.9861811    0 0.0000000 0.0000000;\
        0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000    0 0.3524609 0.6475391')


mu = np.array([-0.24961936,  0.03976593, -0.13224166, -0.13034515,  0.02174669,  0.01002080,  0.15869354,
0.04861858,  0.12816279])
```

```
sigma = np.array([23.2065944,  0.9067232,  2.5132022,  0.9190294,  0.1149957,  0.2674167,  2.5216932,
0.2619554,  1.2810032])
pi = np.eye(9)

n = 10000000

p = pi*(A_BAC)

p_is = np.eye(9)
cv = -20
for i in range(0,p.shape[0]):
   for j in range(0,p.shape[1]):
      y = stats.norm(cv,sigma[j]).rvs(int(n))
      y = y[y<cv]
      pt = (1.0/n) * (p[i,j] * stats.norm(mu[j],sigma[j]).pdf(y)/stats.norm(cv,sigma[j]).pdf(y)).sum()
      p_is[i,j] = pt


print("BAC: "+str(p_is.sum(axis=1)))
```