FE621 HW4

Shihao Zhang

2018-12-05

Problem 1.Comparing different Monte Carlo schemes (a)

```
#To solve the problem, use equation (4.2) to (4.10) in textbook(CS)
#and follow the pseudo-code in page 85 to implement Monte Carlo schemes
set.seed(1)
OptionMC <- function(iscall,K,Tm,S0,sig,r,div,N,M){
  begintime<-Sys.time()
  #precompute constants
  dt <- Tm/N
  nudt <- (r-div-0.5*sig^2)*dt
  sigsdt <- sig*sqrt(dt)
  InS <- log(S0)
  if(iscall=="call"){cp <- 1} ##distinguish call and put option
  if(iscall=="put"){cp <- (-1)}

  sumCT1 <- sumCT2 <- 0
  InSt<-CT<-matrix(nrow = 1,ncol = M) #initialize the matrix

  for (j in 1:M) { #for each simulation
    InSt[j] <- InS
    CT[j] <- 0
  }

  #break the loop to avoid for loop
  #Using random variables(epsilon) to simulate M tirals of InSt value at final time T
  for (i in 1:N) {
    epsilon<-rnorm(M)
    InSt<-InSt+nudt+sigsdt*epsilon
  }

  ST <- exp(InSt)
  for (j in 1:M) {
    CT[j] <- max(0,cp*(ST[j]-K))
    sumCT1 <- sumCT1+CT[j]
    sumCT2 <- sumCT2+CT[j]*CT[j]
  }

  value <- sumCT1/M*exp(-r*Tm)
  SD <- sqrt((sumCT2-sumCT1*sumCT1/M)*exp(-r*2*Tm)/(M-1))
  SE <- SD/sqrt(M)
  endtime<-Sys.time()
  timecost<-endtime-begintime
  return(c(value,SD,SE,timecost))
}
MC_Time_Call <- system.time(Call <- OptionMC("call",100,1,100,0.2,0.06,0.03,300,1e6))
MC_Time_Call #This is the system time for call simulation routine
```

```
##    user  system elapsed
```

```
##   35.02    2.55   37.56
```

```r
MC_Time_Put <- system.time(Put <- OptionMC("put",100,1,100,0.2,0.06,0.03,300,1e6))
MC_Time_Put #This is the system time for put simulation routine
```

```
##    user  system elapsed
##   41.91    2.44   44.34
```

```r
result1 <- cbind(Call,Put)
row.names(result1) <- c("Option Value","Standard Deviation(SD)","Standard Error(SE)",
                        "Time elapse")
result1 #This is Option value, estimate SD, estimate SE, Time elapse
```

```
##                               Call          Put
## Option Value            9.14851746  6.276483992
## Standard Deviation(SD) 13.69956943  9.084304850
## Standard Error(SE)      0.01369957  0.009084305
## Time elapse            37.52611899 44.342413902
```

b.##________the antithetic variates method______##

```r
set.seed(1)
#the antithetic variates method
OptionMC_Antithetic <- function(iscall,K,Tm,S0,sig,r,div,N,M){
  begintime<-Sys.time()
  #precompute constants
  dt <- Tm/N
  nudt <- (r-div-0.5*sig^2)*dt
  sigsdt <- sig*sqrt(dt)
  InS <- log(S0)
  if(iscall=="call"){cp <- 1} ##distinguish call and put option
  if(iscall=="put"){cp <- (-1)}

  sumCT1 <- sumCT2 <- 0
  InSt1<-InSt2<-CT<-matrix(nrow = 1,ncol = M) #initialize the matrix

  for (j in 1:M) { #for each simulation
    InSt1[j] <- InSt2[j] <- InS
    CT[j] <- 0
  }

  #break the loop to avoid for loop
  #Using random variables(epsilon) to simulate M tirals of InSt value at final time T
  for (i in 1:N) {
    epsilon<-rnorm(M)
    InSt1<-InSt1+nudt+sigsdt*epsilon
    InSt2<-InSt2+nudt+sigsdt*(-epsilon)
  }

  ST1 <- exp(InSt1)
  ST2 <- exp(InSt2)
  for (j in 1:M) {
    CT[j] <- 0.5*(max(0,cp*(ST1[j]-K))+max(0,cp*(ST2[j]-K)))
    sumCT1 <- sumCT1+CT[j]
    sumCT2 <- sumCT2+CT[j]*CT[j]
  }
```

```r
  value <- sumCT1/M*exp(-r*Tm)
  SD <- sqrt((sumCT2-sumCT1*sumCT1/M)*exp(-r*2*Tm)/(M-1))
  SE <- SD/sqrt(M)
  endtime<-Sys.time()
  timecost<-endtime-begintime
  return(c(value,SD,SE,timecost))
}
Call_Antithetic <- OptionMC_Antithetic("call",100,1,100,0.2,0.06,0.03,300,1e6)
Put_Antithetic <- OptionMC_Antithetic("put",100,1,100,0.2,0.06,0.03,300,1e6)
result2 <- cbind(Call_Antithetic,Put_Antithetic)
row.names(result2) <- c("Option Value","Standard Deviation(SD)","Standard Error(SE)",
                        "Time elapse")
result2 #This is Antithetic Option value, estimate SD, estimate SE, Time elapse
```

```
##                        Call_Antithetic Put_Antithetic
## Option Value                9.13114316    6.265871176
## Standard Deviation(SD)      7.20132985    4.640494716
## Standard Error(SE)          0.00720133    0.004640495
## Time elapse                45.88050795   45.756406069
```

______the delta-based control variate with beta[1]=-1_____##

```r
#MC Valuation with a delta-based control variate with beta[1]=-1
set.seed(1)
OptionMC_deltacontrol <- function(iscall,K,Tm,S0,sig,r,div,N,M){
  #from Figure 4.9 in the textbook(CS)
  begintime<-Sys.time()
  #precompute constants
  dt <- Tm/N
  nudt <- (r-div-0.5*sig^2)*dt
  sigsdt <- sig*sqrt(dt)
  erddt<-exp((r-div)*dt)

  if(iscall=="call"){cp <- 1} ##distinguish call and put option
  if(iscall=="put"){cp <- (-1)}
  beta1<-(-1)

  sumCT1 <- sumCT2 <- 0
  St<-CT<-cv<-matrix(nrow = 1,ncol = M) #initialize the matrix

  for (j in 1:M) { #for each simulation
    St[j] <- S0
    CT[j] <- cv <- 0
  }


  for (i in 1:N) { #for each time step
    t <- (i-1)*dt
    d1<-(log(St/K)+(r-div+0.5*sig^2)*Tm)/(sig*sqrt(Tm))
    if(iscall=="call"){delta<-exp(-div*t)*pnorm(d1)} #equation 1.22(page 8) in the textbook(CS)
    if(iscall=="put"){delta<-exp(-div*t)*(pnorm(d1)-1)}
```

```
    epsilon<-rnorm(M)
    Stn<-St*exp(nudt+sigsdt*epsilon)
    cv<-cv+delta*(Stn-St*erddt)
    St<-Stn
  }

  for (j in 1:M) {
    CT[j] <- max(0,cp*(St[j]-K))+beta1*cv[j]
    sumCT1 <- sumCT1+CT[j]
    sumCT2 <- sumCT2+CT[j]*CT[j]
  }

  value <- sumCT1/M*exp(-r*Tm)
  SD <- sqrt((sumCT2-sumCT1*sumCT1/M)*exp(-r*2*Tm)/(M-1))
  SE <- SD/sqrt(M)
  endtime<-Sys.time()
  timecost<-endtime-begintime
  return(c(value,SD,SE,timecost))
}
Call_deltacontrol <- OptionMC_deltacontrol("call",100,1,100,0.2,0.06,0.03,300,1e6)
Put_deltacontrol <- OptionMC_deltacontrol("put",100,1,100,0.2,0.06,0.03,300,1e6)
result3 <- cbind(Call_deltacontrol,Put_deltacontrol)
row.names(result3) <- c("Option Value","Standard Deviation(SD)","Standard Error(SE)",
                        "Time elapse(need to time 60s)")
result3 #This is deltacontrol Option value, estimate SD, estimate SE, Time elapse
```

```
##                               Call_deltacontrol Put_deltacontrol
## Option Value                        9.135590083      6.267116654
## Standard Deviation(SD)              2.041743837      2.085641979
## Standard Error(SE)                  0.002041744      0.002085642
## Time elapse(need to time 60s)       2.034595684      1.928345001
```

______the combined antithetic variates with delta-based control variate method_____##

```
#Monte Carlo Valuation of European Option in a Black-Scholes World
#with Antithetic and Delta-based Control Variates
set.seed(1)
#Follow figure 4.11 in textbook(CS)
OptionMC_Antideltacontrol<-function(isCall,K,Tm,S0,sigma,r,div,N,M){
  cp <-  ifelse(isCall, 1, -1)

  #initialization ----
  begintime <- Sys.time()
  dt=Tm/N
  nudt=(r-div-0.5*sigma^2)*dt
  sigsdt=sigma*sqrt(dt)
  erddt=exp((r-div)*dt)
  beta1=-1

  #delta
  BSdelta <- function(isCall=T,S0=rep(100,M),K,t=0,Tm,sigma,r,div){
```

```
    Tm=Tm-t
    d1 <- (1/(sigma*sqrt(Tm)))*(log(S0/K)+(r-div+0.5*sigma^2)*Tm)
    if(isCall){
      delta=exp(-div*Tm)*pnorm(d1)
    }
    else{
      delta=exp(-div*dt)*(pnorm(d1)-1)
    }
    return(delta)
  }

  sum_CT=0
  sum_CT2=0

  St1=rep(S0,M)
  cv1=rep(0,M)
  St2=rep(S0,M)
  cv2=rep(0,M)

  #price paths ----
  delta1<-delta2<-NULL
  Stn1<-Stn2<-NULL
  for(i in 1:N){
    epsilon=rnorm(M)
    t=(i-1)*dt
    delta1=BSdelta(isCall=isCall,S0=St1,K=K,t=t,Tm=Tm,sigma=sigma,r=r,div=div)
    delta2=BSdelta(isCall=isCall,S0=St2,K=K,t=t,Tm=Tm,sigma=sigma,r=r,div=div)
    Stn1=St1*exp(nudt+sigsdt*epsilon)
    cv1=cv1+delta1*(Stn1-St1*erddt)
    St1=Stn1
    Stn2=St2*exp(nudt-sigsdt*epsilon)
    cv2=cv2+delta2*(Stn2-St2*erddt)
    St2=Stn2
  }

  CT=0.5*(pmax(0,cp*(St1-K))+beta1*cv1+pmax(0,cp*(St2-K))+beta1*cv2)
  sum_CT=sum(CT)
  sum_CT2=sum(CT^2)

  #results ----

  value=sum_CT/M*exp(-r*Tm)
  SD=sqrt((sum_CT2-sum_CT*sum_CT/M)*exp(-2*r*Tm)/(M-1))
  SE=SD/sqrt(M)
  endtime <- Sys.time()
  timecost=endtime-begintime
  return(c(value,SD,SE,timecost))
}
Call_AntiandDelta <- OptionMC_Antideltacontrol(isCall=T,100,1,100,0.2,0.06,0.03,300,1e6)
Put_AntiandDelta <- OptionMC_Antideltacontrol(isCall=F,100,1,100,0.2,0.06,0.03,300,1e6)
result4 <- cbind(Call_AntiandDelta,Put_AntiandDelta)
row.names(result4) <- c("Option Value","Standard Deviation(SD)","Standard Error(SE)",
                        "Time elapse(need to time 60s)")
```

```r
result4 #This is Antithetic and Delta-based Control Option value, estimate SD, estimate SE, Time elapse
```

```
##                                Call_AntiandDelta Put_AntiandDelta
## Option Value                        9.1351673153     6.2675591419
## Standard Deviation(SD)              0.4128965669     0.3775907334
## Standard Error(SE)                  0.0004128966     0.0003775907
## Time elapse(need to time 60s)       3.5218874176     3.5279910326
```

____________Compare the results____________

```r
options(scipen = 200,digits=6) #do not use Scientific notation
result3_new <- result3
result3_new[4,] <- result3[4,]*60
result4_new <- result4
result4_new[4,] <- result4[4,]*60
result_compare <- as.data.frame(cbind(result1,result2,result3_new,result4_new))
result_compare
```

```
##                              Call        Put Call_Antithetic
## Option Value            9.1485175  6.2764840      9.13114316
## Standard Deviation(SD) 13.6995694  9.0843048      7.20132985
## Standard Error(SE)      0.0136996  0.0090843      0.00720133
## Time elapse            37.5261190 44.3424139     45.88050795
##                        Put_Antithetic Call_deltacontrol Put_deltacontrol
## Option Value               6.26587118        9.13559008       6.26711665
## Standard Deviation(SD)     4.64049472        2.04174384       2.08564198
## Standard Error(SE)         0.00464049        0.00204174       0.00208564
## Time elapse               45.75640607      122.07574105     115.70070004
##                        Call_AntiandDelta Put_AntiandDelta
## Option Value                 9.135167315      6.267559142
## Standard Deviation(SD)       0.412896567      0.377590733
## Standard Error(SE)           0.000412897      0.000377591
## Time elapse                211.313245058    211.679461956
```

```r
##Discuss the methods implemented
#The option values calculated through the four method are almost the same(although the Antithetic and D
#For the standard deviation(SD), the SD calculated through Simple Monte Carlo is much higher #than othe
#Also, when we apply the Antithetic Variety scheme, compared to the simple MC scheme,
#the variety will reduce from 1 to 0.25.
#For the time elapse, I try 1e6 trials for all methods, the Simple Monte Carlo Method costs
#the least time(less than 1 min), while the combination methods cost most time(more than 3 min)
```

Problem 2.Simulating the Heston Model

```r
#follow the paper part"4.Euler schemes for the CEV-SV model"
set.seed(1)
MonteCarlo_Heston<- function(K,T,S,r,div,N,M,V0,kappa,theta,rho,omega,alpha,f1,f2,f3)
{
  #In the paper, omega is the sigma of the volatility, know as volatility of volatility
  begintime<-Sys.time()
  dt<-T/N
  Vt<-V<-V_h<-St<-CT<-matrix(nrow =1, ncol = M)
  for(i in 1:M){
```

```
    Vt[i]<-V[i]<-V_h[i]<-V0
    St[i]<-S
  }

  V_origin<-V
  V_h_new<-Vt
  sum_CT<-sum_CT2<-0

  #simulating the path
  for(i in 1:N){
    epsilon1<-rnorm(M)
    epsilon2<-rnorm(M)

    #Integral equation (22) in paper
    St<-St+r*St*dt+sqrt(V)*St*(rho*epsilon1+sqrt(1-rho^2)*epsilon2)*sqrt(dt)
    #Apply equation(21) in paper
    V_h_new<-f1(V_h)-kappa*dt*(f2(V_h)-theta)+omega*(f3(V_h)^alpha)*epsilon1*sqrt(dt)
    V_origin<-f3(V_h_new)
    V_h<-V_h_new
    V<-V_origin
  }

  #Calculate option price same as the simple monte carlo function:
  for(j in 1:M){
    CT[j]<-max(St[j]-K,0)
    sum_CT<-sum_CT+CT[j]
    sum_CT2<-sum_CT2+CT[j]*CT[j]
  }

  Value<-exp(-r*T)*sum_CT/M*exp(-r*T)
  #The exact call option price(benchmark) is in this case C0=6.8061
  bias <- Value-6.8061
  SD <- sqrt((sum_CT2-sum_CT*sum_CT/M)*exp(-2*r*T)/(M-1))
  #the root mean square error(RMSE)
  RMSE <- SE <- SD/sqrt(M)

  endtime<-Sys.time()
  timecost<-endtime-begintime
  return(c(Value,bias,RMSE,timecost))
}

#Apply the Euler discretization schemes
#According to Table 1
#define three fixing functions for the five schemes:

#x+=max(x,0)
MaxX<-function(x){
  return(ifelse(x<0,0,x))
}
#the abosolute value function |x|
absolute<-function(x){
  return(abs(x))
}
```

```r
#identity fucntion x
ori<-function(x){
  return(x)
}
#Scheme
#Absorption
A<-MonteCarlo_Heston(K=100,T=1,S=100,r=0.0319,div,N=300,M=100000,V0=0.010201,kappa=6.21
              ,theta=0.019,rho=-0.7,omega=0.61,alpha=0.5,f1=MaxX,f2=MaxX,f3=MaxX)
#Reflection
R<-MonteCarlo_Heston(K=100,T=1,S=100,r=0.0319,div,N=300,M=100000,V0=0.010201,kappa=6.21
             ,theta=0.019,rho=-0.7,omega=0.61,alpha=0.5,f1=absolute,f2=absolute,f3=absolute)
#Higham and Mao
HM<-MonteCarlo_Heston(K=100,T=1,S=100,r=0.0319,div,N=300,M=100000,V0=0.010201,kappa=6.21
              ,theta=0.019,rho=-0.7,omega=0.61,alpha=0.5,f1=ori,f2=ori,f3=absolute)
#Partial truncation
PT<-MonteCarlo_Heston(K=100,T=1,S=100,r=0.0319,div,N=300,M=100000,V0=0.010201,kappa=6.21
              ,theta=0.019,rho=-0.7,omega=0.61,alpha=0.5,f1=ori,f2=ori,f3=MaxX)
#Full truncation
FT<-MonteCarlo_Heston(K=100,T=1,S=100,r=0.0319,div,N=300,M=100000,V0=0.010201,kappa=6.21
              ,theta=0.019,rho=-0.7,omega=0.61,alpha=0.5,f1=ori,f2=MaxX,f3=MaxX)

#Output Table listing the estimated call option price, the bias, the root mean square #error(RMSE),and
Result21 <- cbind(A,R,HM,PT,FT)
colnames(Result21)<-c('Absorption','Reflection','Higham and Mao',
                      'Partial truncation','#Full truncation')
rownames(Result21)<-c('OptionValue','Bias','RMSE','Calculate_Time')
Result21
```

```
##                  Absorption Reflection Higham and Mao Partial truncation
## OptionValue       6.6623154  6.7460095      6.6253103          6.6206297
## Bias             -0.1437846 -0.0600905     -0.1807897         -0.1854703
## RMSE              0.0239891  0.0245457      0.0237072          0.0234926
## Calculate_Time   23.1264000  9.9350259      9.7609279         17.6034720
##                  #Full truncation
## OptionValue             6.5624881
## Bias                   -0.2436119
## RMSE                    0.0233987
## Calculate_Time         21.4521840
```

```r
#The option value calculated by the Reflection method is mostly close to the exact call
#option price C0=6.8061. With a bias of -0.0600905, and the Reflection method cost the
#least time.
#Interesting thing is the other 4 methods performance are almost identical.
```

Problem 3.Multiple Monte Carlo Processes Question1

```r
##question 1-----------------------------------
##Given the information
P <- 10*10^6 #Total Value of portfolio at initial
IBM <- P*0.4 #Total Value of IBM Share
#Denotation
#X is the evolution of the IBM share price
#Y is unit of TBill
#Z is the number of Yuan obtained for $1
X0 <- 80 #IBM share price
```

```r
Yuan<-P*0.3 #Total Value of Yuan
Z0<-6.1 #Yuan Price

#Calculate the number of shares and the amount in Yuan that the portfolio
#contains in initial
NumberOfShares<-IBM/X0
AmountOfYuan<-Yuan*Z0 #in chinese rmb

result31 <- cbind(NumberOfShares,AmountOfYuan)
result31
```

```
##      NumberOfShares AmountOfYuan
## [1,]          50000     18300000
```

Questin2&3

```r
##question2&3-----------------------------------
set.seed(1)
#Assuming the Brownian motions are independent
VAR_MC<-function(M,N,div,r,sigma,x0,y0,z0,type,confidencelevel=0.99)
{
  dt<-0.001
  t<-10/252 #for all assets for 10days
  nudt<-(r-div-0.5*sigma^2)*dt
  sigsdt<-sigma*sqrt(dt)
  lnx<-log(x0)

  X<-rep(lnx,M)
  Y<-rep(y0,M)
  Z<-rep(z0,M)


  for(i in 1:round(t/dt))
  {
    epsilon1<-rnorm(M) #Brownian motions are independent
    epsilon2<-rnorm(M)
    epsilon3<-rnorm(M)
    #integral both sides of the three stochastic processes X,Y,Z we have:
    X<-X+nudt+sigsdt*epsilon1
    Y<-Y+100*(90000+1000*t-Y)*dt+sqrt(Y)*epsilon2*sqrt(dt)
    Z<-Z+5*(6-Z)*dt+0.01*sqrt(Z)*epsilon3*sqrt(dt)
  }
  Xt<-exp(X) #Final IBM share price at time T
  Yt<-Y #Final TBill price at time T
  Zt<-Z #Final Yuan price exchange for $1 at time T
  #calculate the number of Treasury bill at initial
  NumberOfTBill<-round(P*0.3/90000)

  #calculate the Total Value at final time T:
  Pt <-NumberOfShares*Xt+NumberOfTBill*Yt+AmountOfYuan/Zt
  R_sorted <- sort(((Pt-P)/P),decreasing = FALSE)

  #confidence level
  VAR<-R_sorted[(1-confidencelevel)*length(R_sorted)]
  CVAR<-sum(R_sorted[1:(M*0.01)])/((1-confidencelevel)*length(R_sorted))
```

```r
  if(type=='var'){return(VAR)}
  else{return(CVAR)}
}
#Calculate VAR &CVAR for the portfolio (a = 0.01, N = 10 days)

VAR_99<-VAR_MC(M=3*1e6,N=300,div=0,r=0.01,sigma=0.3,x0=80,
                    y0=90000,z0=6.1,type='var')
VAR_95<-VAR_MC(M=3*1e6,N=300,div=0,r=0.01,sigma=0.3,x0=80,
                    y0=90000,z0=6.1,type='var',confidencelevel = 0.95)
CVAR_99<-VAR_MC(M=3*1e6,N=300,div=0,r=0.01,sigma=0.3,x0=80,
                           y0=90000,z0=6.1,type='cvar')
CVAR_95<-VAR_MC(M=3*1e6,N=300,div=0,r=0.01,sigma=0.3,x0=80,
                           y0=90000,z0=6.1,type='cvar',confidencelevel = 0.95)
Result_risk<-matrix(nrow = 2,ncol = 2)
colnames(Result_risk)<-c('confidence level=95%','confidence level=99%')
rownames(Result_risk)<-c('VaR','CVaR')
Result_risk[1,1]<-VAR_95
Result_risk[1,2]<-VAR_99
Result_risk[2,1]<-CVAR_95
Result_risk[2,2]<-CVAR_99
#VAR&CVAR for the portfolio(in return)
Result_risk
```

```
##      confidence level=95% confidence level=99%
## VaR            -0.0400968           -0.0545303
## CVaR           -0.0123108           -0.0615584
```

```r
#VAR&CVAR for the portfolio(in dollar)
Result_risk_dollar <- Result_risk*P
Result_risk_dollar
```

```
##      confidence level=95% confidence level=99%
## VaR               -400968              -545303
## CVaR              -123108              -615584
```