Homework5

Problem 1 Comparing different Monte Carlo schemes

(a)

```
MC.valuation <- function(type, S, K, T, r, sig,div){

  timestart<-Sys.time()
  #precompute constants
  N = 300
  M = 1e+06
  dt = T/N
  nudt = (r-div-0.5*sig^2)*dt
  sigsdt = sig*sqrt(dt)
  lnS = log(S)

  sum_CT = 0
  sum_CT2 = 0

  for(j in 1:M){ #for each simulation
    lnSt = lnS
    for(i in 1:N){ #for each time step
      ybi = rnorm(1)
      lnSt = lnSt+nudt+sigsdt*ybi
    }

    ST = exp(lnSt)
    if(type=="call")
      CT = max(0, ST - K)
    if(type=="put")
      CT = max(0, K - ST)
    sum_CT = sum_CT + CT
    sum_CT2 = sum_CT2 + CT^2
  }
  value = sum_CT/M*exp(-r*T) #discount option value to time 0
  SD = sqrt((sum_CT2 - sum_CT^2/M)*exp(-2*r*T)/(M-1))
  SE = SD/sqrt(M) #standard error
  cat("option price",value,"\n","standard error",SE,"\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
}
MC.valuation(type="call", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
MC.valuation(type="put", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
```

	Price	SE	Time
Call option	9.1286	0.01379	684.42 sec
Put option	6.2593	0.00916	645.83 sec

(b)

```
#with Antithetic variance reduction
MC.avr <- function(type, S, K, T, r, sig,div){

  timestart<-Sys.time()
  #precompute constants
  N = 300
  M = 1e+06
  dt = T/N
  nudt = (r-div-0.5*sig^2)*dt
  sigsdt = sig*sqrt(dt)
  lnS = log(S)

  sum_CT = 0
  sum_CT2 = 0

  for(j in 1:M){ #for each simulation
    lnSt1 = lnS
    lnSt2 = lnS
    for(i in 1:N){ #for each time step
      ybi = rnorm(1)
      lnSt1 = lnSt1+nudt+sigsdt*ybi
      lnSt2 = lnSt2+nudt+sigsdt*(-ybi)
    }

  St1 = exp(lnSt1)
  St2 = exp(lnSt2)
  if(type=="call")
    CT = 0.5*(max(0, St1 - K) + max(0, St2 - K))
  if(type=="put")
    CT = 0.5*(max(0, K - St1) + max(0, K - St2))

  sum_CT = sum_CT + CT
  sum_CT2 = sum_CT2 + CT^2
  }
  value = sum_CT/M*exp(-r*T) #discount option value to time 0
  SD = sqrt((sum_CT2 - sum_CT^2/M)*exp(-2*r*T)/(M-1))
  SE = SD/sqrt(M) #standard error
  cat("option price",value,"\n","standard error",SE,"\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
```

```
}
MC.avr(type="call", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
MC.avr(type="put", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)

# Function to calculate opntion price by BS formular
BS <- function(type, S0, K, tau, r, sigma,div){

  d1 <- (log(S0/K)+(r-div+sigma^2/2)*tau) / (sigma*sqrt(tau))
  return(exp(-div*tau)*pnorm(d1))
}

#with delta-based control variate
MC.dbcv <- function(type, S, K, T, r, sig,div){

  timestart<-Sys.time()
  #precompute constants
  N = 300
  M = 1e+06
  dt = T/N
  nudt = (r-div-0.5*sig^2)*dt
  sigsdt = sig*sqrt(dt)
  erddt = exp((r-div)*dt)

  beta1 = -1

  sum_CT = 0
  sum_CT2 = 0

  for(j in 1:M){ #for each simulation
    St = S
    cv = 0
    for(i in 1:N){ #for each time step
      t = (i-1)*dt
      delta = BS(type, S, K, T, r, sig,div)
      ybi = rnorm(1)
      Stn = St*exp(nudt + sigsdt*ybi)
      cv = cv + delta*(Stn - St*erddt)
      St = Stn
    }

    if(type=="call")
      CT = max(0, St - K) + beta1*cv
    if(type=="put")
      CT = max(0, K - St) + beta1*cv

    sum_CT = sum_CT + CT
    sum_CT2 = sum_CT2 + CT^2
  }
  value = sum_CT/M*exp(-r*T) #discount option value to time 0
```

```
  SD = sqrt((sum_CT2 - sum_CT^2/M)*exp(-2*r*T)/(M-1))
  SE = SD/sqrt(M) #standard error
  cat("option price",value,"\n","standard error",SE,"\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
}
MC.dbcv(type="call", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
MC.dbcv(type="put", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)

#with Antithetic and delta-based control variate
MC.avr.dbcv <- function(type, S, K, T, r, sig,div){

  timestart<-Sys.time()
  #precompute constants
  N = 300
  M = 1e+06
  dt = T/N
  nudt = (r-div-0.5*sig^2)*dt
  sigsdt = sig*sqrt(dt)
  erddt = exp((r-div)*dt)
  beta1 = -1

  sum_CT = 0
  sum_CT2 = 0

  for(j in 1:M){ #for each simulation
    St1 = S
    St2 = S
    cv1 = 0
    cv2 = 0
    for(i in 1:N){ #for each time step
      t = (i-1)*dt
      delta1 = BS(type, S, K, T, r, sig,div)
      delta2 = BS(type, S, K, T, r, sig,div)
      ybi = rnorm(1)
      Stn1 = St1*exp(nudt + sigsdt*ybi)
      Stn2 = St2*exp(nudt + sigsdt*(-ybi))
      cv1 = cv1 + delta1*(Stn1 - St1*erddt)
      cv2 = cv2 + delta2*(Stn2 - St2*erddt)
      St1 = Stn1
      St2 = Stn2
    }

    if(type=="call")
      CT = 0.5*(max(0, St1 - K) + beta1*cv1 + max(0, St2 - K) + beta1*cv2)
    if(type=="put")
      CT = 0.5*(max(0, K - St1) + beta1*cv1 + max(0, K - St2) + beta1*cv2)
```

```
    sum_CT = sum_CT + CT
    sum_CT2 = sum_CT2 + CT^2
  }
  value = sum_CT/M*exp(-r*T) #discount option value to time 0
  SD = sqrt((sum_CT2 - sum_CT^2/M)*exp(-2*r*T)/(M-1))
  SE = SD/sqrt(M) #standard error
  cat("option price",value,"\n","standard error",SE,"\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
}
MC.avr.dbcv(type="call", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
MC.avr.dbcv(type="put", S=100, K=100, T=1, r=0.06, sig=0.20,div=0.03)
```

	MC	Antithetic	Delta-based	Anti&Delta
Call option	9.1286	9.1401	9.1358	9.1352
Call SD	13.682	7.236	0.5738	0.4255
Call SE	0.01379	0.00732	0.000562	0.000418
Call Time	684.42	902.70	3050.1	5788.13
Put option	6.2763	6.2701	6.2675	6.2668
Put SD	9.0853	4.6410	0.4762	0.3897
Put SE	0.00916	0.00459	0.000498	0.000391
Put Time	645.83	900.63	3094.57	5890.30

Comment：we can see from the table that Delta-based and Antithetic and delta-based monte carlo take longer than the other two, but they get more specific option price than the other two. Obviously, they have smaller standard deviation and standard error than nomal Monte carlo and Antithetic MC.

Problem 2 Simulating the Heston model

(a)

```
heston.price <- function(scheme,params,S,r,q,T,K,C0){

  timestart<-Sys.time()
  #input all parameters
  kappa = params[1];
  theta = params[2];
  sig = params[3];
  V0    = params[4];
  rho   = params[5];
  lambda = params[6];

  #precompute constants
  N = 100
  M = 1e+04
```

```
dt = T/N
sigsdt = sig*sqrt(dt)
lnS = log(S)

sum_CT = 0
sum_CT2 = c()

for(j in 1:M){ #for each simulation
  lnSt = lnS
  v=V0
  for(i in 1:N){ #for each time step
    ybi = rho*rnorm(1)+sqrt(1-rho^2)*rnorm(1)
    #Absorption
    if(scheme=="Absorption"){
      v = max(0,v)+kappa*(theta-max(0,v))*dt+sigsdt*sqrt(max(0,v))*rnorm(1)
      v = max(0,v)
      lnSt = lnSt+(r-q-0.5*lambda^2*v)*dt+lambda*sqrt(v*dt)*ybi
    }
    #Reflection
    else if(scheme=="Reflection"){
      v = abs(v)+kappa*(theta-abs(v))*dt+sigsdt*sqrt(abs(v))*rnorm(1)
      v = abs(v)
      lnSt = lnSt+(r-q-0.5*lambda^2*v)*dt+lambda*sqrt(v*dt)*ybi
    }
    #Higham and Mao
    else if(scheme=="Higham and Mao"){
      v = v+kappa*(theta-v)*dt+sigsdt*sqrt(abs(v))*rnorm(1)
      v = abs(v)
      lnSt = lnSt+(r-q-0.5*lambda^2*v)*dt+lambda*sqrt(v*dt)*ybi
    }
    #Partial truncation
    else if(scheme=="Partial truncation"){
      v = v+kappa*(theta-v)*dt+sigsdt*sqrt(max(0,v))*rnorm(1)
      v = max(0,v)
      lnSt = lnSt+(r-q-0.5*lambda^2*v)*dt+lambda*sqrt(v*dt)*ybi
    }
    #Full truncation
    else if(scheme=="Full truncation"){
      v = v+kappa*(theta-max(0,v))*dt+sigsdt*sqrt(max(0,v))*rnorm(1)
      v = max(0,v)
      lnSt = lnSt+(r-q-0.5*lambda^2*v)*dt+lambda*sqrt(v*dt)*ybi
    }
  }
  ST = exp(lnSt)
  CT = max(0, ST - K)
  sum_CT = sum_CT + CT
  sum_CT2 = c(sum_CT2 ,CT)
}
value = sum_CT/M*exp(-r*T) #discount option value to time 0
bias = value - C0
```

```
  sum_CT2 = sum_CT2*exp(-r*T)
  RMSE = sqrt(bias^2+var(sum_CT2))
  cat("option price",value,"\n","bias",bias,"\n","root standard error",RMSE,"
\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
}
heston.price(scheme = "Absorption",params = c(6.21,0.019,0.61,0.010201,-0.7,
1), S=100,r=0.0319,q=0,T=1,K=100,C0=6.8061)

## option price 6.878251
##  bias 0.0721514
##  root standard error 9.856377
## Time difference of 1.211295 mins

heston.price(scheme = "Reflection",params = c(6.21,0.019,0.61,0.010201,-0.7,
1), S=100,r=0.0319,q=0,T=1,K=100,C0=6.8061)

## option price 7.114763
##  bias 0.3086635
##  root standard error 10.20724
## Time difference of 1.025203 mins

heston.price(scheme = "Higham and Mao",params = c(6.21,0.019,0.61,0.010201,-0.
7,1), S=100,r=0.0319,q=0,T=1,K=100,C0=6.8061)

## option price 6.95831
##  bias 0.1522104
##  root standard error 10.12613
## Time difference of 1.154946 mins

heston.price(scheme = "Partial truncation",params =c(6.21,0.019,0.61,0.010201,
-0.7,1), S=100,r=0.0319,q=0,T=1,K=100,C0=6.8061)

## option price 6.989695
##  bias 0.1835947
##  root standard error 9.824872
## Time difference of 1.270409 mins

heston.price(scheme = "Full truncation",params = c(6.21,0.019,0.61,0.010201,-
0.7,1), S=100,r=0.0319,q=0,T=1,K=100,C0=6.8061)

## option price 6.761885
##  bias -0.04421503
##  root standard error 9.671764
## Time difference of 1.312537 mins
```

	Absorption	Reflection	Higham and Mao	Partial truncation	Full truncation
Option price	6.878251	7.114763	6.95831	6.989695	6.761885
bias	0.3086635	0.308663	0.1522104	0.1835947	-0.04421503
RMSE	10.20724	10.20724	10.12613	9.824872	9.671764
Time	1.211295 mins	1.025203 mins	1.154946 mins	1.270409 mins	1.312537 mins

Comment: all the method get option price close to the benchmark and they get kind of similar results in bias, RMSE and time. However, full truncation get option price most close to the bencnmark and has bias less than other function, take longer time though.

(b) bonus

```
Heston_Price <- function(params,S0,r,q,T,k,C0){

  timestart<-Sys.time()
  #input all parameters
  kappa = params[1];
  theta = params[2];
  sigma = params[3];
  V0    = params[4];
  rho   = params[5];
  lambda = params[6];

  i <- sqrt(-1+0i)
  kappa <- c(4,2,1)
  u1 <- 0.5
  u2 <- -0.5
  a <- kappa*theta
  b1 <- kappa+lambda-rho*sigma
  b2 <- kappa+lambda
  N = 1e+04
  up = 700

  psi <- function(Phi,b,u){
    d <- sqrt((rho*sigma*Phi*i-b)^2 - sigma^2*(2*u*Phi*i-Phi^2))
    g <- (b-rho*sigma*Phi*i+d)/(b-rho*sigma*Phi*i-d)
    C <- (r-q)*Phi*i*T+kappa*theta*((b-rho*sigma*Phi*i+d)*T -
                                      2*log((1-g*exp(d*T))/(1-g)))/sigma^2
    D <- (b-rho*sigma*Phi*i+d)/sigma^2 * ((1-exp(d*T))/(1-g*exp(d*T)))
    y <- exp( C+D*V0+i*Phi*log(S0))
    return(y)
  }

  expre <- function(Phi,b,u,k){
    y <- Re(exp(-i*Phi*log(k))*psi(Phi,b,u)/(i*Phi))
    return(y)
  }
```

```
  #use simpon method to calculate integral
  Simpon <- function(up,N,b,u,k){
    In <- 0
    h <- up/N
    for(j in 1:N+1){
      Phi <- j*h
      Phi1 <- (j-1)*h
      area <- (Phi - Phi1)/6 * (expre(Phi1,b,u,k) +
                                  4*expre((Phi + Phi1)/2,b,u,k)+expre(Phi,b,u,
k))
      In <- area + In
    }
    return(In)
  }

  #calculate option price by Hedson model
  simpon1 <- mean(Simpon(up,N,b1,u1,k))
  simpon2 <- mean(Simpon(up,N,b2,u2,k))
  P1 <- 1/2 + 1/pi * simpon1
  P2 <- 1/2 + 1/pi * simpon2
  price <- S0*P1 - k*exp(-(r-q)*T)*P2

  bias = abs(price - C0)
  #RMSE = sqrt(bias^2+var(price)) there is no RMSE,because we only have one v
alue
  cat("option price",price,"\n","bias",bias,"\n")
  timeend<-Sys.time()
  runningtime<-timeend-timestart
  print(runningtime)
}
Heston_Price(params = c(6.21,0.019,0.61,0.010201,-0.7,1),S0=100,r=0.0319,q=0,
T=1,k=100,C0=6.8061)
```

```
## option price 5.607216
```

```
## bias 1.108884
```

```
## Time difference of 8.231302 secs
```

Comment: the option price that I get is kind of less than the option price in (a), but it get results faster than method in (a).

(c) bonus

```
i <- sqrt(-1+0i)
S0 <- 100
K <- 100
r <- 0.00319
T <- 1
sigma <- 0.61
```

```r
k <- log(K)

alpha <- 5
lambda <- 0.1
N <- 2^12
eta <- 2*pi/N/lambda

chara_fun <- function(u){
  exp(i*(log(S0)+(r-sigma^2/2)*T)*u - 0.5*sigma^2*T*u^2)
}

psi_fun <-function(nu){
  term1 = exp(-r*T)*chara_fun(nu-(alpha+1)*i)
  term2 = alpha^2+alpha-nu^2+i*(2*alpha+1)*nu
  term1/term2
}

Kronecker.delta <- function(x){
  if(x==0)
    return(1)
  else
    return(0)
}

call_price <- function(u){
  #substitute some parameters
  a = N*eta
  b = 1/2*N*lambda #strike level from -b to b
  ku = -b+lambda*(u-1)

  term = c()
  #Using the trapezoid rule for the integral
  for(j in 1:N){
    nu.j = eta*(j-1)
    term1 = exp(-i*2*pi/N*(j-1)*(u-1)) * exp(i*b*nu.j)
    term2 = psi_fun(nu.j) * eta/3 * (3+(-1)^j-Kronecker.delta(j-1))
    term = c(term,term1*term2)
  }
  return(Re(exp(-alpha*ku)/pi * sum(term)))
}
b = 1/2*N*lambda
u = round((k+b)/lambda+1)
cat("option price by FFT: ",call_price(u),"\n")

## option price by FFT:  24.28207
```

Problem 3 Generating correlated BM and pricing basket options

(a)

```
mat.corr <- matrix(data=c(1.0,0.5,0.2,0.5,1.0,-0.4,0.2,-0.4,1.0),nrow=3,ncol=
3)
L = t(chol(mat.corr))
L #lower triangular matrix
```

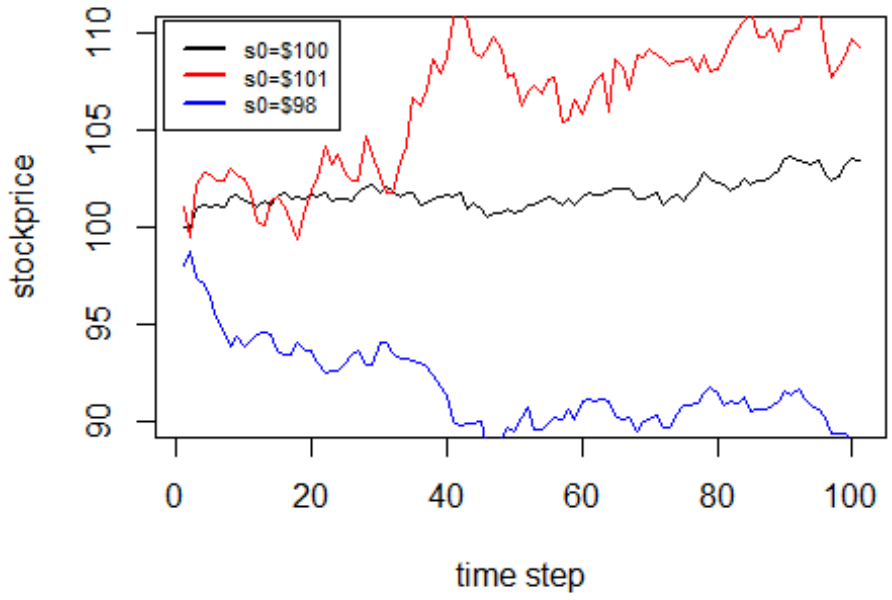
```
##      [,1]       [,2]      [,3]
## [1,]  1.0  0.0000000 0.0000000
## [2,]  0.5  0.8660254 0.0000000
## [3,]  0.2 -0.5773503 0.7916228
```

(b)

```
#input parameters
s0 <- c(100,101,98)
mu <- c(0.03,0.06,0.02)
sig <- c(0.05,0.2,0.15)
nu <- mu-0.5*sig^2
T <- 100/365
m <- 1000
dt <- 1/365
n <- T/dt

set.seed(1)
ST1=ST2=ST3=c()
for(i in 1:m){
  x <- matrix(rnorm(3*n),nrow=n,ncol=3)
  ep <- x%*%L
  e1 <- append(0,ep[,1])
  e2 <- append(0,ep[,2])
  e3 <- append(0,ep[,3])
  s1=s2=s3=c()
  S1=s0[1] #initial value for asset 1
  S2=s0[2] #initial value for asset 2
  S3=s0[3] #initial value for asset 3
  for(j in 1:(n+1)){
    S1 <- S1*exp(nu[1]*dt+e1[j]*sig[1]*sqrt(dt))
    S2 <- S2*exp(nu[2]*dt+e2[j]*sig[2]*sqrt(dt))
    S3 <- S3*exp(nu[3]*dt+e3[j]*sig[3]*sqrt(dt))
    s1 <- c(s1,S1)
    s2 <- c(s2,S2)
    s3 <- c(s3,S3)
  }
  ST1 <- c(ST1,s1[101])
  ST2 <- c(ST2,s2[101])
  ST3 <- c(ST3,s3[101])
}
plot(1:(n+1),s1,ylim=c(90,110),xlab="time step",ylab = "stockprice",,type="l")
```

```
lines(1:(n+1),s2,col="red")
lines(1:(n+1),s3,col="blue")
legend("topleft",c("s0=$100","s0=$101","s0=$98"),inset = .01,col=c("black","r
ed","blue"),lwd=2,cex=0.7)
```




```
stock1 <- mean(ST1) #stock price of asset1
stock2 <- mean(ST2) #stock price of asset2
stock3 <- mean(ST3) #stock price of asset3
data.frame(stock1,stock2,stock3)

##    stock1   stock2   stock3
## 1 100.841 102.9889 98.28467
```

(c)

```
basket.price <- function(type,K,r,T,m){

  sum_CT = 0
  for(i in 1:m){
    x <- matrix(rnorm(300),nrow=100,ncol=3)
    ep <- x%*%t(L)
    e1 <- append(0,ep[,1])
    e2 <- append(0,ep[,2])
    e3 <- append(0,ep[,3])
    s1=s2=s3=c()
    S1=s0[1] #initial value for asset 1
    S2=s0[2] #initial value for asset 2
    S3=s0[3] #initial value for asset 3
```

```
    for(j in 1:(n+1)){
      S1 <- S1*exp(nu[1]*dt+e1[j]*sig[1]*sqrt(dt))
      S2 <- S2*exp(nu[2]*dt+e2[j]*sig[2]*sqrt(dt))
      S3 <- S3*exp(nu[3]*dt+e3[j]*sig[3]*sqrt(dt))
      s1 <- c(s1,S1)
      s2 <- c(s2,S2)
      s3 <- c(s3,S3)
    }
    a1=a2=a3=1/3
    U = a1*s1+a2*s2+a3*s3 #price of basket stocks

    if(type=="call"){
      alpha=1
      CT = max(0,alpha*(U[101]-K))
    }
    else if(type=="put"){
      alpha=-1
      CT = max(0,alpha*(U[101]-K))
    }
    sum_CT = sum_CT + CT
  }
  value = sum_CT/m*exp(-r*T)
  return(value)
}
cat("call option: ",basket.price(type = "call",K = 100,r=0.11/3,T=100/365,m=1
000))

## call option:  1.958669

cat("\n","put option: ",basket.price(type = "put",K = 100,r=0.11/3,T=100/365,
m=1000))

##
##  put option:  1.305579
```

(d)

```
basket.exotic.price <- function(B,s,mu,sig,K,r,T,m){

  sum_CT = 0
  for(i in 1:m){
    x <- matrix(rnorm(300),nrow=100,ncol=3)
    ep <- x%*%t(L)
    e1 <- append(0,ep[,1])
    e2 <- append(0,ep[,2])
    e3 <- append(0,ep[,3])
    s1=s2=s3=c()
    S1=s0[1] #initial value for asset 1
    S2=s0[2] #initial value for asset 2
    S3=s0[3] #initial value for asset 3
    for(j in 1:(n+1)){
```

```
      S1 <- S1*exp(nu[1]*dt+e1[j]*sig[1]*sqrt(dt))
      S2 <- S2*exp(nu[2]*dt+e2[j]*sig[2]*sqrt(dt))
      S3 <- S3*exp(nu[3]*dt+e3[j]*sig[3]*sqrt(dt))
      s1 <- c(s1,S1)
      s2 <- c(s2,S2)
      s3 <- c(s3,S3)
    }
    a1=a2=a3=1/3
    U = a1*s1+a2*s2+a3*s3 #price of basket stocks
    #print(U[100])
    A2 = mean(s2)
    A3 = mean(s3)
    CT = max(0,U[101]-K) #condition(iv)
    for(i in 1:(n+1)){
      if(s2[i] > B){  #condition(i)
        CT = max(0,s2[101]-K)
        break;
      }
    }
    if(which.max(s2) > which.max(s3)) #condition(ii)
      CT = max(0,s2[101] - K)^2
    if(A2 > A3) #condition(iii)
      CT = max(0,A2 - K)

    sum_CT = sum_CT + CT
  }
  value = sum_CT/m*exp(-r*T)
  return(value)
}
basket.exotic.price(B=104,s=s0,mu=mu,sig=sig,K=100,r=0,T=100/365,m=1000)

## [1] 5.192726
```