Barrier Option in Black-Scholes Model with Monte-Carlo

By Xinning Liu

1. Instruction

This report focusses on pricing a single barrier option - an "up and out" call option- under Black-Scholes Model. First, it will compute the theoretical price under the Black-Scholes Model assumption. Second, it will simulate the price using Monte-Carlo method.

2. Theoretical price

We assume that the price of underlying asset S_t satisfies the diffusion equation:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t$$

Where $W_t = W\ (t, \omega)$, $(t, \omega) \in \mathbb{R}_+ \times \Omega$ denote a standard Brownian motion under a given probability measure $\mathcal{P}$ on Ω.

According to "Closed Form Formulas for Exotic Options and their Lifetime Distribution" by Raphael DOUDAY, 2000, the theoretical price of an "up and out" call option at $t = 0$ of asset S_t with strike price K, single upper barrier H, risk-neutral drift μ, constant volatility σ, maturity T, and refinance interest rate r follows the equation below:

$$\begin{aligned} UOC(S_0, K, H) = {} & e^{(\mu - r)T} S_0 (N(d_1) - N(d_3) - \alpha' \big(N(d_6) - N(d_8)\big) \\ & - e^{-rT} K (N(d_2) - N(d_4) - \alpha \big(N(d_5) - N(d_7)\big) \end{aligned}$$

Where $N\ (\cdot)$ is the c.d.f. of standard normal distribution, and other variables as following:

$$h = \frac{1}{\sigma} \log \frac{H}{S} \qquad k = \frac{1}{\sigma} \log \frac{K}{S}$$

$$\lambda = \frac{\mu}{\sigma} - \frac{\sigma}{2} \qquad \lambda' = \frac{\mu}{\sigma} + \frac{\sigma}{2}$$

$$\alpha = e^{2\lambda h} \qquad \alpha' = e^{2\lambda' h} = \frac{\alpha H^2}{S^2}$$

$$d_1 = \lambda' \sqrt{T} - k/\sqrt{T} \qquad d_5 = -\lambda \sqrt{T} - h/\sqrt{T}$$

$$d_2 = \lambda \sqrt{T} - k/\sqrt{T} \qquad d_6 = -\lambda' \sqrt{T} - h/\sqrt{T}$$

$$d_3 = \lambda' \sqrt{T} - h/\sqrt{T} \qquad d_5 = -\lambda \sqrt{T} - (2h - k)/\sqrt{T}$$

$$d_4 = \lambda \sqrt{T} - h/\sqrt{T} \qquad d_8 = -\lambda' \sqrt{T} - (2h - k/\sqrt{T}$$

When strike price $K = 100$, single upper barrier $H = 120$, risk-neutral drift $\mu = 1\%$, constant volatility $\sigma = 20\%$, maturity $T = 1$, and refinance interest rate $r = 0$, the initial asset price $S_0 = 100$, the price of an "up and out" call option at $t = 0$ is

$$UOC(100{,}100{,}120) = 1.1355$$

The option simulated below will have the same constraints.

3. Monte Carlo method

We will use 1 basic Monte Carlo method and 2 variance reduction method to simulate the price of an "up and out" call option.

All simulations run in RStudio with R version 3.4.2 in a computer with processor 1.4 GHz Intel Core i5, memory 4GB.

a. Basic Monte Carlo method under Euler Scheme

Under Euler Scheme, we have

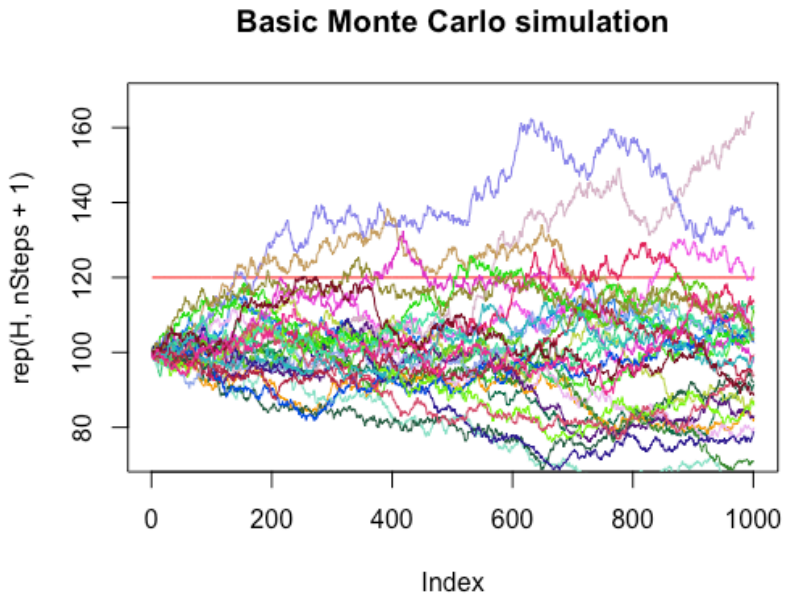
$$\Delta S_{t_n} = \mu S_{t_{n-1}} \Delta t + \sigma S_{t_{n-1}} \Delta W_{t_n}$$

Since W_t is a standard Brownian motion, $\Delta W_{t_n} = \sqrt{\Delta t}\, Z_{t_n}$, where Z_t is a standard normal variable, we have

$$\Delta S_{t_n} = \mu S_{t_{n-1}} \Delta t + \sigma S_{t_{n-1}} \sqrt{\Delta t}\, Z_{t_n}$$

$$S_{t_n} = S_{t_{n-1}} + \Delta S_{t_n} = S_{t_{n-1}} (1 + \mu \Delta t + \sigma \sqrt{\Delta t}\, Z_{t_n})$$

The price of asset S_t simulation with number of steps $n = 1000$, number of iterations $N = 30$ plots below:



Once the price hit the boundary H, the option deactivates with price 0.

Note: Since the discrete property of the simulation, we will lose the scenario that the price goes up beyond the barrier and goes back between the steps, the simulation price should be over-valued.

i) Result analysis

To get a statistical result, we run the simulation 20 times and compute the average of the price (MCmean), average difference between the MC price and the theoretical price (MCdiff), and the standard deviation of MC price (MCsd).

We can find that all simulated prices are greater than the theoretical price, and the difference shrinks by the number of steps n increasing. As the number of iterations N increasing, we can see the standard deviation decrease. When $n * N > 1e + 08$, the simulation time become unaffordable under the computer environment.

The $RMSE\ (relative\ mean\ square\ error) = \mathrm{Sd}\ (MCdiff) = \mathrm{Sd}(\mathrm{MC\ Price} - \mathrm{Theo\ Price})$, i.e. $RMSE = \mathrm{Sd}\ (MC\ Price)$. Notice when $n > 1e + 04$, $MCsd > MCdiff$, the MC Price may cover the theoretical price.

Table 1 average of the MC price

N \ n	10	20	40	80	100	1000	1e+04	1e+05
1000	1.8794	1.6973	1.4847	1.3748	1.4043	1.2395	1.1811	1.1427
10000	1.8787	1.6328	1.4923	1.3931	1.3691	1.1999	1.1661	
1e+05	1.8645	1.6493	1.4960	1.3895	1.3612	1.2076		
1e+06	1.8632	1.6504	1.4957	1.3896	1.3615			

Table 2 average difference between MC price and theoretical price

N \ n	10	20	40	80	100	1000	1e+04	1e+05
1000	0.7439	0.5618	0.3492	0.2393	0.2688	0.1040	0.0456	0.0072
10000	0.7432	0.4973	0.3568	0.2576	0.2336	0.0644	0.0306	
1e+05	0.7290	0.5138	0.3605	0.2540	0.2257	0.0721		
1e+06	0.7277	0.5149	0.3602	0.2541	0.2260			

Table 3 Standard deviation of MC price

N \ n	10	20	40	80	100	1000	1e+04	1e+05
1000	0.1378	0.1034	0.1345	0.0727	0.1406	0.0979	0.1044	0.1128
10000	0.0512	0.0434	0.0365	0.0384	0.0443	0.0277	0.0333	
1e+05	0.0128	0.0103	0.0122	0.0105	0.0115	0.0092		
1e+06	0.0043	0.0031	0.0045	0.0037	0.0037			

Table 4 Time consumption

N \ n	10	20	40	80	100	1000	1e+04	1e+05
1000	0.1	0.1	0.2	0.4	0.6	6.2	58.5	967.4
10000	0.7	1.2	2.2	4.1	5.2	55.7	985.6	
1e+05	7.4	12.1	22.5	51.8	53.9	1136.8		
1e+06	74.2	128.8	234.3	601.7	819.9			

b. Variance reduction method

Here we use two variance reduction techniques: i) rescale by value and ii) rescale by weight. Both under the Euler scheme.

i) Rescale by value

Here we rescale all random variates (Z) we used by $X = \frac{Z-\overline{Z}}{\mathrm{Sd}(Z)}$ to make them more standard normal i.e. $\bar{X} = 0$, $\mathrm{Sd}(X) = 1$, $\Delta W_{t_n} = \sqrt{\Delta \mathrm{t}}\, X_{t_n}$

ii) Rescale by weight

Here we use a Vanilla call option with same strike and maturity as control variate. We force the Vanilla call's price to match the Black-Scholes formula.

Unlike the method mentioned above, which use symmetric weight

$$w_i = \frac{1}{N}$$

for all iteration, this method wants to find an optimal weight so that the control variate call option can have the same value with the theoretical Black-Scholes price.

The weight follows the below equations, which is a quadratic programming problem:

$$\min \sum w_i^2$$

$$\sum w_i = 1$$

$$\sum w_i \, (S_T^i - K)_+ e^{-T\,r} = BS(K, 0)$$

$$w_i \geq 0$$

iii) Result analysis

Here number of steps $n = 10000$, number of iterations $N = 1000$. Also run the simulation 20 times.

We also run the simulation by 40 times and 100 times to see the difference.

Table 5 Comparison of three methods with theoretical price =1.1355, 20 times

	MCmean	MCdiff	MCsd	Time Consumption
Basic Monte Carlo	1.17230	0.03680012	0.1035550	40.623
Rescale by value	1.17113	0.03563050	0.1095377	45.536
Rescale by weight	1.16921	0.03371035	0.1029506	48.011

Table 6 Comparison of three methods with theoretical price =1.1355, 40 times

	MCmean	MCdiff	MCsd	Time Consumption
Basic Monte Carlo	1.126434	-0.009065432	0.09889007	63.958
Rescale by value	1.131434	-0.004066025	0.10321665	73.500
Rescale by weight	1.123233	-0.012266991	0.09841909	77.202

Table 7 Comparison of three methods with theoretical price =1.1355, 100 times

	MCmean	MCdiff	MCsd	Time Consumption
Basic Monte Carlo	1.143308	0.007808317	0.1102240	171.883
Rescale by value	1.141119	0.005619160	0.1122854	201.722
Rescale by weight	1.139473	0.003973219	0.1098732	206.524

We can see both variance reduction techniques reduce the difference between MC price and theoretical price, rescale by weight can also reduce the standard deviation of the MC price. We may notice that the MCsd's do not perform better when simulation runs increasing, which is because MCsd is relevant with the number of iterations N.

The rescale by weight method needs quadratic programming. When number of iterations $N \geq 1e + 04$, the time consumption will be unaffordable.

Here we run the simulation 20 times with $N = 5000, n = 10000$

Table 8 Comparison of three methods with theoretical price =1.1355, 20 times, N=5000

	MCmean	MCdiff	MCsd	Time Consumption
Basic Monte Carlo	1.157097	0.02159772	0.04814338	257.330
Rescale by value	1.159306	0.02380649	0.04711901	290.090
Rescale by weight	1.152320	0.01681988	0.04765833	971.149

We can see, with $N = 5000$, rescale by weight has a better MCdiff, also all three methods' standard deviation MCsd reduced to half.

4. Conclusion

A single barrier option has a theoretical pricing. A basic Monte Carlo method will be a bias estimator of the price. Variance reduction techniques can help Monte Carlo to perform better. An unbiased estimator should also consider Brownian bridge.

R code

Function to compute c.d.f. of standard normal distribution:

```
cdfnorm<-function(x) integrate(dnorm,-Inf,x)$value
```

Function to compute the theoretical price of an “up and out” call:

```
UOCTheo<-function(S0,K,H,Tm,mu,sigma,r){
 lamda=mu/sigma-sigma/2
 lamda_=mu/sigma+sigma/2
 h=1/sigma*log(H/S0)
 k=1/sigma*log(K/S0)
 alpha=exp(2*lamda*h)
 alpha_=exp(2*lamda_*h)

 d1= lamda_*sqrt(Tm)-k/sqrt(Tm)
 d2= lamda *sqrt(Tm)-k/sqrt(Tm)
 d3= lamda_*sqrt(Tm)-h/sqrt(Tm)
 d4= lamda *sqrt(Tm)-h/sqrt(Tm)
 d5=-lamda *sqrt(Tm)-h/sqrt(Tm)
 d6=-lamda_*sqrt(Tm)-h/sqrt(Tm)
 d7=-lamda *sqrt(Tm)-(2*h-k)/sqrt(Tm)
 d8=-lamda_*sqrt(Tm)-(2*h-k)/sqrt(Tm)
 UOC<-function(S,K,H) exp((mu-r)*Tm)*S*(cdfnorm(d1)-cdfnorm(d3)-
alpha_*(cdfnorm(d6)-cdfnorm(d8)))-exp(-r*Tm)*K*(cdfnorm(d2)-
cdfnorm(d4)-alpha*(cdfnorm(d5)-cdfnorm(d7)))
 UOC(S0,K,H)
}
```

Function to compute the Black-Scholes Vanilla call option price

```
cBS<-function(t,x) {
 dPlus<-function(t,x) 1/(sigma*sqrt(t))*(log(x/K)+(r+sigma^2/2)*t)
 dMinus<-function(t,x) 1/(sigma*sqrt(t))*(log(x/K)+(r-sigma^2/2)*t)
 if (t==Tm){
  0
 }else{x*cdfnorm(dPlus(Tm-t,x))-K*exp(-r*(Tm-t))*cdfnorm(dMinus(Tm-
t,x))}
}
```

Initialization:

```
K=100
S0=100
H=120
Tm=1
mu=1*0.01
sigma=20*0.01
r=0

pTheo<-UOCTheo(S0,K,H,Tm,mu,sigma,r)
```

```
cBSPrice0<-cBS(0,K)
```

Basic Monte Carlo method:

```
nB = 20
# nSteps in 10:100
# nTimes in 1000:1000000
timeTable<-matrix(rep(0,4*10),nrow=4)
MCmean<-matrix(rep(0,4*10),nrow=4)
MCdiff<-matrix(rep(0,4*10),nrow=4)
MCsd<-matrix(rep(0,4*10),nrow=4)

for(x in 1:4){
 nTimes=10^(x+2)
 for(y in 1:10){
  nSteps=10*y
  dt=Tm/nSteps
  dtSd=sqrt(dt)

  starttime=proc.time()
  MCPriceA<-c()

  for(t in 1:nB) {
   W=rnorm(nSteps*nTimes)
   # Basic Monte-Carlo: Euler scheme, no variance reduction method
   px<-c()
   X<-matrix(mu*dt+sigma*dtSd*W,nrow = nTimes)
   for (j in 1:nTimes){
    S=S0
    b=FALSE
    for (i in 1:nSteps){
     S=S*(1+X[j,i])
     if (S>H)
     {b=TRUE}
    }
    if (b) {p=0} else {p=max(S-K,0)*exp(-r*Tm)}
    px[j]<-p
   }
   MCPriceA[t]<-mean(px)
  }
  timeTable[x,1]<-(proc.time()-starttime)[3]
  MCmean[x,1]<-mean(MCPriceA)
  MCdiff[x,1]<-mean(MCPriceA)-pTheo
  MCsd[x,1]<-sd(MCPriceA)
  print(paste("nTimes=",nTimes,"nSteps=",nSteps))
 }
}
```

The Monte Carlo methods comparison:

```
max_K<-function(S) max(S-K,0)
library(quadprog)
```

```
nB = 20

nSteps=10000
nTimes=5000
dt=Tm/nSteps
dtSd=sqrt(dt)

MCPriceA<-rep(0,nB)
MCPriceB<-rep(0,nB)
MCPriceC<-rep(0,nB)
MCPriceD<-rep(0,nB)
timeTable<-matrix(rep(0,4*nB),nrow=nB)
ST<-rep(0,nTimes)
px<-rep(0,nTimes)
for(t in 1:nB) {
 W=rnorm(nSteps*nTimes)
 # a) basic Monte Carlo
 starttime=proc.time()

 X<-matrix(mu*dt+sigma*dtSd*W,nrow = nTimes)
 for(j in 1:nTimes){
  S=S0
  b=FALSE
  for (i in 1:nSteps){
   S=S*(1+X[j,i])
   if (S>H)
   {b=TRUE}
  }
  ST[j]<-S
  if (b) {p=0} else {p=max(S-K,0)*exp(-r*Tm)}
  px[j]<-p
 }
 MCPriceA[t]<-mean(px)
 timeTable[t,1]<- (proc.time()-starttime)[3]

 # c) rescale by weight
 starttime=proc.time()
 Dmat=diag(1,nTimes)
 dvec=rep(0,nTimes)
 A<-matrix(c(rep(1,nTimes),sapply(ST, max_K)*exp(-
r*Tm),diag(1,nTimes)),ncol=2+nTimes)
 bvec=c(1,cBSPrice0,rep(0,nTimes))
 w<-solve.QP(Dmat=Dmat, dvec=dvec, Amat=A, bvec=bvec, meq=2)$solution
 MCPriceC[t]<-px%*%w
 timeTable[t,3]<- (proc.time()-starttime)[3]+timeTable[t,1]

 # b) rescale by value
 starttime=proc.time()

 sdW=sd(W)
 meanW=mean(W)
 Z=(W-meanW)/sdW
 X<-matrix( mu*dt+sigma*dtSd*Z, nrow = nTimes )
 for (j in 1:nTimes){
```

```
  S=S0
  b=FALSE
  for (i in 1:nSteps){
   S=S*(1+X[j,i])
   if (S>H)
   {b=TRUE}
  }
  ST[j]<-S
  if (b) {p=0} else {p=max(S-K,0)*exp(-r*Tm)}
  px[j]<-p
 }
 MCPriceB[t]<-mean(px)
 timeTable[t,2]<- (proc.time()-starttime)[3]


 # c) rescale by value and weight
 starttime=proc.time()
 Dmat=diag(1,nTimes)
 dvec=rep(0,nTimes)
 A<-matrix(c(rep(1,nTimes),sapply(ST, max_K)*exp(-
r*Tm),diag(1,nTimes)),ncol=2+nTimes)
 bvec=c(1,cBSPrice0,rep(0,nTimes))
 w<-solve.QP(Dmat=Dmat, dvec=dvec, Amat=A, bvec=bvec, meq=2)$solution
 MCPriceD[t]<-px%*%w
 timeTable[t,4]<- (proc.time()-starttime)[3]+timeTable[t,2]

 print(timeTable[t,])
}
MC_average<-
c(mean(MCPriceA),mean(MCPriceB),mean(MCPriceC),mean(MCPriceD))
MC_Diff<-
c(mean(MCPriceA),mean(MCPriceB),mean(MCPriceC),mean(MCPriceD))-
pTheo
MC_Sd<-c(sd(MCPriceA),sd(MCPriceB),sd(MCPriceC),sd(MCPriceD))
print(c(sum(timeTable[,1]),sum(timeTable[,2]),sum(timeTable[,3]),sum(time
Table[,4])))
```