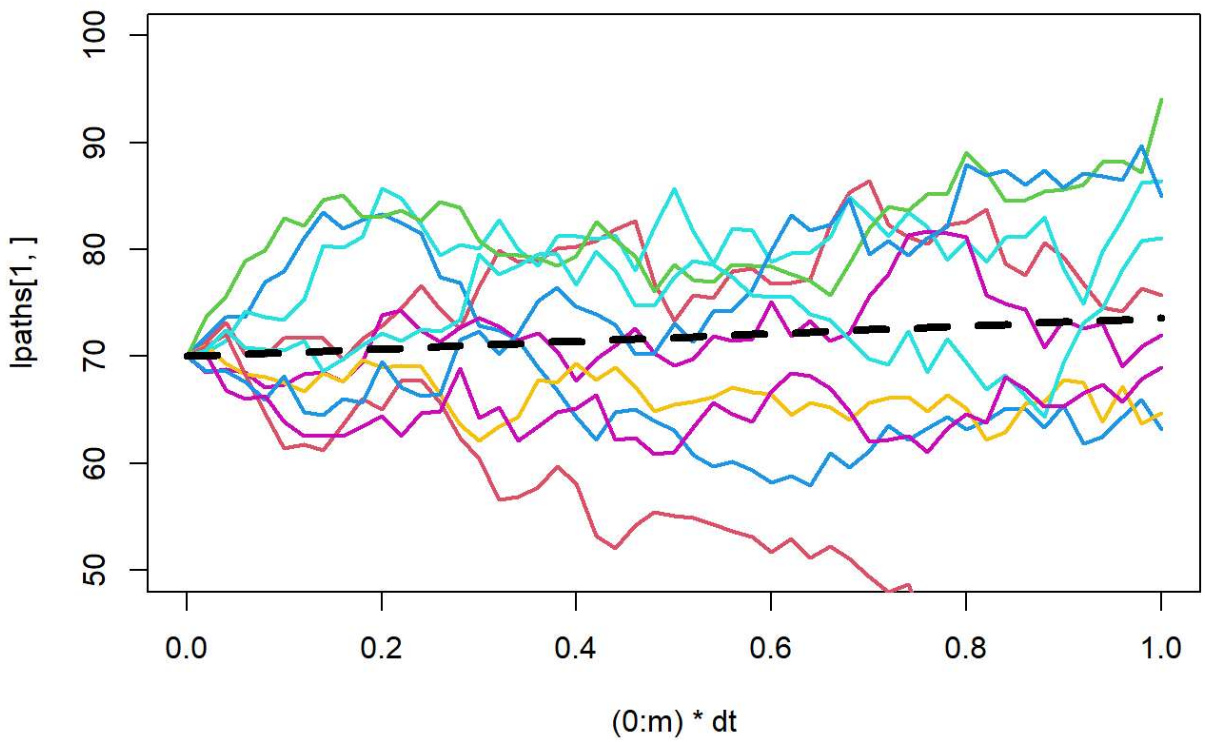
Simulation: Pricing Exotic Derivatives

This is sourced from STAD70 course practice questions and sample R code taught by professor Sotos. If you have any questions/concerns/comments feel free to email me: cristal.wang111@gmail.com (mailto:cristal.wang111@gmail.com).

1.Barrier Option - MC price

```
#####
# Barrier Option

n=10000 # number of paths
m=50 # number of steps within each path
r=.05 # risk-free rate
v=.2 # volatility
M=1 # expiration
S0=70 # starting asset price
K=80 # strike
B=90 # barrier

dt=M/m
mu=(r-v^2/2)*dt
sig=v*sqrt(dt)

Z=matrix(rnorm(m*n),n,m)

### Generate GBM discretized paths
lpaths=cbind( rep(S0,n) , t(S0*exp( apply( mu+sig*Z , 1, cumsum) )) )

### Plot Sample GBM Paths & Risk-Free Path
plot((0:m)*dt,lpaths[1,],type='l', lwd=2, col=2 , ylim=c(50,100)) # plot paths
for(i in 2:10){
  lines((0:m)*dt,lpaths[i,],type='l', lwd=2, col=i%%6+2)
}
lines((0:m)*dt,S0*exp((0:m)*dt*r), lwd=4, lty=2 )
```




```
### Barrier Option MC Price:

ST=lpaths[,m+1] # Prices at expiry
maxS=apply(lpaths,1,max) # Maximum of paths
minS=apply(lpaths,1,min)

Cuo_payoff=exp(-r*M)*pmax(0,ST-K)*(maxS < B)  # up-and-out call payoff
Cdo_payoff=exp(-r*M)*pmax(0,ST-K)*(minS > B)  # down-and-out call payoff
Cui_payoff=exp(-r*M)*pmax(0,ST-K)*(maxS > B)  # up-and-in call payoff
Cdi_payoff=exp(-r*M)*pmax(0,ST-K)*(minS < B)  # down-and-in call payoff

PUO_payoff=exp(-r*M)*pmax(0,K-ST)*(maxS < B) # Up-and-Out Put (PUO)
PDO_payoff=exp(-r*M)*pmax(0,K-ST)*(minS > B) # Down-and-Out Put (PDO)
PUI_payoff=exp(-r*M)*pmax(0,K-ST)*(maxS > B) # Up-and-In Put (PUI)
PDI_payoff=exp(-r*M)*pmax(0,K-ST)*(minS < B) # Down-and-In Put (PDI)

payoff = Cuo_payoff
(C_UandO=mean(payoff)) # MC price of up-and-out call
```

```
## [1] 0.3657452
```

2.Hitting probability of standard BM

Find the probability that standard BM $\{Wt\}$ hits barrier B=1 before time T=1

2.1.True Prob

```
#####
# Hitting probability of standard BM

B=1; T=1 # expiration

(p.true = 2*pnorm(-B,0,sd=sqrt(T))) # true hitting prob
```

```
## [1] 0.3173105
```

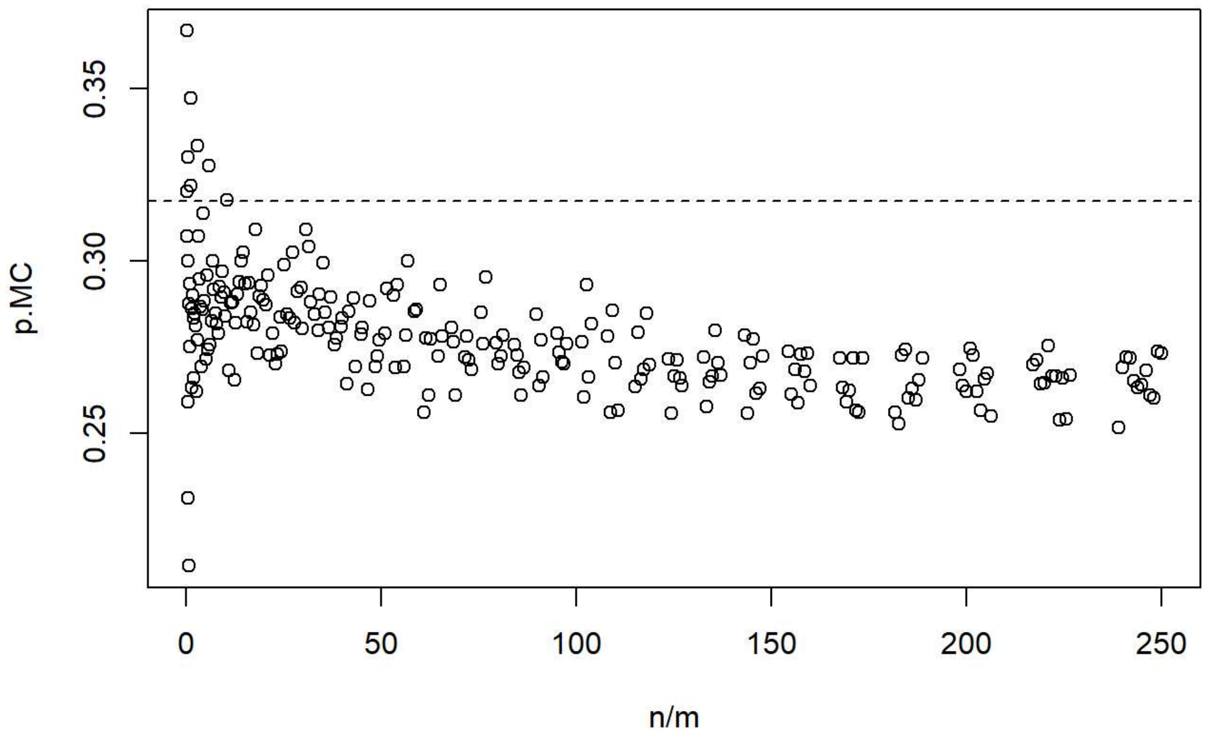
2.2.Biased MC estimates

More generally, for path-dependent payoffs MC is not necessarily unbiased

Fortunately, bias can be reduced by increasing number of steps (m) in time discretization

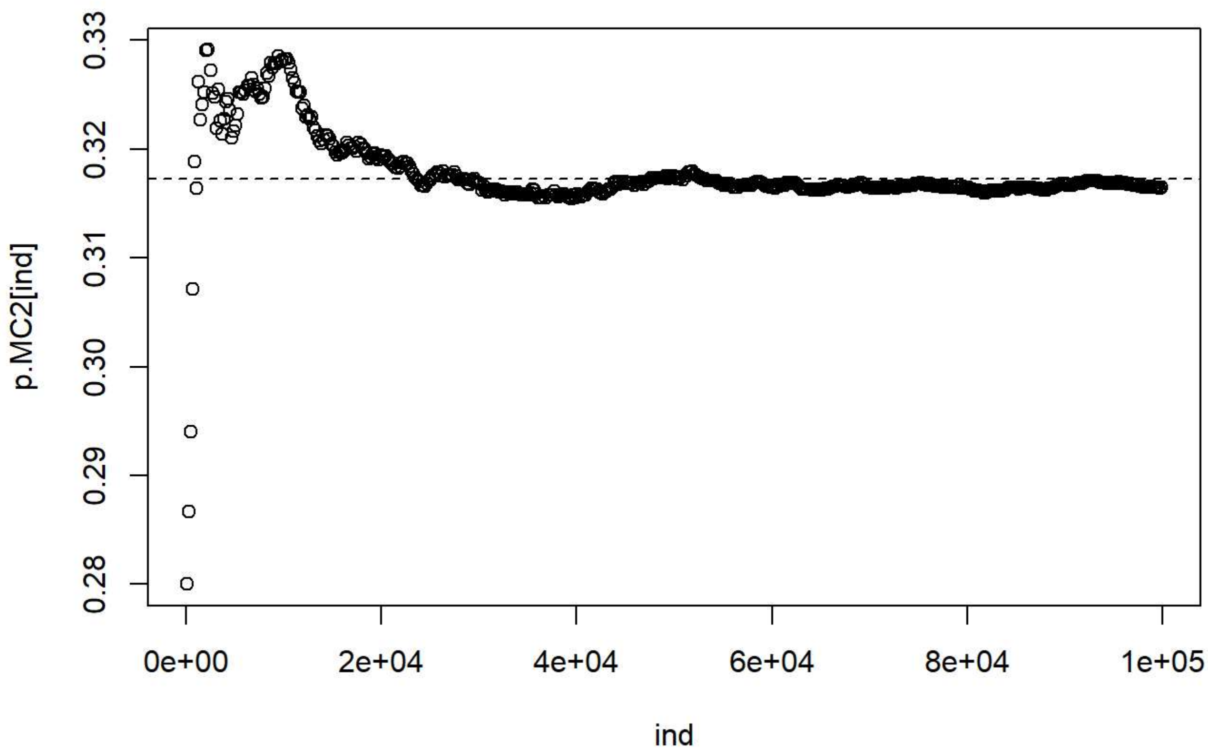
- Trade-off between # paths (n) & # steps (m)
  - $n \uparrow \Rightarrow \text{Var} \downarrow \quad \& \quad m \uparrow \Rightarrow \text{Bias} \downarrow$ (Bias-Variance Trade-off)

```
B=1; T=1 # expiration
nm=100000
n=seq(100,5000, by=20)
m=floor(nm/n)


###################
#### Biased MC estimates of hitting prob using path discretization
p.MC=NULL
for(i in 1:length(n)){
  Z=cbind( rep(0,n[i]), matrix(rnorm(n[i]*m[i]),n[i],m[i]))
  dt=T/m[i]
  W=t(apply(Z*sqrt(dt),1,cumsum))
  max.W=apply( W, 1, max)
  p.MC[i]= mean( max.W>B )
}

# plot of biased MC estimates vs n/m ratio
plot( n/m, p.MC); abline(h=p.true, lty=2)
```



2.3.Unbiased MC estimates (use Maxima Distribution)

```
####################
#### Unbiased MC estimate of hitting prob using distr. of maximum
n=100000; B=1; T=1

radn.max.W=abs(rnorm(n)*sqrt(T))
p.MC2= cumsum( radn.max.W>1 )/1:n

# plot of unbiased MC estimates vs n
ind=seq(100,n,by=200)
plot( ind, p.MC2[ind]); abline(h=p.true, lty=2)
```



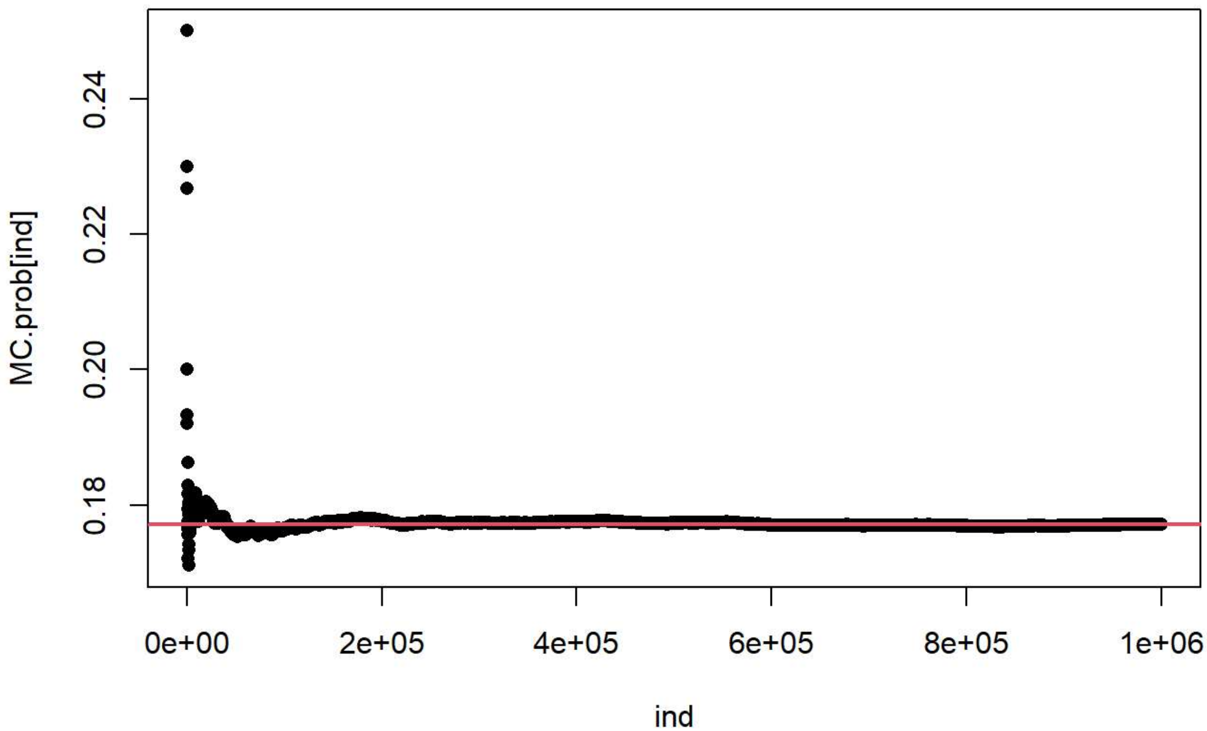
3.Extrema of Geometric BM

3.1.Brownian Bridge Simulation & True theoretical probability - max of GBM

```
n=1000000 # number of paths
r=.03 # risk-free rate
v=.25 # volatility
M=1 # expiration
S0=100 # starting asset price
K=90 # strike
B=140 # barrier

##### Simulating maximum of GBM using Brownian Bridge

Z=rnorm(n)
XT=(r-v^2/2)*M + v*sqrt(M)*Z
MT= (XT + sqrt( XT^2 -2*v^2*M*log(runif(n))))/2

# MC probability of hitting barrier
MC.prob=cumsum( S0*exp(MT) >= B )/1:n
ind=seq(100,n,by=100)
plot(ind,MC.prob[ind], pch=16)


##### True theoretical probability
lB=log(B/S0); nu = (r-v^2/2)
(true.prob=pnorm( (-lB+nu*M)/(v*sqrt(M)) ) + exp(2*nu*lB/v^2)*pnorm( (-lB-nu*M )/(v*sqrt(M)) ))
```

```
## [1] 0.1771409
```

```
abline( h=true.prob, lwd=2, col=2)
```



MC.prob[ind]
0.18
0.20
0.22
0.24
0e+00
2e+05
4e+05
6e+05
8e+05
1e+06
ind

3.2.Up Barrier - MC Price & True Theoretical Price of Barrier Option (Up-and-XXX) Call/Put

```
n=1000000 # number of paths
r=.03 # risk-free rate
v=.25 # volatility
M=1 # expiration
S0=100 # starting asset price
K=90 # strike
B=140 # barrier

##### Simulating maximum of GBM using Brownian Bridge

Z=rnorm(n)
XT=(r-v^2/2)*M + v*sqrt(M)*Z
MT= (XT + sqrt( XT^2 -2*v^2*M*log(runif(n))))/2

##### MC price of Up-and-XXXX call/Put (B>K)
ST=S0*exp(XT)
MST=S0*exp(MT)

Pui_payoff = exp(-r*M)*pmax(K-ST,0)*(MST>B) # Up-and-In Put (PUI)
Puo_payoff = exp(-r*M)*pmax(K-ST,0)*(MST<B) # Up-and-Out Put (PUO)
Cui_payoff = exp(-r*M)*pmax(ST-K,0)*(MST>B) # up-and-in call payoff
Cuo_payoff = exp(-r*M)*pmax(ST-K,0)*(MST<B) # up-and-out call payoff

payoff = Cuo_payoff # <<<<<=============== Set here ============== ||
(Up_MC.price=mean(payoff)) # MC price of Up-and-XXX call/Put
```

```
## [1] 8.094341
```

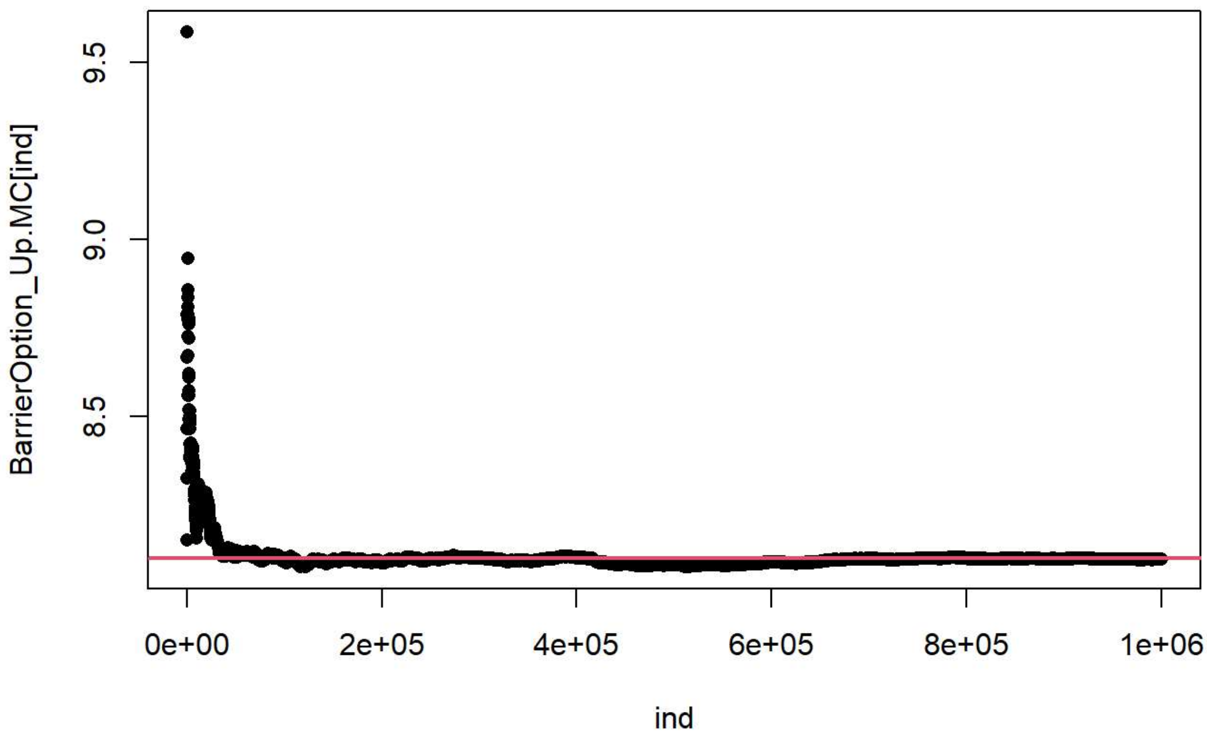
```
BarrierOption_Up.MC= cumsum(payoff)/1:n
plot(ind,BarrierOption_Up.MC[ind], pch=16)


###### True theoretical price of Up-and-XXX Call/Put
#install.packages("derivmkts")
library(derivmkts)
```

```
## Warning: package 'derivmkts' was built under R version 4.4.3
```

```
s=S0;strike=K;volatility=v;rf=r;tt=M;dividend_yield=0;Barrier = B

P_UI.true=putupin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_UO.true=putupout(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_UI.true=callupin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_UO.true=callupout(s,strike,volatility,rf,tt,dividend_yield,Barrier)

Price_UX.true = C_UO.true # <<<<<================ Set here ============== ||
print(paste("True Price by Package:",Price_UX.true))
```

```
## [1] "True Price by Package: 8.09694861987121"
```

```
abline(h=Price_UX.true, lwd=2, col=2)
```




```
# - - -  Alternative (Ignore)
# install.packages('fExoticOptions') # Not work in current R version
# library(fExoticOptions) # R library with exotic option price formulas
# StandardBarrierOption(TypeFlag = "cuo", S = 100, X = 90,
#                         H = 140, K = 0, Time = 1, r = 0.03, b = 0.03, sigma = 0.25)
# C_UO.true= 8.09697
# abline(h=C_UO.true, lwd=2, col=2)
```

True theoretical price of Barrier Options by package:

```
###### True theoretical price of Barrier Options
#install.packages("derivmkts")
library(derivmkts)
s=S0;strike=K;volatility=v;rf=r;tt=M;dividend_yield=0;Barrier = B
C_UO.true=callupout(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_UI.true=callupin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_DO.true=calldownout(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_DI.true=calldownin(s,strike,volatility,rf,tt,dividend_yield,Barrier)

P_UI.true=putupin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_UO.true=putupout(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_DI.true=putdownin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_DO.true=putdownout(s,strike,volatility,rf,tt,dividend_yield,Barrier)

print(C_UO.true)
```

```
## [1] 8.096949
```

3.3.Down Barrier - MC Price & True Theoretical Price of Barrier Option (Up-and-XXX) Call/Put

```
Z=rnorm(n)
XT.reflected=-(r-v^2/2)*M + v*sqrt(M)*Z #use negative drift for reflected path
MT.reflected=(XT.reflected + sqrt( XT.reflected^2 -2*v^2*M*log(runif(n))))/2 # reflected max

ST=S0*exp(-XT.reflected) # final value
mT=S0*exp(-MT.reflected) # minimum

payoff_2.a = exp(-r*M) * (ST-mT)
mean_2.a=mean(payoff_2.a) # MC estimate
se_2.a=sd(payoff_2.a)/sqrt(n) # st. dev
```

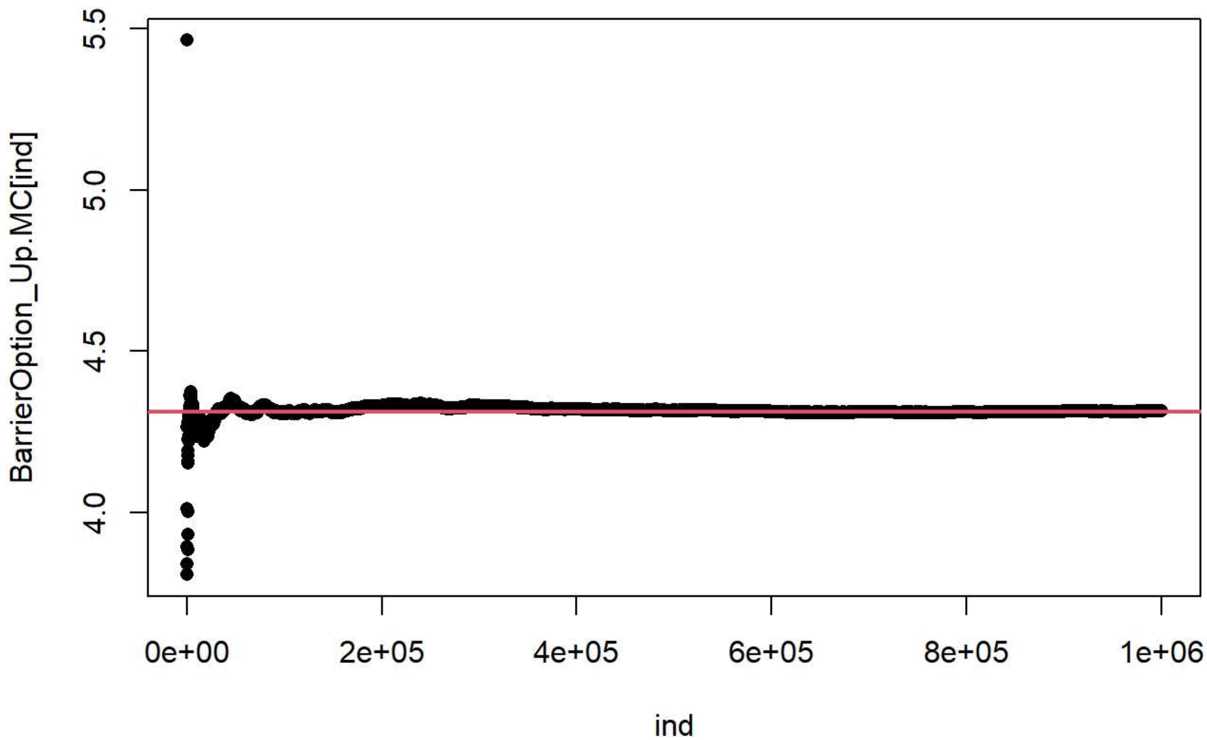
```
n=1000000 # number of paths
r=.03 # risk-free rate
v=.25 # volatility
M=1 # expiration
S0=100 # starting asset price
K=90 # strike
B=140 # barrier

##### Simulating maximum of GBM using Brownian Bridge

Z=rnorm(n)
XT.reflected=-(r-v^2/2)*M + v*sqrt(M)*Z # use negative drift for reflected path
MT.reflected=(XT.reflected + sqrt( XT.reflected^2 -2*v^2*M*log(runif(n))))/2 # reflected max

##### MC price of Up-and-XXXX call/Put (B>K)
ST=S0*exp(-XT.reflected) # final value
minST=S0*exp(-MT.reflected) # minimum

Pdi_payoff = exp(-r*M)*pmax(K-ST,0)*(minST<B) # Down-and-In Put (PUI)
Pdo_payoff = exp(-r*M)*pmax(K-ST,0)*(minST>B) # Down-and-Out Put (PUO)
Cdi_payoff = exp(-r*M)*pmax(ST-K,0)*(minST<B) # Down-and-in call payoff
Cdo_payoff = exp(-r*M)*pmax(ST-K,0)*(minST>B) # Down-and-out call payoff

payoff = Pdi_payoff # <<<<<================ Set here ============== ||
(Down_MC.price=mean(payoff)) # MC price of Down-and-XXX call/Put
```

```
## [1] 4.315432
```

```
BarrierOption_Up.MC= cumsum(payoff)/1:n
plot(ind,BarrierOption_Up.MC[ind], pch=16)


###### True theoretical price of Down-and-XXX Call/Put
#install.packages("derivmkts")
library(derivmkts)
s=S0;strike=K;volatility=v;rf=r;tt=M;dividend_yield=0;Barrier = B

C_DO.true=calldownout(s,strike,volatility,rf,tt,dividend_yield,Barrier)
C_DI.true=calldownin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_DI.true=putdownin(s,strike,volatility,rf,tt,dividend_yield,Barrier)
P_DO.true=putdownout(s,strike,volatility,rf,tt,dividend_yield,Barrier)

Price_DX.true = P_DI.true # <<<<<================ Set here ============== ||
print(paste("True Price by Package:",Price_DX.true))
```

```
## [1] "True Price by Package: 4.31197380056923"
```

```
abline(h=Price_DX.true, lwd=2, col=2)
```



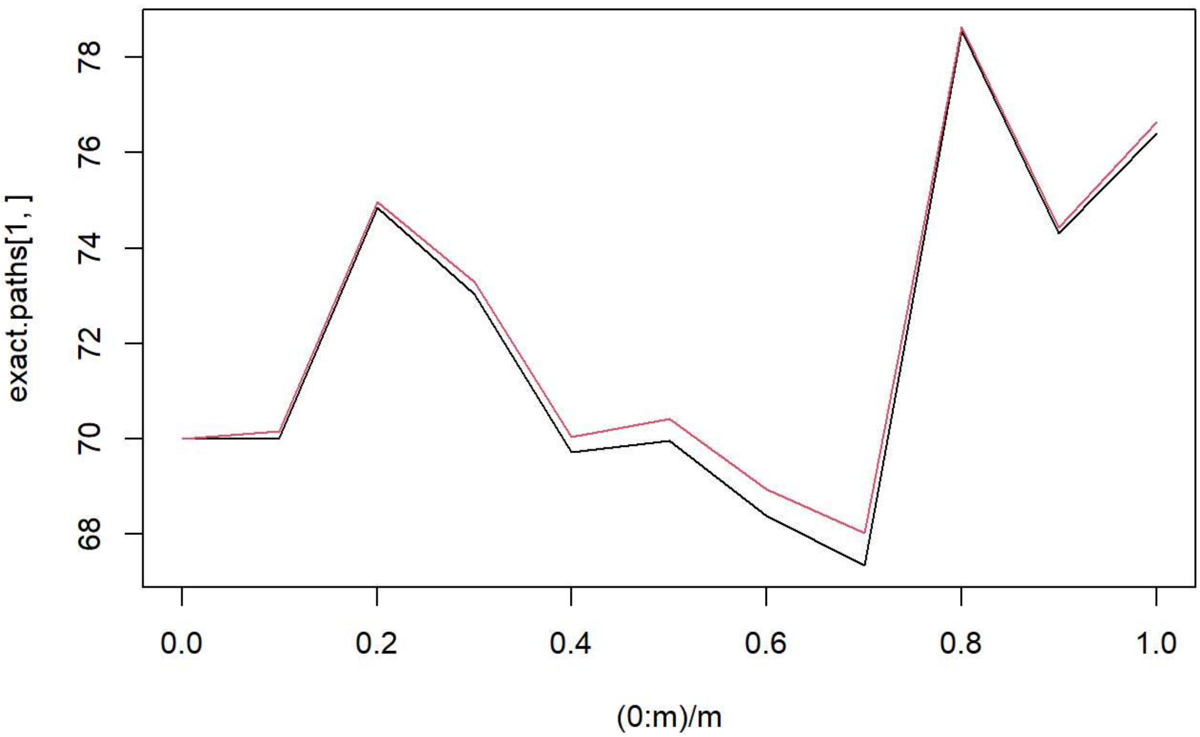
4.Euler Discretization

4.1.GBM discretized paths by Euler method (approximate)

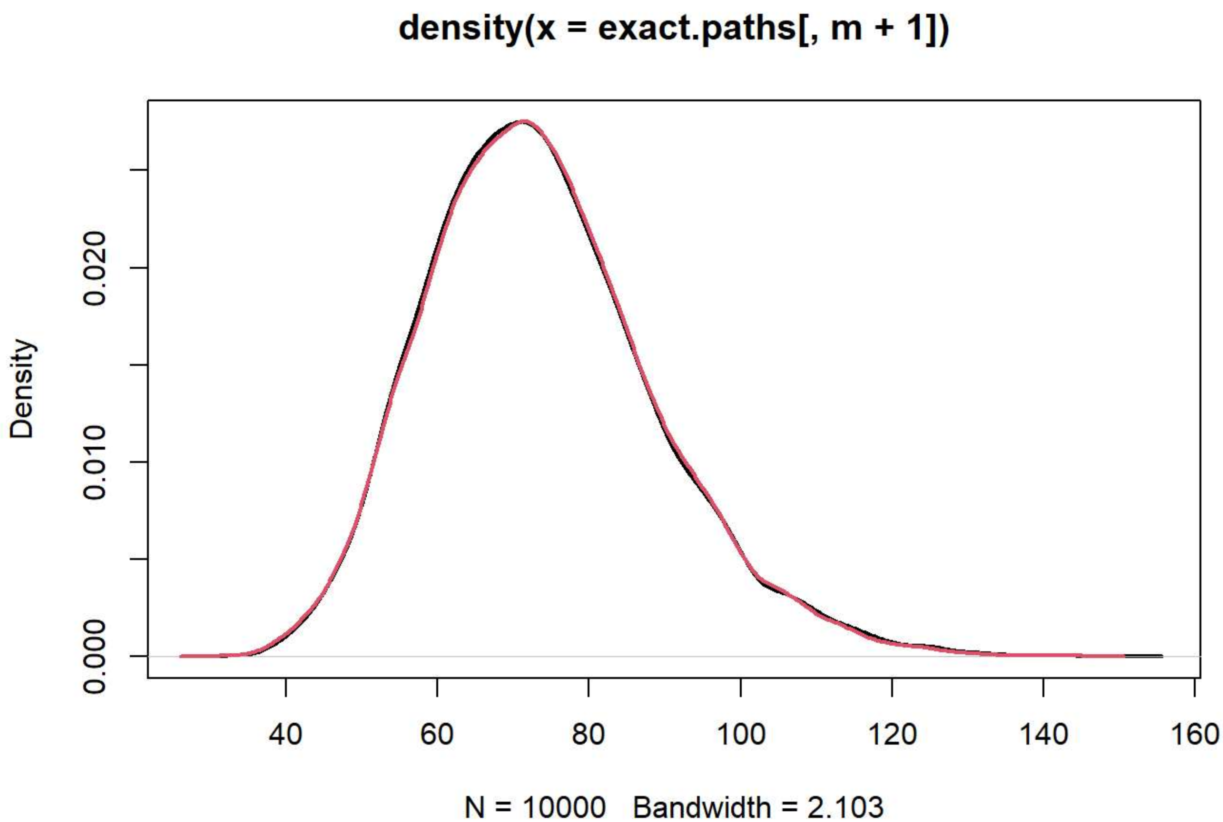
- Euler discretization of SDE: $dS_t = \mu(t, S_t) \times dt + \sigma(t, St) \times dW_t$
- Discrete Time: $t_i = i(T/m) = i\Delta t, \quad i = 0, \ldots, m$
- Simulate (approx.) path recursively, using:

$$S(t_i) = S_{t_{\text{prev}}} + \mu(t_{\text{prev}}, S_{t_{\text{prev}}}) \times \Delta t + \sigma(t_{\text{prev}}, S_{t_{\text{prev}}}) \times (\sqrt{\Delta t} \cdot Z_i) \;\; for\; i = 1, \ldots m, \; where\; Z_i \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1)$$

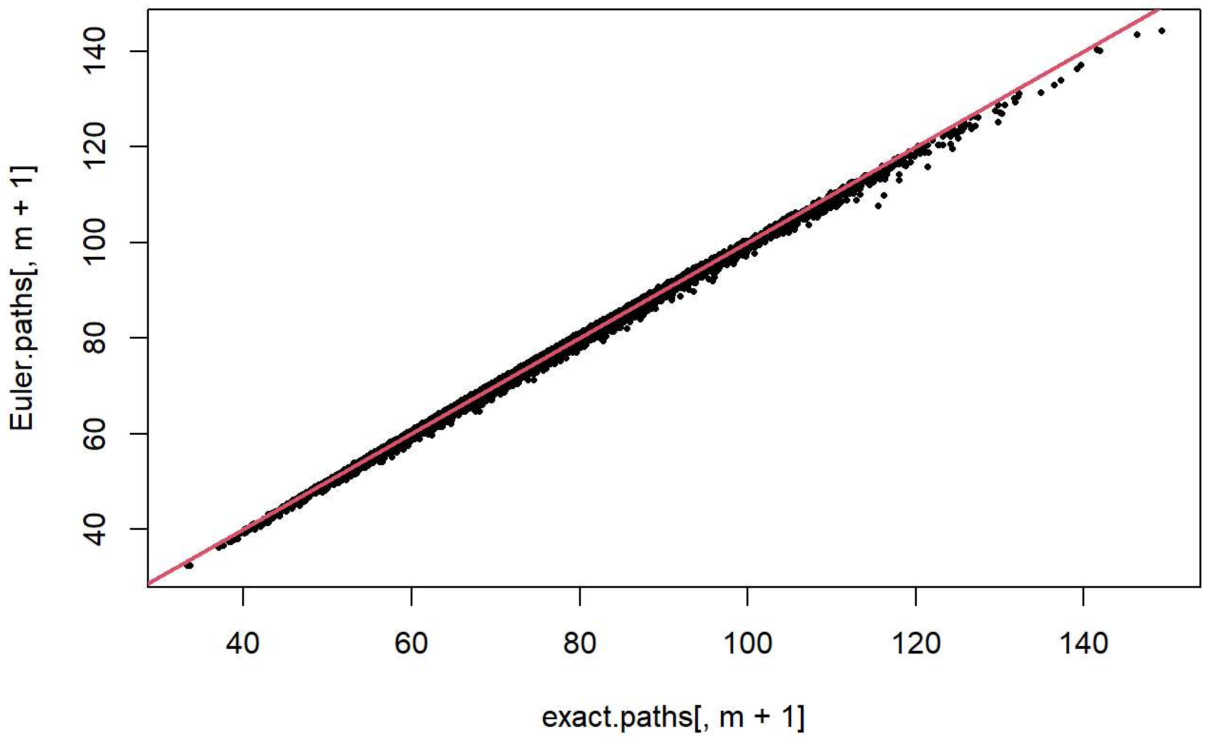
```
#####
# Euler Discretization GBM: dS_t = (r*S_t)*dt + (v*S_t)*dW_t

n=10000 # number of paths
m=10 # number of steps within each path
r=.05 # risk-free rate
v=.2 # volatility
M=1 # expiration
S0=70 # starting asset price

dt=M/m
mu=(r-v^2/2)*dt
sig=v*sqrt(dt)

Z=matrix(rnorm(m*n),n,m)

# EXACT GBM discretized paths
exact.paths=cbind( rep(S0,n) , t(S0*exp( apply( mu+sig*Z , 1, cumsum) )) )

# Euler method (approximate) GBM discretized paths
Euler.paths= matrix(S0, n, m+1)
for(i in 1:m){
  S_prev=Euler.paths[,i]
  Euler.paths[,i+1]=S_prev + (S_prev*r*dt + S_prev*v*sqrt(dt)*Z[,i])
}

plot((0:m)/m,exact.paths[1,], type='l') # Plot of exactly simulated GMB discretized path
lines((0:m)/m,Euler.paths[1,], type='l',col=2) # Plot of Euler (approx) simulated GMB discretize
d path
```




```
plot( density(exact.paths[,m+1]), lwd=2 ) # density estimate of S(1) for exactly simulated GMB
lines( density(Euler.paths[,m+1]), lwd=2, col=2 ) # density estimate of S(1) for Euler simulated
GMB
```



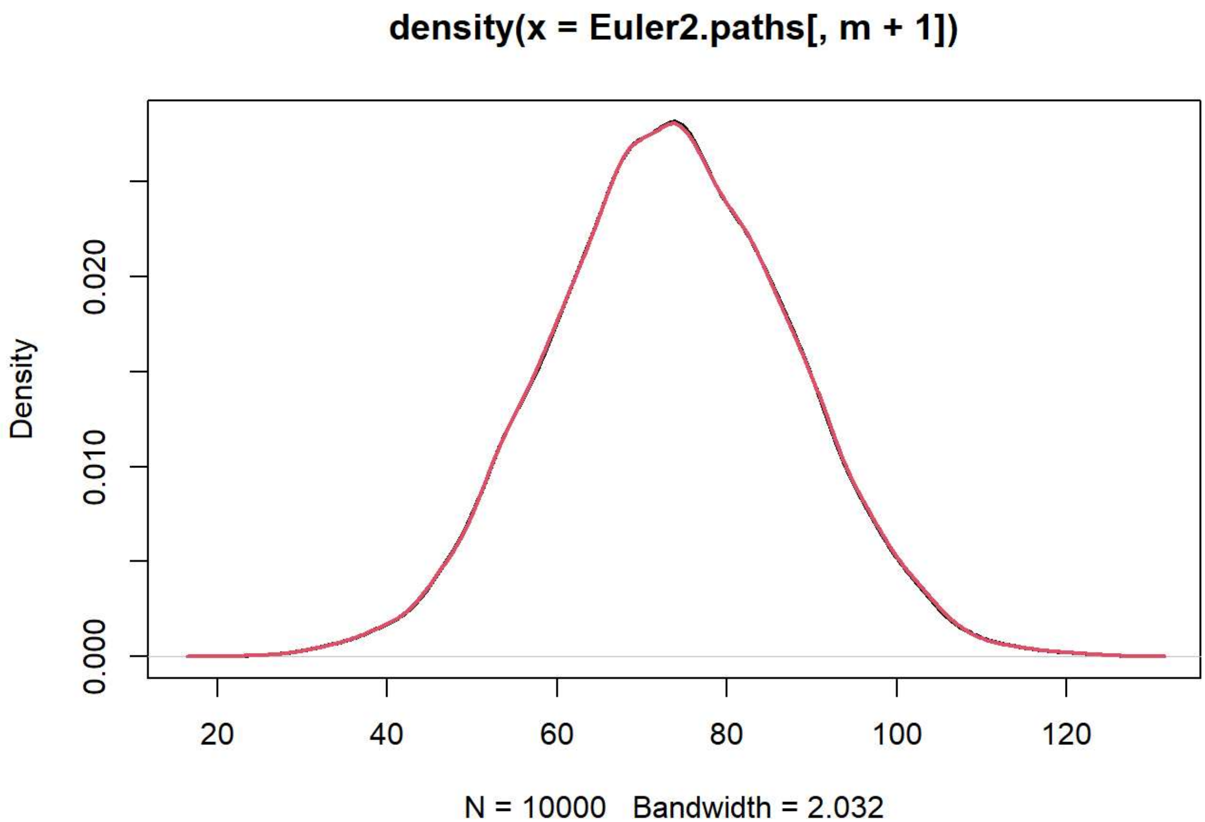

```
plot(exact.paths[,m+1],Euler.paths[,m+1], pch=16, cex=.5)
abline(0,1,lwd=2,col=2)
```



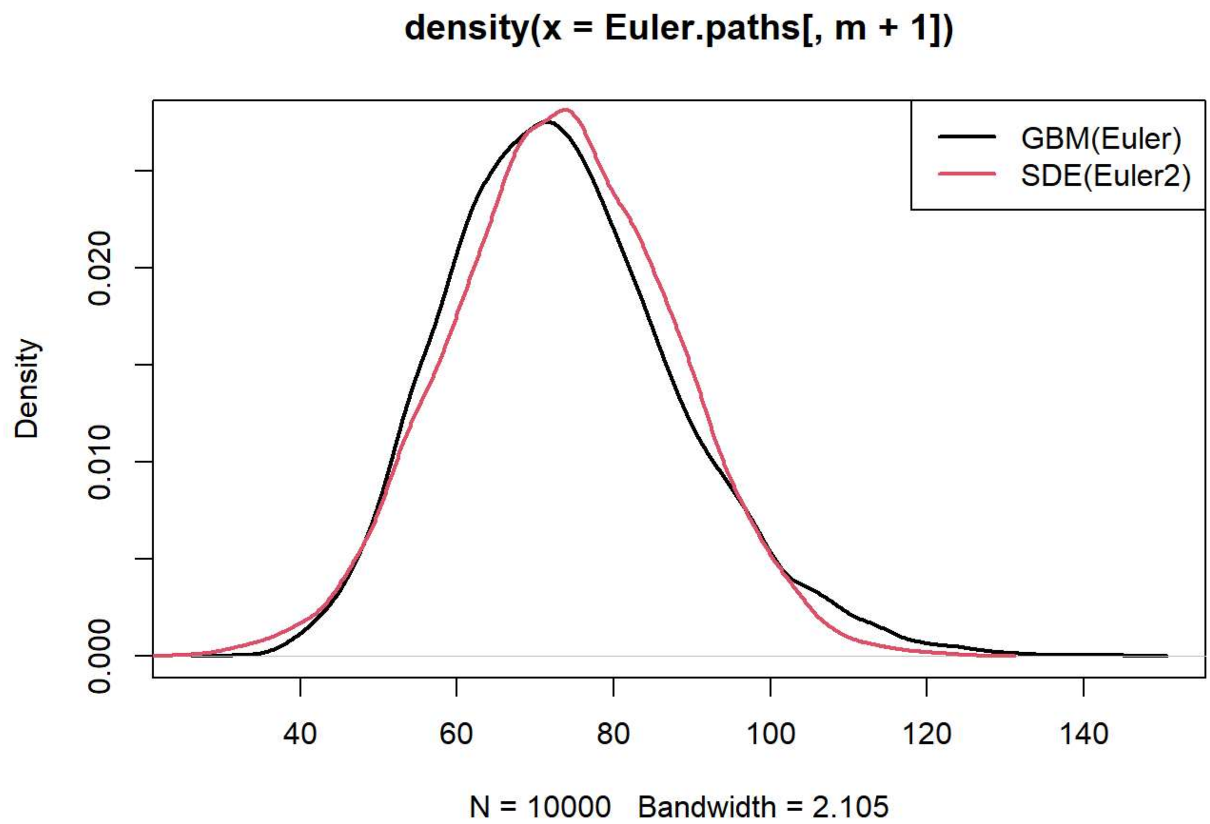
Euler.paths[, m + 1]
exact.paths[, m + 1]
40
60
80
100
120
140
40
60
80
100
120
140

4.2.Euler discretization Example

```
#####
## Euler discretization of dS_t = U(t,St)*dt + Sigma(t,St)*dW_t
## S(ti)=St_prev + U(t_prev,St_prev)*dt + Sigma(t_prev,St_prev)*(sqrt(dt)*Z[,i])

n=10000 # number of paths
m=10 # number of steps within each path
r=.05 # risk-free rate
v=.2 # volatility
M=1 # expiration
S0=70 # starting asset price

dt=M/m
mu=(r-v^2/2)*dt
sig=v*sqrt(dt)

Z=matrix(rnorm(m*n),n,m)


######
## Euler discretization of dS_t = (S_t*r)*dt + (S_0*v)*dW_t
## Simulate (approx.) path recursively:
## S(ti)=S_prev + (S_prev*r)*dt + (S0*v)*(sqrt(dt)*Z[,i])

Euler2.paths= matrix(S0, n, m+1)
for(i in 1:m){
  S_prev=Euler2.paths[,i]
  Euler2.paths[,i+1]=S_prev + (S_prev*r*dt + S0*v*sqrt(dt)*Z[,i])
}

######
## exact
ZZ=rowSums(Z)/sqrt(m)
SS=exp(r)*(S0+ZZ*v*S0*sqrt( (1-exp(-2*r))/(2*r) ))


plot( density(Euler2.paths[,m+1]), lwd=2 ) # density estimate of S(1) for GMB
lines( density(SS), lwd=2, col=2 ) # density estimate of S(1) for SDE
```




```
plot( density(Euler.paths[,m+1]), lwd=2,col=1 ) # density estimate of S(1) for GMB
lines( density(Euler2.paths[,m+1]), lwd=2, col=2 ) # density estimate of S(1) for SDE

legend("topright",legend = c("GBM(Euler)", "SDE(Euler2)"),col = c(1, 2), lwd = 2)
```



5.European lookback option - Extrema of Brownian Motion (ABM)

A European lookback option is a path-dependent option whose payoff at maturity depends on the maximum/minimum price of the underlying asset before maturity. There are two types of lookback options, namely fixed strike & floating strike, with payoffs:

	Call	Put
Fixed strike	$(M_T - K)_+$	$(K - m_T)_+$
Float strike	$(S_T - m_T)_+$	$(M_T - S_T)_+$

where $M_T = \max_{0\leq t\leq T}\{S_t\}$ and $m_T = \min_{0\leq t\leq T}\{S_t\}$.

5.1.Floating Strike (mT) European lookback call option

Perform Monte Carlo simulation for estimating the price of a **floating strike** European lookback call option. Assume the underlying asset price follows Geometric BM with $S_0 = 90, T = 1, r = 2\%, \sigma = 20\%$, and use **unbiased estimation** (i.e. simulate from the exact distribution of the minimum). Use $n = 100,000$ samples and create a 95% **confidence interval**. Compare you result with the exact price of the option, which is 14.26674.

```
set.seed(12345)

n=1000000 # number of paths
r=.02 # risk-free rate
v=.2 # volatility
M=1 # maturity
S0=90 # starting asset price

# use reflection trick for simulating minimum:
# minimum of arithmetic BM is minus the maximum
# of process with opposite drift

Z=rnorm(n)
XT.reflected=-(r-v^2/2)*M + v*sqrt(M)*Z #use negative drift for reflected path
MT.reflected=(XT.reflected + sqrt( XT.reflected^2 -2*v^2*M*log(runif(n))))/2 # reflected max
ST=S0*exp(-XT.reflected) # final value
mT=S0*exp(-MT.reflected) # minimum
payoff_2.a = exp(-r*M) * (ST-mT) # <<<<<<============ Payoff ==============||
mean_2.a=mean(payoff_2.a) # MC Price estimate
se_2.a=sd(payoff_2.a)/sqrt(n) # st. dev
CI_2.a=mean_2.a+c(-1,1)*qnorm(.975)*se_2.a # 95% CI

print( paste("MC Float Strike Lookback call price =", mean_2.a) )
```

```
## [1] "MC Float Strike Lookback call price = 14.2733852982329"
```

```
print( paste( c("MC Float Strike Lookback call 95% CI =", CI_2.a), collapse = "  " ) )
```

```
## [1] "MC Float Strike Lookback call 95% CI =  14.2487087208825  14.2980618755833"
```

```
print( paste("Exact Float Strike Lookback call price =", 14.2667437) )
```

```
## [1] "Exact Float Strike Lookback call price = 14.2667437"
```

5.2.Fixed strike (MT-K)+ European lookback call option

Perform Monte Carlo simulation for estimating the price of a **fixed strike European lookback call option**. Assume the underlying asset price follows Geometric BM with $S_0 = 90$, $K = 90$, $T = 1$, $r = 2\%$, $\sigma = 20\%$, and use unbiased estimation (i.e. simulate from the exact distribution of the maximum). Use $n = 100,000$ samples and create a 95% confidence interval. Compare you result with the exact price of the option, which is 16.04886.

```
set.seed(12345)

n=1000000 # number of paths
r=.02 # risk-free rate
v=.2 # volatility
M=1 # maturity
S0=90 # starting asset price
K=90 # fixed strike



Z=rnorm(n)
XT=(r-v^2/2)*M + v*sqrt(M)*Z
MT=S0 * exp ( (XT+sqrt(XT^2 -2*v^2*M*log(runif(n))))/2 )
payoff_2.b = exp(-r*M)*pmax(MT-K,0) # <<<<<<============ Payoff ==============||
mean_2.b=mean(payoff_2.b) # MC Price estimate
se_2.b=sd(payoff_2.b)/sqrt(n) # st. dev
CI_2.b=mean_2.b+c(-1,1)*qnorm(.975)*se_2.b # 95% CI

print( paste("MC Fixed Strike Lookback call price =", mean_2.b) )
```

```
## [1] "MC Fixed Strike Lookback call price = 16.0619820139605"
```

```
print( paste( c( "MC Fixed Strike Lookback call 95% CI =", CI_2.b), collapse = "  " ) )
```

```
## [1] "MC Fixed Strike Lookback call 95% CI =  16.0356847983757  16.0882792295453"
```

```
print( paste("Exact Fixed Strike Lookback call price =", 16.0488631) )
```

```
## [1] "Exact Fixed Strike Lookback call price = 16.0488631"
```

5.3.Price Process Simulate By Euler Discretization; Price Fixed strike (MT-K)+ European lookback call option

Repeat the previous part assuming the risk-neutral asset price dynamics:

$$dS_t = \mu(t, S_t)dt + \sigma(t, S_t)dW_t = (rS_t)dt + \left(\sigma\cos(e^t)\sqrt{S_t}\right)dW_t$$

Note that the **price process is not a Geometric Brownian Motion anymore**. Use **Euler discretization** with $m = 50$ steps and simulate $n = 10,000$ paths. In order to approximate the maximum of each path, simulate the maximum of each step using the result for the extrema of an arithmetic BM:

$$M_{t_i}|S_{t_{i-1}}, S_{t_i} = S_{t_{i-1}} + \frac{\Delta S_{t_i} + \sqrt{\Delta S_{t_i}^2 - 2 \cdot \Delta t \cdot \sigma^2(t_{i-1}, S_{t_{i-1}}) \cdot \log(U)}}{2}$$

where

$$M_{t_i} = \max_{t_{i-1} \le t \le t_i} \{S_t\}, \quad \Delta S_{t_i} = S_{t_i} - S_{t_{i-1}}, \quad \Delta t = t_i - t_{i-1} \& U \sim \text{Uniform}(0,1)$$

The maximum of the entire path will be the maximum over all steps in the path, i.e. $M_T = \max_{0 \le t \le T}\{S_t\} = \max_{i=0,\ldots,m}\{M_{t_i}\}$.

Include a 95% confidence interval with your answer.

```
set.seed(12345)
n=10000 # number of paths
r=.02 # risk-free rate
v=.2 # volatility
M=1 # maturity
S0=90 # starting asset price
K=90 # fixed strike
m=50 # number of steps

Dt=M/m ; ST=matrix(S0,n,m+1); MT=matrix(0,n,m); mT=matrix(0,n,m)
for(i in 1:m){
  ST.prev=ST[,i]
  ### ------- Step mu & Sigma ------- ###
  step.mu=(r*ST.prev)*Dt;
  step.sig=v*cos(exp(i*Dt)) * sqrt(ST.prev)*sqrt(Dt)
  ###
  DST=step.mu+step.sig*rnorm(n)
  ST[,i+1]=ST[,i]+DST
  MT[,i]=ST[,i]+(DST+sqrt(DST^2-2*step.sig^2*log(runif(n))))/2
}
MT.all=apply(MT,1,max)
ST.final=ST[,m+1]
payoff_2.c = exp(-r*M)*pmax(MT.all-K,0) # <<<<<<============ Payoff ==============||
mean_2.c=mean(payoff_2.c) # MC price estimate
se_2.c=sd(payoff_2.c)/sqrt(n) # st. dev
CI_2.c=mean_2.c+c(-1,1)*qnorm(.975)*se_2.c # 95% CI

print( paste("MC Fixed Strike Lookback call price =", mean_2.c) )
```

```
## [1] "MC Fixed Strike Lookback call price = 2.20954794422818"
```

```
print( paste( c( "MC Fixed Strike Lookback call 95% CI =", CI_2.c), collapse = "  " ) )
```

```
## [1] "MC Fixed Strike Lookback call 95% CI =  2.19522000544052  2.22387588301584"
```

6.(Multi-Assets) European rainbow option - Multi-Price Process Simulate By Euler Discretization

Consider a European rainbow option with payoff given by

$$\left(\max_i \{ S_T^{(i)} \} - \frac{1}{d} \sum_{i=1}^{d} S_T^{(i)} \right)_+ ,$$

i.e. the payoff is the maximum final price minus the average final price of all d assets (note that the maximum/average is over assets, not time). Estimate the price of this option using Monte Carlo simulation with $n = 10,000 \text{ d-dimensional paths}$, and provide a 95% confidence interval with your answer. Assume that $d = 5$, $K = 100$, $T = 1$, $r = 0.03$, $S^{(i)}(0) = 100$, $\forall i = 1, \ldots, d$, and that the assets follow the multivariate SDE:

$$dS(t) = r \circ S(t)dt + \sigma \circ \tilde{S}(t) \circ d\mathbf{W}(t)$$

for $\sigma = [.1 \quad .2 \quad .3 \quad .4 \quad .5]$, $\text{Corr}(W_i(1), W_j(1)) = .3, \forall i \neq j$. where $\tilde{S}(t) = [S^{(5)}(t) \quad \cdots \quad S^{(1)}(t)]^T$ is the reverse of $S(t)$.

This process does not follow a multivariate Geometric BM, so use Euler discretization with $m = 25$ steps to approximate the final prices of the assets.

```
n=10000 #<<<=== # paths ===||
m = 25 #<<<=== # Steps ===||
d = 5 # <<<=== dimension ===||
Rho= matrix(.3,d,d); diag(Rho)=1 # <<<===correlation matrix===||
V=c(1:5/10) # <<<===volatilities===||
Sig=Rho*(V%*%t(V)) # <<<=== covariance matrix ===||
L=chol(Sig) # Cholesky factorization of covariance matrix
Drft=matrix(r-V^2/2, n, d, byrow=TRUE) # drift


#### Euler discretization of dS_t = U(t,St)*dt + Sigma(t,St)*dW_t
# S(ti)=St_prev + U(t_prev,St_prev)*dt + Sigma(t_prev,St_prev)*(sqrt(dt)*Z[,i])
Dt=M/m; ST=matrix(S0, n, d)
for(i in 1:m){
  Z=matrix(rnorm(n*d),n,d)
  ST=ST+(r*ST*Dt + Z%*%L*sqrt(Dt)*ST[,d:1] ) # <<<<<<============ Payoff ==============||
  ST[ST<0]=0 # set possible negative prices to 0
}

maxT=apply(ST,1,max)
avgT=apply(ST,1,mean)
payoff_3=exp(-r*M)*(maxT-avgT)
mean_3=mean(payoff_3) # MC price estimate
se_3=sd(payoff_3)/sqrt(n)
CI_3=mean_3+c(-1,1)*qnorm(.975)*se_3

print(paste("rainbow option price =", mean_3))
```

```
## [1] "rainbow option price = 30.3007746721434"
```

```
print(paste(c("rainbow option 95% CI =", CI_3), collapse = "  "))
```

```
## [1] "rainbow option 95% CI =  29.9847120401529  30.6168373041339"
```