Monte Carlo Simulation

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.Brownian Motion 1D (GBM)

Consider the arithmetic Brownian motion $X_t = \mu t + \sigma W_t$, where $\{W_t\}$ is standard Brownian motion. Find the conditional distribution of

$$\begin{bmatrix} X_{t_1} \\ \vdots \\ X_{t_n} \end{bmatrix} | X_1 = x, \quad \text{where } 0 < t_1 < \ldots < t_n < 1,$$

also known as the multivariate Brownian bridge. Write the parameters of the distribution in terms of $\mu, \sigma, t_1, \ldots, t_n$.

Solution: The (unconditional) joint distribution is

$$\begin{bmatrix} X_{t_1} \\ \vdots \\ X_{t_n} \\ X_1 \end{bmatrix} \sim N\left(\mu \begin{bmatrix} t_1 \\ \vdots \\ t_n \\ 1 \end{bmatrix}, \quad \sigma^2 \begin{bmatrix} t_1 & \cdots & t_1 & t_1 \\ \vdots & \ddots & \vdots & \vdots \\ t_1 & \cdots & t_n & t_n \\ t_1 & \cdots & t_n & 1 \end{bmatrix} \right)$$

Using the Normal conditional distribution formula, we get:

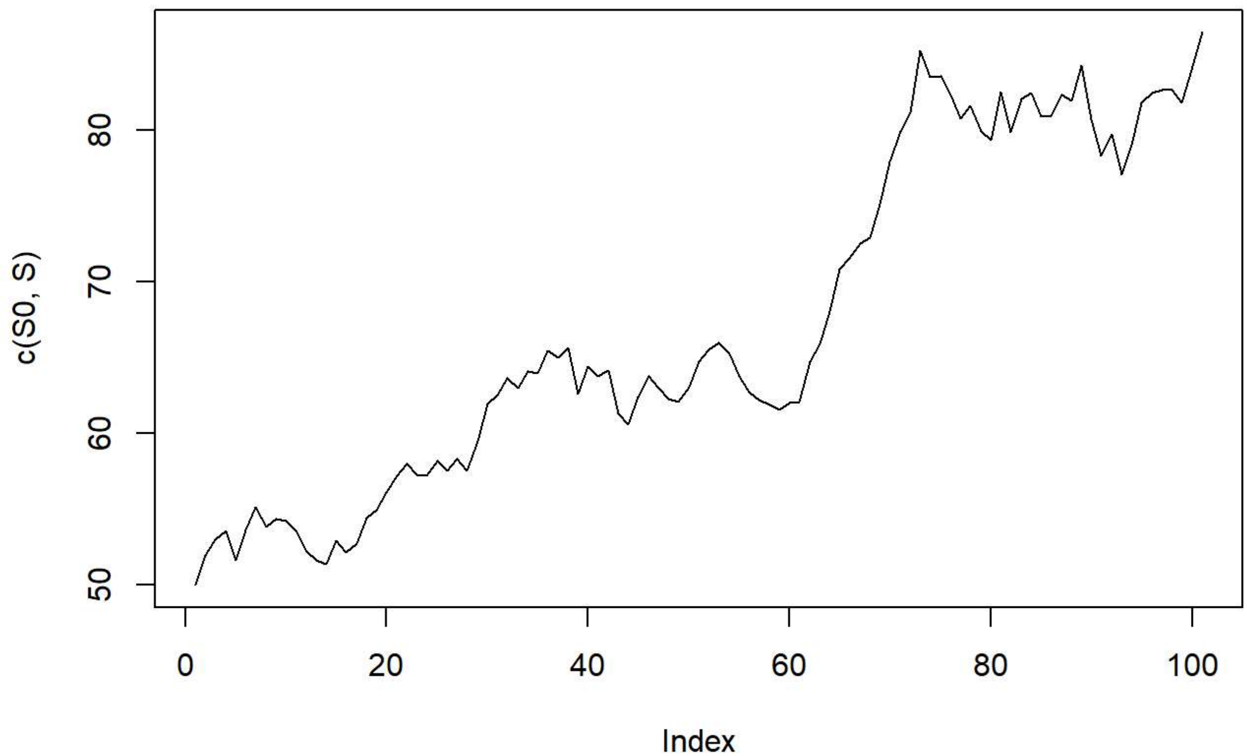
$$\begin{bmatrix} X_{t_1} \\ \vdots \\ X_{t_n} \end{bmatrix} |(X_1 = x) \sim N\left(\mu \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix} + \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix} (x - \mu), \quad \sigma^2 \left(\begin{bmatrix} t_1 & \cdots & t_1 \\ \vdots & \ddots & \vdots \\ t_1 & \cdots & t_n \end{bmatrix} - \begin{bmatrix} t_1 \\ \vdots \\ t_n \end{bmatrix} \begin{bmatrix} t_1 & \cdots & t_n \end{bmatrix} \right) \right)$$

$$\sim N\left(x\begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_n \end{bmatrix}, \quad \sigma^2 \begin{bmatrix} t_1(1-t_1) & t_1(1-t_2) & \cdots & t_1(1-t_n) \\ t_1(1-t_2) & t_2(1-t_2) & \cdots & t_2(1-t_n) \\ \vdots & \vdots & \ddots & \vdots \\ t_1(1-t_n) & t_2(1-t_n) & \cdots & t_n(1-t_n) \end{bmatrix} \right)$$

1.1.Unconditional BM (ABM,GBM)

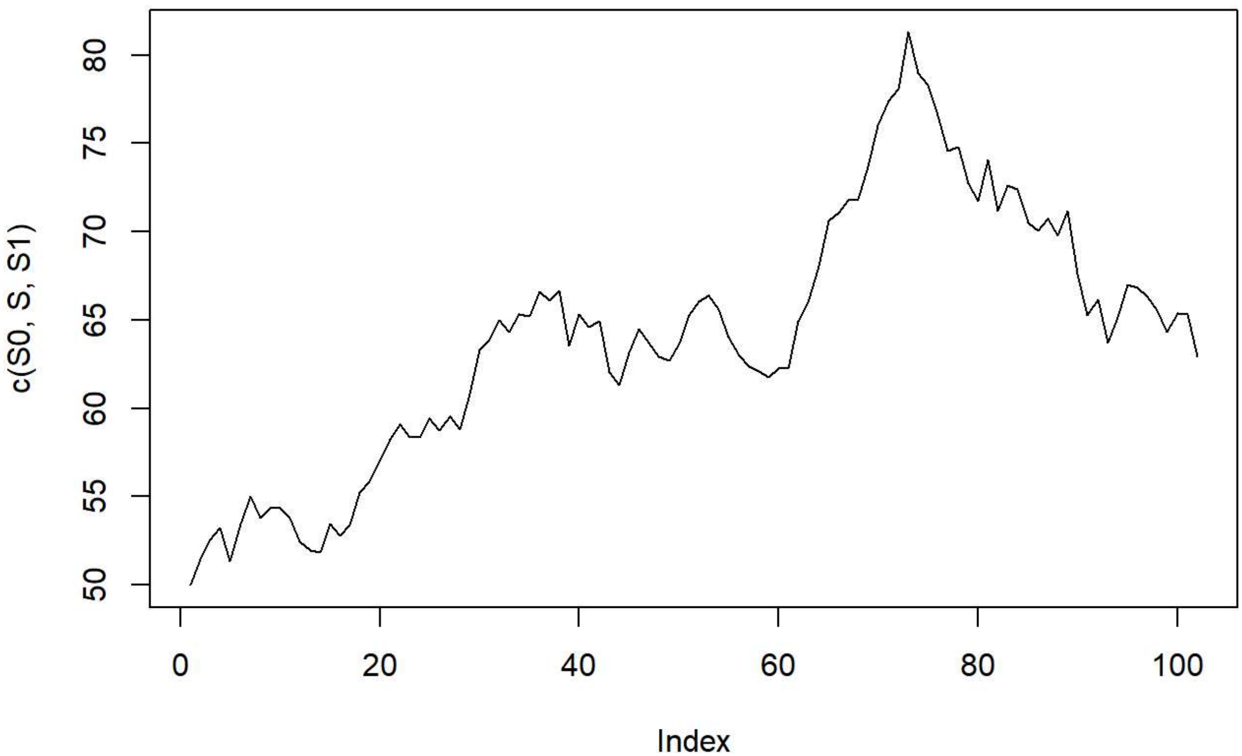
Consider the geometric Brownian motion $S_t = 50\exp\{.03t + .2W_t\}$; fix $n = 100$ and let $t_i = i/(n+1), \quad i = 0, \dots, n$. Write R code that simulates and plots a conditional path of the process, sampled at times $\{t_i\}_{i=0}^{n+1}$, given that $W_1 = 1$. Use the result of the previous question combined with the Cholesky decomposition for generating the correlated values of W at $(t_1, \dots, t_n)$.

```
### unconditional ABM & GBM (Not given W1=1)

set.seed(1234567890)
S0=50
mu=.03 # std BM: mu=0
sigma=.2 # std BM: sigma=1
n=100 # number of interior points
t.i=(1:n+1)/(n+1) # Time points from (1/101, 2/101, ..., 100/101,1)

MU.X=t.i*mu # mean
mat1=matrix(t.i,n,n)
mat2=matrix(t.i,n,n,byrow=T)
SIGMA.X= (pmin(mat1,mat2))*sigma^2 # ABM covariance matrix
L=chol(SIGMA.X) # Cholesky decomposition of SIGMA.X

Z=rnorm(n)
X=MU.X + t(L)%*%Z #ABM
S=S0*exp(X) #GBM
plot(c(S0,S), type='l') # Plot GBM
```



1.2.Brownian Bridge (ABM, GBM)

```
### BM (given W1=1)

set.seed(1234567890)
S0=50
mu=.03
sigma=.2
n=100 # number of interior points
t.i=(1:n)/(n+1) # Time points from (1/101, 2/101, ..., 100/101), exclude 0 and 1

X1=mu+sigma*1 # Xt=u*t+sigma*Wt
S1=S0*exp(X1)  # Final asset price at time 1
MU.X=t.i*X1 # conditional mean
mat1=matrix(t.i,n,n)
mat2=matrix(t.i,n,n,byrow=T)
SIGMA.X= (pmin(mat1,mat2) - t.i %x% t(t.i))*sigma^2 # conditional covariance matrix
L=chol(SIGMA.X) # Cholesky decomposition of SIGMA.X

Z=rnorm(n)
X=MU.X + t(L)%*%Z #ABM
S=S0*exp(X) #GBM
plot(c(S0,S,S1), type='l')
```



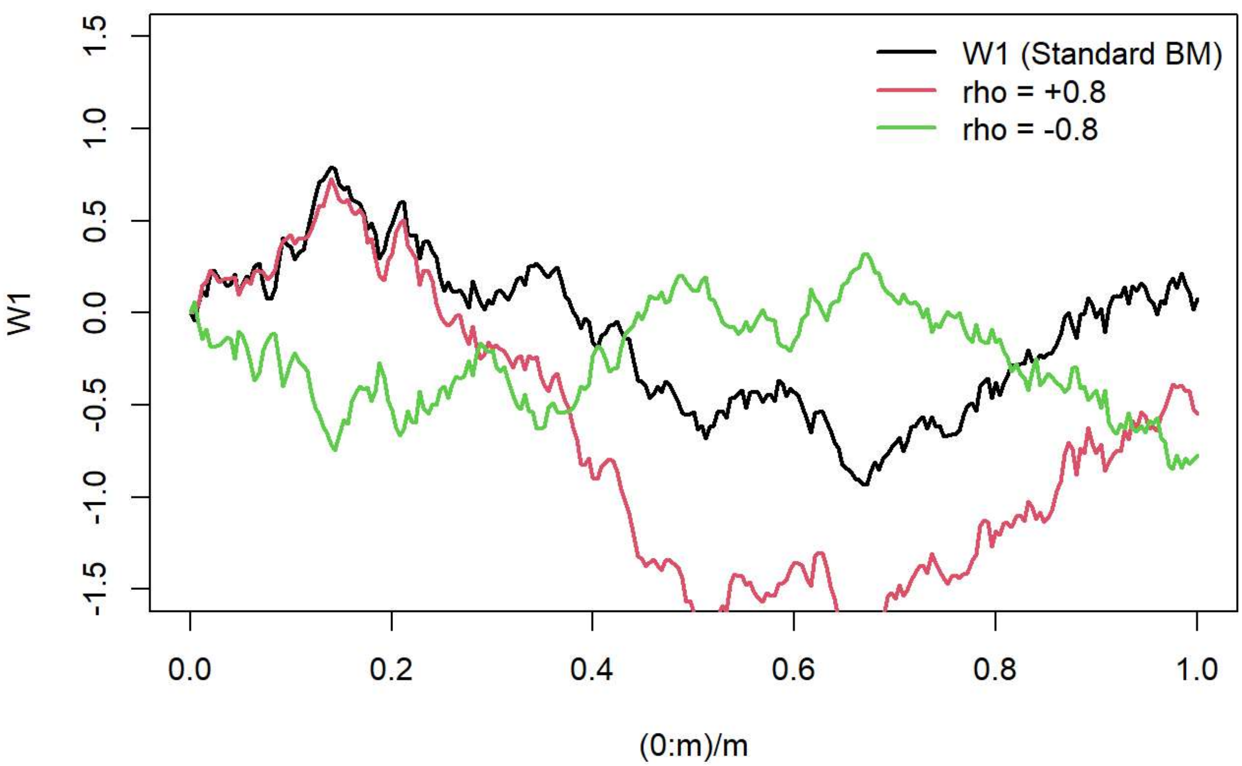

```
### BM (given Wb=a)
```

2.Correlated Brownian Motion

2.1.Simulating Correlated Standard BM

Simulating and Visualizing standard Brownian Motion paths with positive and negative correlations

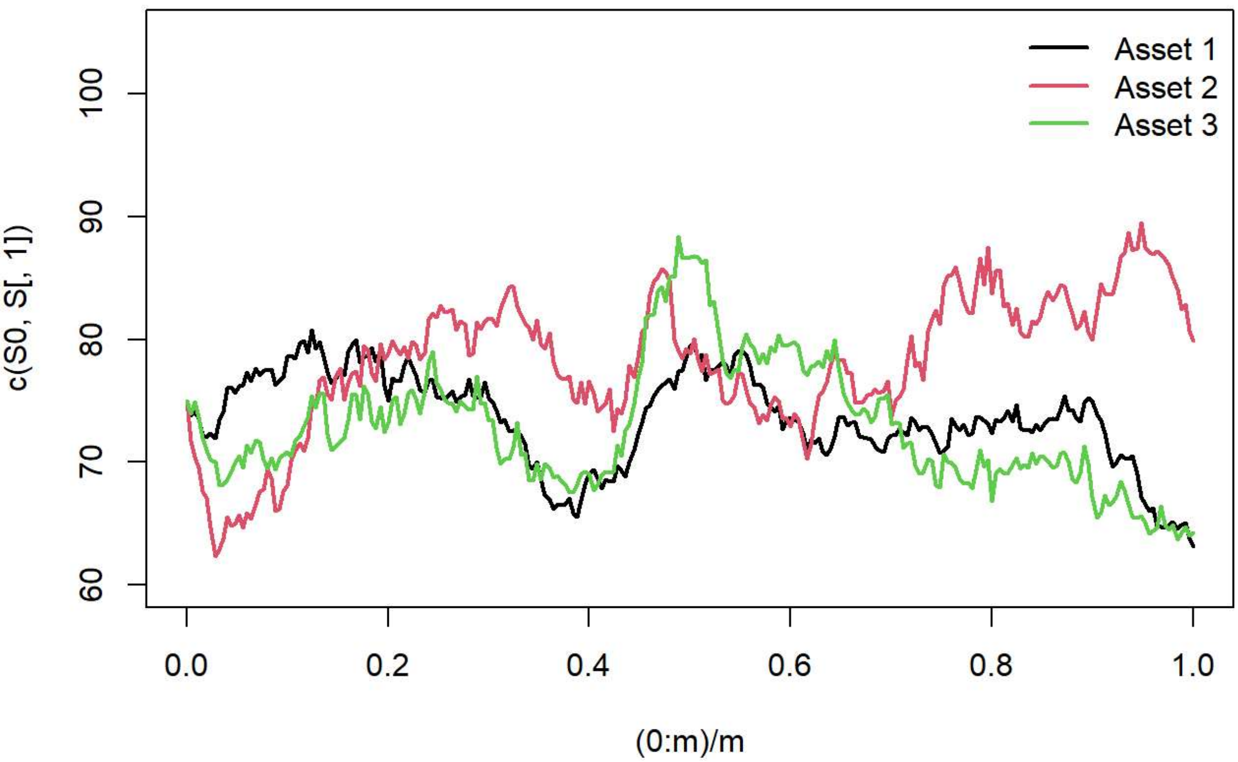
```
#####
# Correlated Standard BM: u=0, sigma=1

m=250 # number of steps

Z1=rnorm(m)
W1=c(0, cumsum(Z1*sqrt(1/m))) # 1st BM path
plot((0:m)/m,W1,type='l', lwd=2, ylim=c(-1.5,1.5)) # 1

rho=+.8
Z2=rho*Z1+sqrt(1-rho^2)*rnorm(m)
W2=c(0,cumsum(Z2*sqrt(1/m))) # 2nd BM path w/ rho=.8
lines((0:m)/m,W2, lwd=2, col=2)

rho=-.8
Z2=rho*Z1+sqrt(1-rho^2)*rnorm(m)
W2=c(0,cumsum(Z2*sqrt(1/m))) # 2nd BM path w/ rho=-.8
lines((0:m)/m,W2, lwd=2, col=3)


legend("topright",legend=c("W1 (Standard BM)", "rho = +0.8", "rho = -0.8"),col=c(1,2,3),lty=
1,lwd=2,bty="n")
```



2.2.Simulate Multivariate Geometric BM

Simulating Correlated 3D Geometric BM

```
#####
# Correlated 3D Geometric BM

### 1.INPUT - parameters

m = 250             # number of steps
Dt = 1 / m          # step size
Rho = matrix(0.4, 3, 3); diag(Rho) = 1  # correlation matrix
V = c(0.2, 0.3, 0.25)                   # volatilities (sigma, std dev)
Sig = Rho * (V %*% t(V))                # covariance matrix
L = chol(Sig)                           # Cholesky factorization
r = 0.02                                # risk free rate
Drft = r - V^2 / 2                      # drift term
S0 = 75                                 # Assume all assets start at 75

### 2.Simulates paths for all assets bt correlated Brownian motion.

Z=matrix( rnorm(m*3) , m, 3)
S=S0*exp(apply(Drft*Dt+ Z%*%L*sqrt(Dt),2, cumsum)) # Generate 3D path


### 3.Plots all GBM asset paths on the same time scale.

plot((0:m)/m, c(S0,S[,1]), type='l', lwd=2, ylim=c(60,105)) # Asset 1
lines((0:m)/m, c(S0,S[,2]), lwd=2, col=2) # Asset 2
lines((0:m)/m, c(S0,S[,3]), lwd=2, col=3) # Asset 3
legend("topright", legend = c("Asset 1", "Asset 2", "Asset 3"),  col = c(1, 2, 3),  lwd = 2,
bty = "n")  # bty="n" removes the box
```



- The system $\{S(t)\}$ follows the SDE:

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

The solution is given by $S(t) = \exp\{X(t)\}$, where:

$$dX(t) = \left(\mu - \frac{\sigma^2}{2}\right)dt + \sigma\,d\mathbf{W}(t)$$

Generate geometric BM variates as:

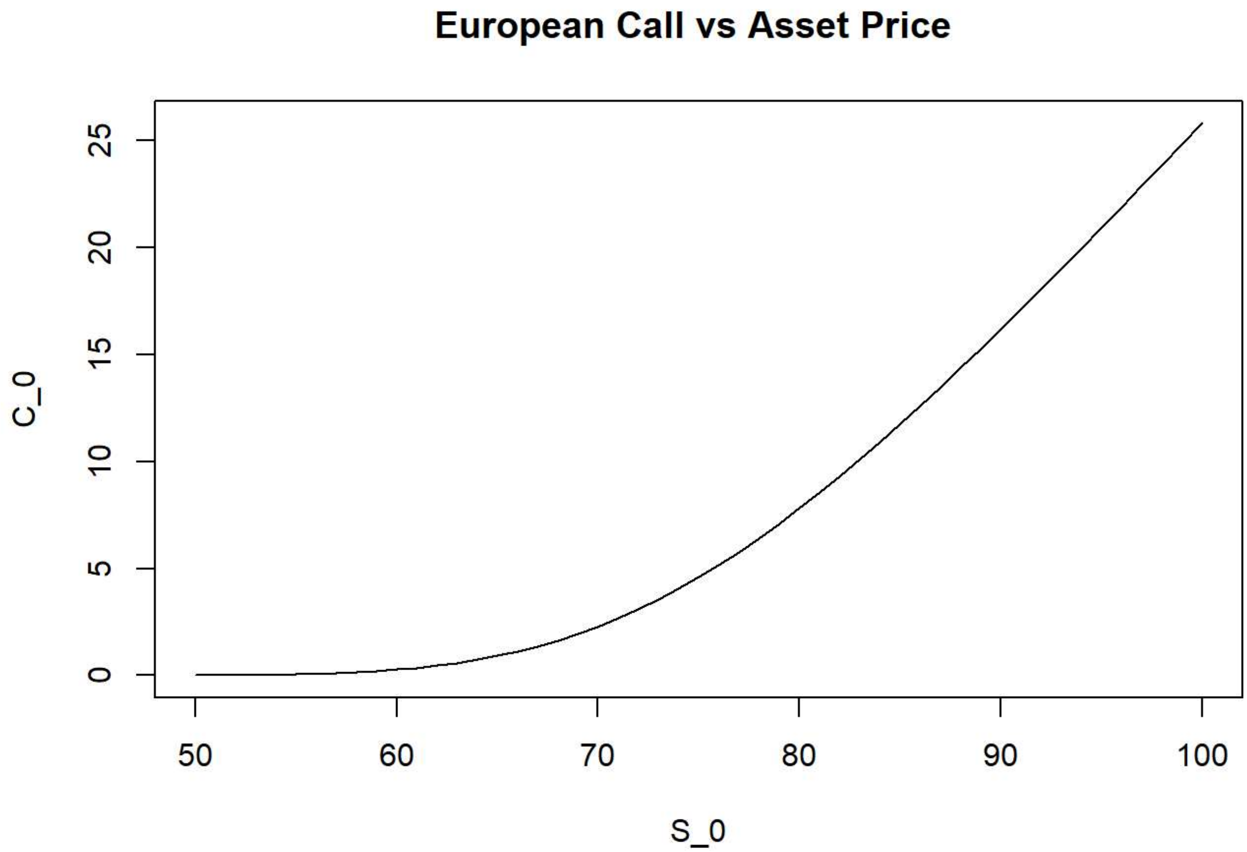
$$S(t_i) = S(t_{i-1}) \circ \exp\left\{\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + LZ_i\sqrt{\Delta t}\right\}, \quad i = 1, \ldots, m$$

  - Where: $Z_i \overset{iid}{\sim} N_d(0, I)$ (standard multivariate normal) ; $LL^T = \Sigma = (\sigma\sigma^T) \circ \rho$ (Cholesky decomposition of covariance matrix) ; ○ denotes element-wise product - ρ is the correlation matrix
  - μ: Vector of drift rates
  - σ: Vector of volatilities
  - $\mathbf{W}(t)$: Vector of correlated Brownian motions
  - Δt: Time step size
  - L: Lower triangular Cholesky factor

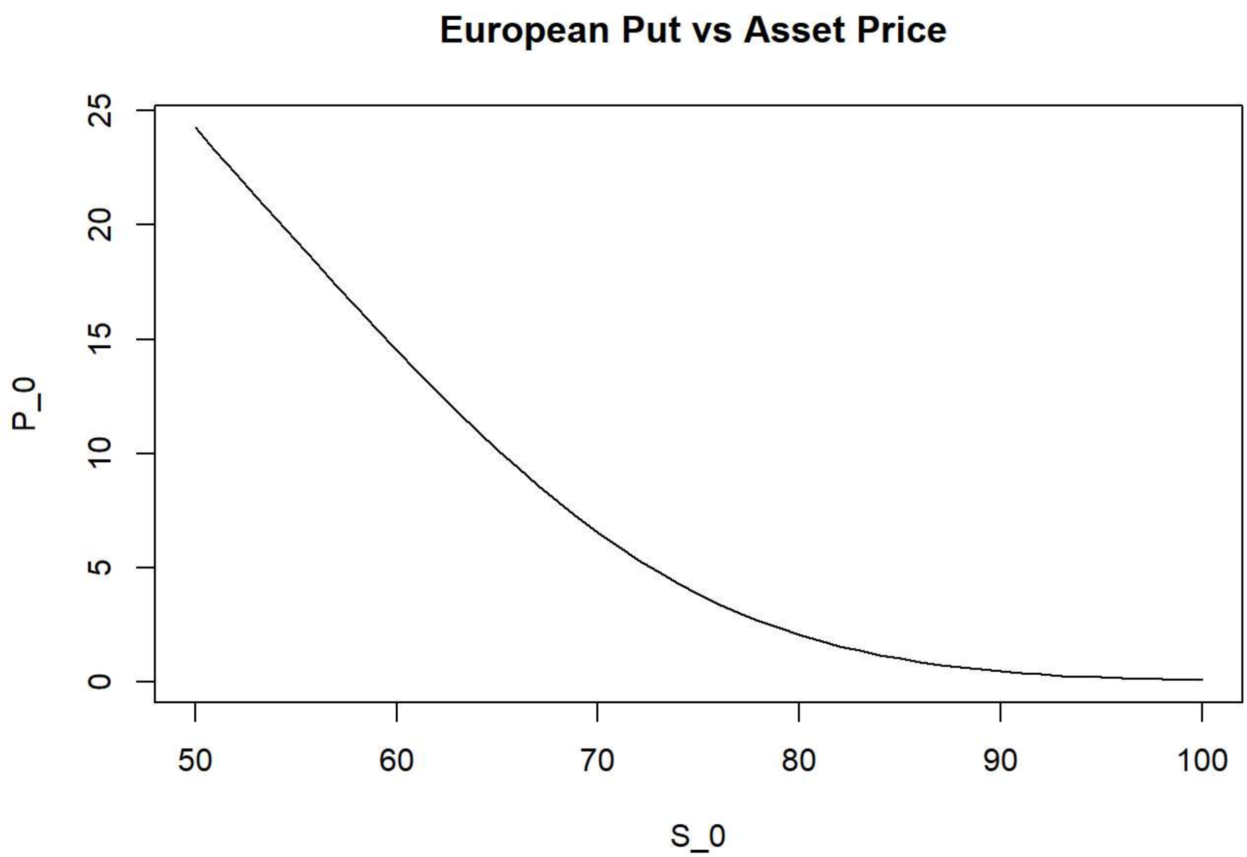
3.Black-Scholes Pricing Function

3.1.European Options

```
#####
# Function to compute Black-Scholes prices
# of European call & put options

BSprice=function(S, X, r, M, v){
  # Returns 2-column matrix: Col1: Call prices; Col2: Put prices
  # arguments: Asset price (S0), Strike (X), risk-free rate (r)
  # Maturity (M), volatility (v)

  d1=(log(S/X)+(r+0.5*v^2)*M)/(v*sqrt(M))
  d2=d1-v*sqrt(M)

  call.pr= S*pnorm(d1) - X*exp(-r*M)*pnorm(d2)
  put.pr=X*exp(-r*M) * pnorm(-d2) - S*pnorm(-d1)

  return(cbind(call.pr,put.pr))
}


prices = BSprice(50:100,75,.02,.5,.2)

plot(50:100, prices[,1], type='l', xlab="S_0", ylab="C_0",
     main="European Call vs Asset Price ")
```




```
plot(50:100, prices[,2], type='l', xlab="S_0", ylab="P_0",
     main="European Put vs Asset Price ")
```



call: payoff=$(S_T - K)_+ = S_T \cdot I_{(S_T \geq K)} - KI_{(S_T \geq K)}$, price=$S_0N(d_1) - Ke^{-rT}N(d_2)$

put: payoff=$(K - S_T)_+$, price= $Ke^{-rT}N(-d_2) - S_0N(-d_1)$

3.2.General Case

AoN1(asset_or_nothing): Payoff= $S_T \cdot I_{(S_T \geq K)}$, Price=$S_0N(d_1)$

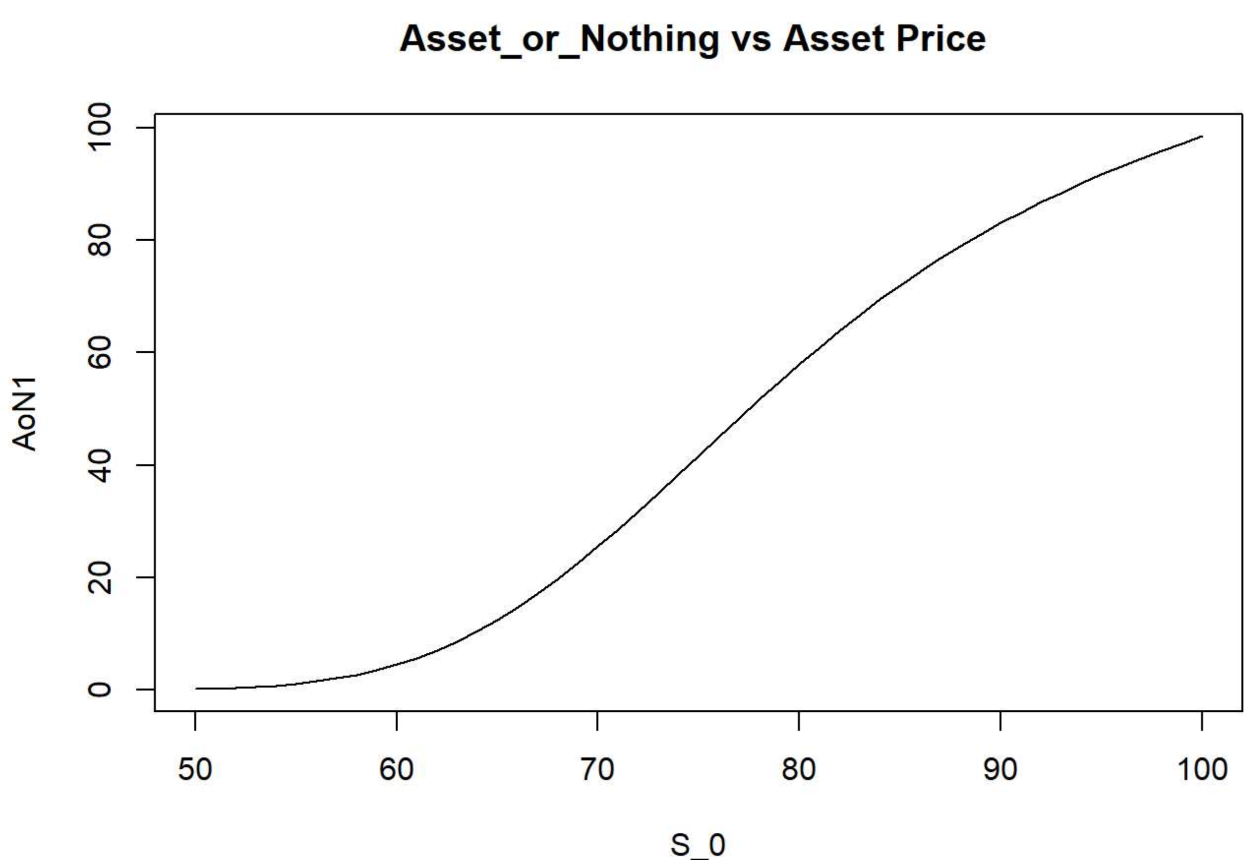
CoN1(cash-one-dollar_or_nothing): Payoff= $I_{(S_T \geq K)}$, Price=$e^{-rT}N(d_2)$

–

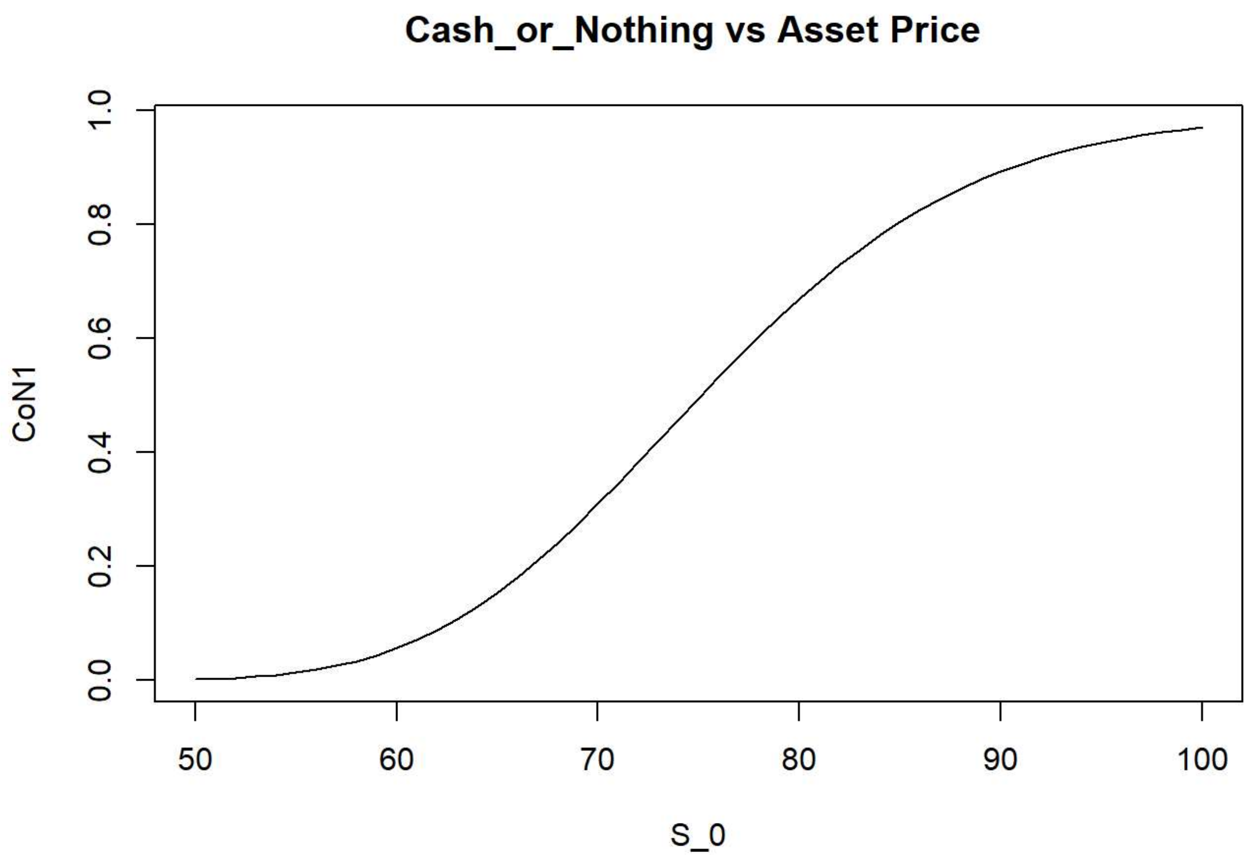
AoN2(other direction, asset_or_nothing): Payoff= $S_T \cdot I_{(S_T \leq K)}$,Price=$S_0N(-d_1)$

CoN2(other direction, cash-one-dollar_or_nothing): Payoff= $I_{(S_T \leq K)}$, Price=$e^{-rT}N(-d_2)$

```
General_BSprice=function(S, X, r, M, v){
  # Returns 2-column matrix: Col1: Call prices; Col2: Put prices
  # arguments: Asset price (S0), Strike (X), risk-free rate (r)
  # Maturity (M), volatility (v)

  d1=(log(S/X)+(r+0.5*v^2)*M)/(v*sqrt(M))
  d2=d1-v*sqrt(M)

  AoN1 = S*pnorm(d1); CoN1 = exp(-r*M)*pnorm(d2)
  AoN2 = S*pnorm(-d1); CoN2 = exp(-r*M)*pnorm(-d2)

  # call.pr= AoN1-X*CoN1 = S*pnorm(d1) - X*exp(-r*M)*pnorm(d2)
  # put.pr= X*CoN2-AoN2 = X*exp(-r*M)*pnorm(-d2) - S*pnorm(-d1)

  return(cbind(AoN1,CoN1,AoN2,CoN2))
}



prices = General_BSprice(50:100,75,.02,.5,.2)

plot(50:100, prices[,1], type='l', xlab="S_0", ylab="AoN1",
     main="Asset_or_Nothing vs Asset Price ")
```



```
plot(50:100, prices[,2], type='l', xlab="S_0", ylab="CoN1",
     main="Cash_or_Nothing vs Asset Price ")
```

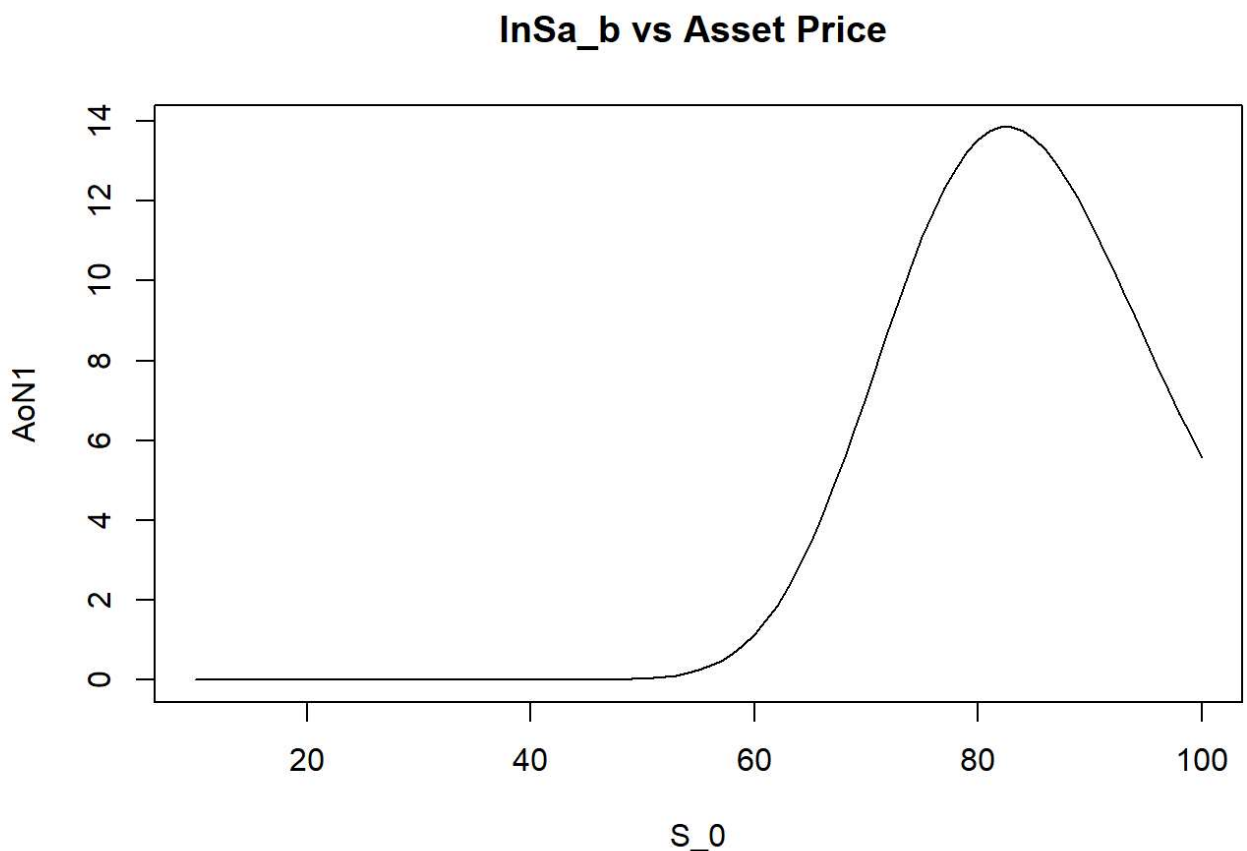


Based on Interval

InSa_b(interval asset): payoff=$S_T \cdot I_{(a \leq S_T \leq b)}$, price=$S_0(N(d_1^a) - N(d_1^b))$

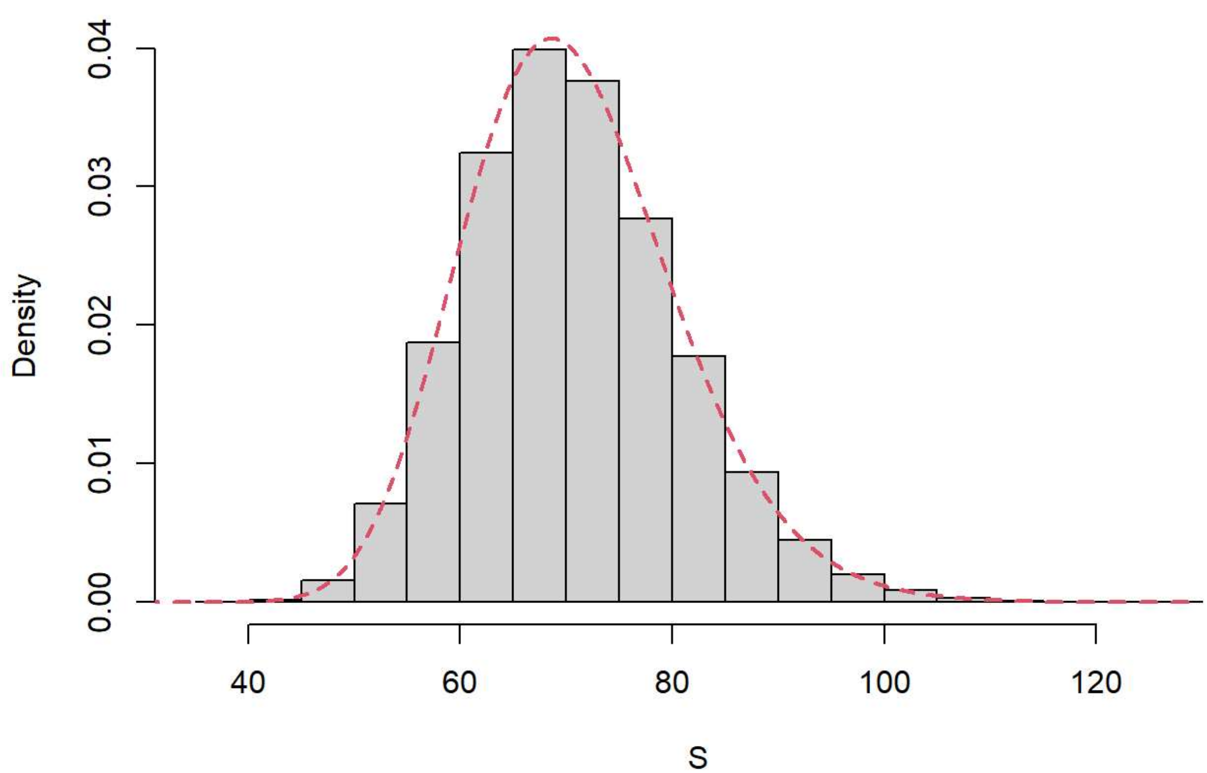
InCa_b(interval one-dollar-cash): payoff=$I_{(a \leq S_T \leq b)}$, price=$e^{-rT}(N(d_2^a) - N(d_2^b))$

```
a_b_BS_Price = function(a, b, S0, r, M, v){
  InSa_b = General_BSprice(S0,a,r,M,v)[,1]-General_BSprice(S0,b,r,M,v)[,1]
  InCa_b = General_BSprice(S0,a,r,M,v)[,2]-General_BSprice(S0,b,r,M,v)[,2]
  return(cbind(InSa_b,InCa_b))
}


prices = a_b_BS_Price(80,85,10:100,.02,.5,.2)


plot(10:100, prices[,1], type='l', xlab="S_0", ylab="AoN1",
     main="InSa_b vs Asset Price ")
```



4.European Call - Monte Carlo Simulation for Option Pricing

```
#####
# Alternatively, you can use the function EuropeanOption() from the RQuantLib library
# install.packages("RQuantLib")
# library(RQuantLib)

set.seed(1234567890)

#####
# European option pricing by simulation

### 1.INPUT - Parameters
S0=70 # asset price at time 0
v=.2 # annual volatility
r=.02 # annual risk-free rate
K=75 # strike price
M=.5 # Time to maturity (in years)
n=100000 # number of variates

### 2.Simulate Asset Price at Maturity
Z=rnorm(n) # Standard Normal variates
S=S0*exp( (r-v^2/2)*M + v*sqrt(M)*Z ) # asset price variates @ T

### 3.Histogram of Simulated Asset Prices
hist(S, main="", prob=TRUE) # asset price variate histogram
lines( seq(30,130,.02), dlnorm(seq(30,130,.02)/S0, # theoretical LogNormal density
                              (r-v^2/2)*M, v*sqrt(M) )/S0, col=2, lty=2, lwd=2)
```



- `S=S0exp((r-v^2/2)*M + v*sqrt(M)*Z)` : This simulates the asset price `S` at maturity using the geometric Brownian motion (GBM) formula: $S_T = S_0 \cdot \exp\left(\left(r - \frac{1}{2}v^2\right)T + v\sqrt{T}Z\right)$

  Where: - S_T: Stock price at time T, - S_0: Initial stock price, - r: Risk-free interest rate, - v: Volatility, - T: Time to maturity, - Z: Standard normal random variable.

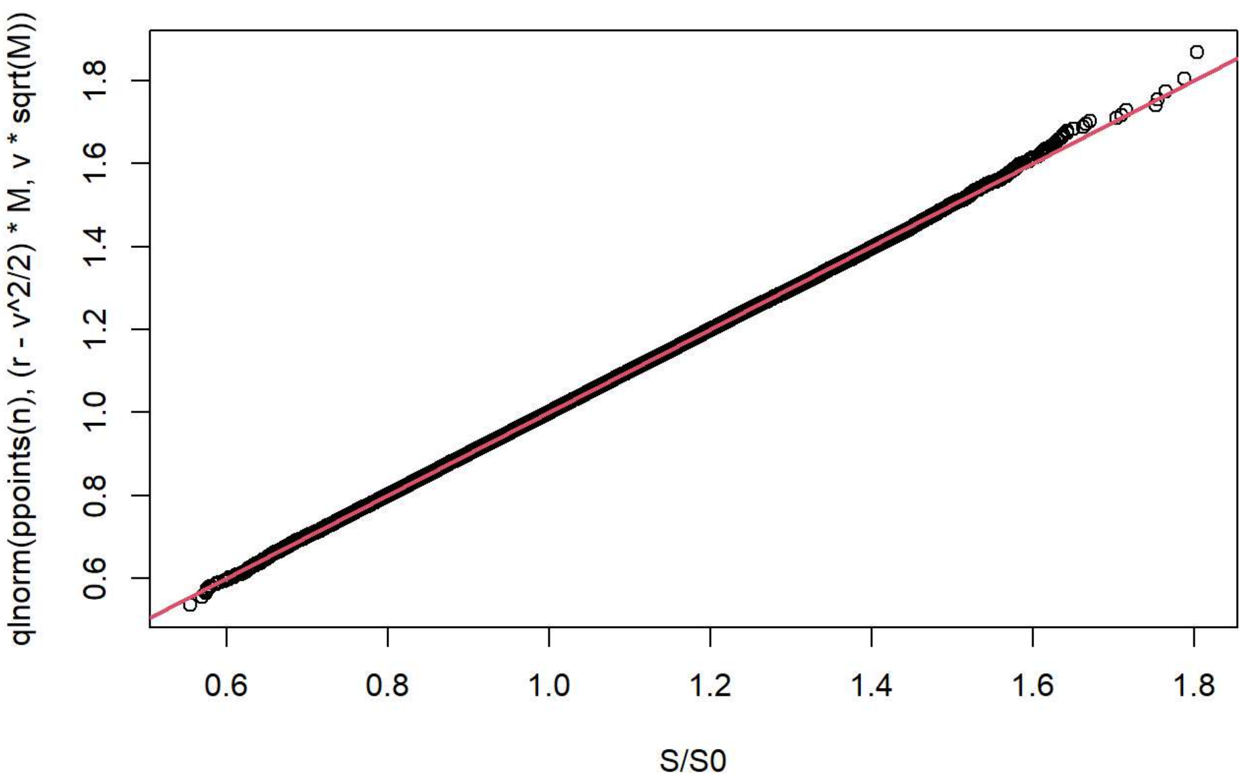
- `dlnorm(seq(30,130,.02)/S0,(r-v^2/2)*M, v*sqrt(M) )/S0` :
  - The log-return of the stock price follows a normal distribution:

    $$\log(S_T/S_0) \sim \mathcal{N}\left((r - \frac{1}{2}v^2)T, v\sqrt{T}\right)$$

    Where: - S_T: Stock price at time T, - S_0: Initial stock price, - r: Risk-free interest rate, - v: Volatility, - T: Time to maturity, - Z: Standard normal random variable.

  - `dlnorm(x, meanlog, sdlog)` : log-normal probability density function.

```
### 4.Lognormal QQ-Plot to Check Distribution Fit
qqplot( S/S0, qlnorm(ppoints(n), (r-v^2/2)*M, v*sqrt(M) ) )
abline(0,1, lwd=2, col=2)
```




```
### 5.Mean and Standard Deviation Check

mean(S)  # must be approx. S0*exp(r*M)
```

```
## [1] 70.71633
```

```
S0*exp(r*M)
```

```
## [1] 70.70351
```

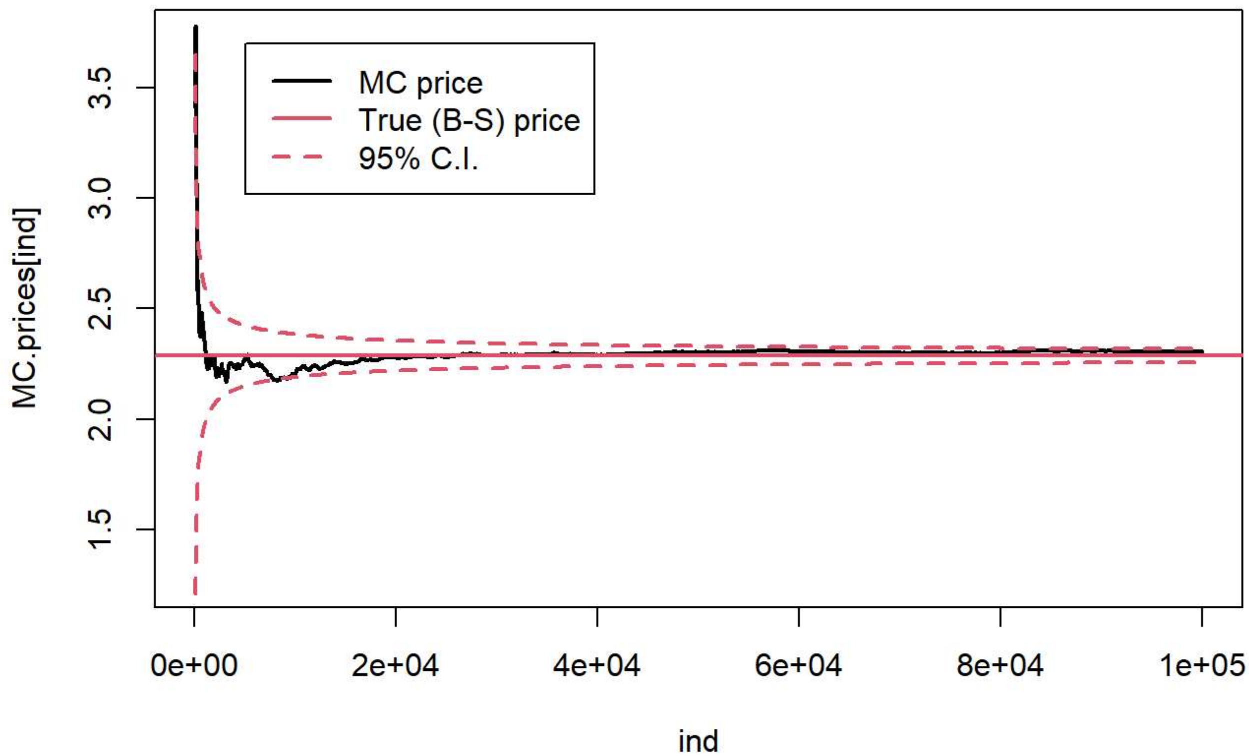
```
sd(S) # must be approx. S0*sqrt( exp(2*r*M) * (exp(v^2*M)-1) )
```

```
## [1] 10.0807
```

```
S0*sqrt( exp(2*r*M) * (exp(v^2*M)-1) )
```

```
## [1] 10.04919
```

```
### 6.1.Monte Carlo Estimate of Call Option Price
call.MCprice=mean( exp(-r*M)*pmax(0,S-K) ) # Monte-Carlo (MC) call price

### 6.2.Compare to Exact Black-Scholes Price
call.BSprice=BSprice(S0,K,r,M,v)[1] # Exact Black-Scholes price
# call.BSprice=EuropeanOption("call",S0,75,0,.02,.5,.2)$value  # using RQuantLib

### 7.MC Price Convergence Plot
MC.prices=cumsum( exp(-r*M)*pmax(0,S-K) ) / (1:n) # average of Monte Carlo estimate of the c
all option price.
S_C=sd(exp(-r*M)*pmax(0,S-K)) # Standard deviation of all discounted payoffs.

a=0.95;Za = qnorm(1-(1-a)/2) ;ind=seq(50,n,by=50)
plot( ind, MC.prices[ind], ylim=c(1.25,3.75),lwd=2, type='l') # Monte Carlo price
abline(h=call.BSprice[1], col=2, lwd=2) # True B-S price
lines(ind, call.BSprice[1]+ Za*S_C*sqrt(1/ind), lwd=2, lty=2, col=2) # Upper 95% CI
lines(ind, call.BSprice[1]- Za*S_C*sqrt(1/ind), lwd=2, lty=2, col=2) # Lower 95% CI
legend( 5000,3.7, c("MC price", "True (B-S) price", "95% C.I."), lty=c(1,1,2), col=c(1,2,2),
lwd=c(2,2,2) )
```



- `call.MCprice=mean( exp(-r*M)*pmax(0,S-K) )`:
  - The option pricing formula is given by:

$$C_0 = e^{-rT} \cdot \mathbb{E}[\max(S_T - K, 0)]$$

- `2*S_C*sqrt(1/ind)` :
  - Approximates a 95% confidence interval using the 2-sigma rule (since $Z_{0.975} \approx 1.96$, people often just use 2).

5.Exchange Option - Monte Carlo Pricing

Exchange option

Suppose $S_1(t)$ and $S_2(t)$ are the prices of two risky assets at time t, each with constant continuous dividend yield q_i. The exchange option gives the right (but not obligation) to exchange asset 2 for asset 1 at maturity T, with payoff:

$$C(T) = \max(0, S_1(T) - S_2(T))$$

Combined Volatility

If the volatilities are σ_i and ρ is their correlation coefficient:

$$\sigma = \sqrt{\sigma_1^2 + \sigma_2^2 - 2\sigma_1\sigma_2\rho}$$

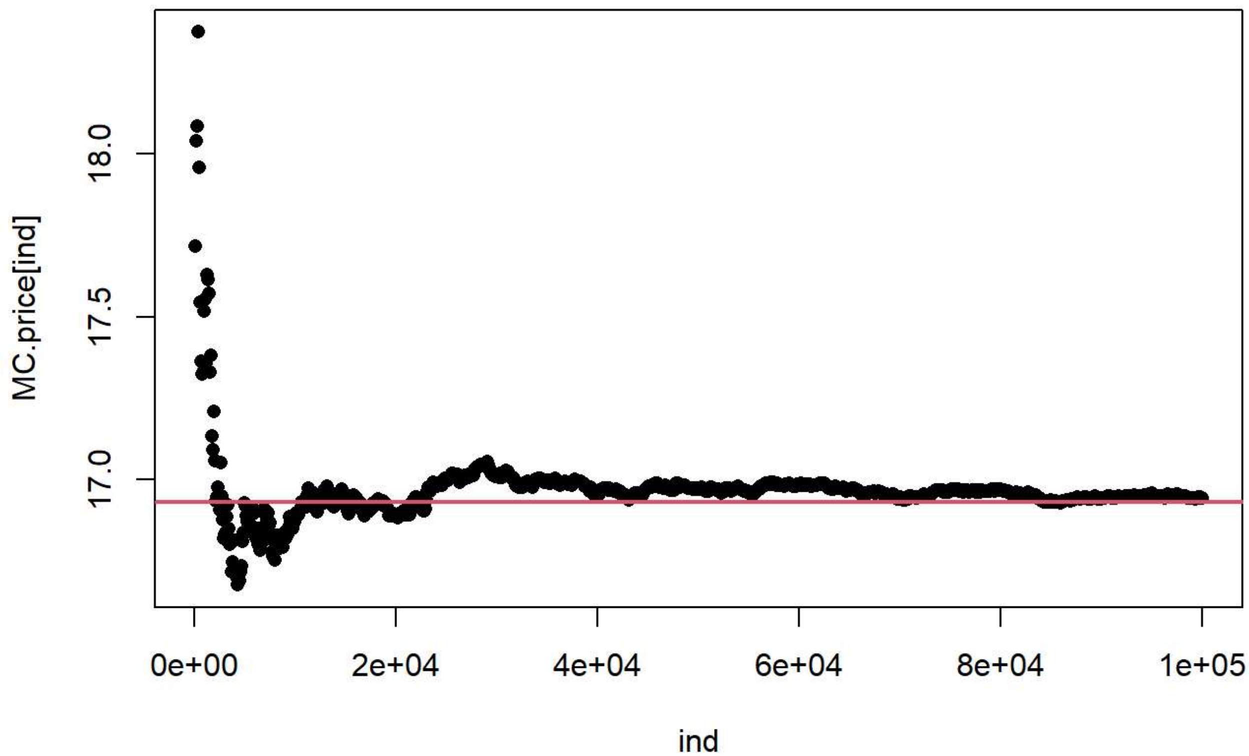
Margrabe's formula states that the fair price for the option fair price at time 0 is:

$$C(0) = e^{-q_1 T} S_1(0) N(d_1) - e^{-q_2 T} S_2(0) N(d_2)$$

Where: $N(\cdot)$ = Standard normal CDF | q_1, q_2 = Dividend yields of S_1 and S_2

$d_1 = \frac{\ln(S_1(0)/S_2(0)) + (q_2 - q_1 + \sigma^2/2)T}{\sigma\sqrt{T}}$ - $d_2 = d_1 - \sigma\sqrt{T}$

```
#####
# MC Pricing of Exchange Option

### 1.INPUT - parameters
n=100000 # number of paths (no steps needed)
TT=1 # maturity
rf=.03 # risk free rate
sigma1=0.4; sigma2=0.3 # volatilities
V=c(sigma1, sigma2)
rho=.3 # correlation
Rho=matrix( c(1,rho,rho,1),2,2 )
Sig=Rho*(V%*%t(V)) # Covariance matrix (element-wise multiply)
L=chol(Sig) # Cholesky factorization of covariance matrix
Drft=rf-V^2/2 # risk-neutral drift for log-prices
S0=c(50,35) # Initial prices of Asset 1 and Asset 2

### 2.Simulate Terminal Prices
Z=matrix(rnorm(n*2), n, 2)
ST=exp( matrix(Drft*TT,n,2,byrow=TRUE)+ Z%*%L*sqrt(TT))*matrix(S0,n,2,byrow=TRUE)
payoff=pmax(ST[,1]-ST[,2],0) # Payoff of the exchange option

MC.price=cumsum(payoff*exp(-rf*TT))/(1:n) # Discounted average
ind=seq(100,n,100)
plot( ind, MC.price[ind], pch=16 )

# Use "Margrabe's formula" (https://en.wikipedia.org/wiki/Margrabe%27s_formula)
# to find the exchange option's price based on the Black-Scholes formula
library(RQuantLib)
exact.price=EuropeanOption("call",S0[1],S0[2],0,0,TT, sqrt(sigma1^2 + sigma2^2 - 2*sigma1*si
gma2*rho ) )$value  # using RQuantLib

abline(h=exact.price, lwd=2, col=2)
```



- `EuropeanOption(type, underlying, strike, dividendYield, riskFreeRate, maturity, volatility)` :
    - You set `riskFreeRate = 0` in this formula only because: You are pricing an **exchange option using Margrabe's formula**, which inherently assumes no discounting is needed — the two risky assets grow at the same rate under risk-neutral measure, so the risk-free rate cancels out in the math.

6.Binary Option - Monte Carlo & Exact Price

Consider a Binary option that pays off $S_T 1_{\{S_T \geq K\}} = \begin{cases} S_T, & S_T \geq K \\ 0, & S_T < K \end{cases}$ at maturity T. This is a European asset-or-nothing binary call, and has a analytical price in the Black-Scholes model (i.e. for constant interest rate and an underlying asset following Geometric Brownian motion). Consider a model where the interest rate is $r = .05$ and the asset has initial price $S(0) = 100$ and volatility $\sigma = .4$. Find the price of the asset-or-nothing call with strike $K = 100$ and maturity $T = .5$ using Monte Carlo simulation with $n = 10,000$. Report your point estimate and the approximate 95% confidence interval for the price, and compare it to the exact price.

```
set.seed(1234567890)
S0=100; r=0.05; sigma=.4; T= 0.5; K = 100

# exact price

C.exact = General_BSprice(S0,K,r,T,sigma)[,1]
print( paste("Exact binary call price =", C.exact) )
```

```
## [1] "Exact binary call price = 59.0880178044312"
```

```r
set.seed(1234567890)
S0=100; r=0.05; sigma=.4; T= 0.5; K = 100

# exact price
d1 = ( log(S0/K) + ( r + sigma^2 / 2 ) * T ) / ( sigma * sqrt(T) )
C.exact = S0 * pnorm( d1 )
print( paste("Exact binary call price =", C.exact) )
```

```
## [1] "Exact binary call price = 59.0880178044312"
```

```r
# MC simulation
n=10000
Z = rnorm( n )
ST = S0 * exp( ( r - sigma^2 / 2 ) * T + sigma * sqrt(T) * Z )
payoff_3 = exp( -r * T ) * ifelse( ST>K, ST, 0 )
mean_3 = mean(payoff_3); se_3 = sd(payoff_3) / sqrt(n)

a=0.95; CI_3 = mean_3 + c(-1,1) * qnorm(1-(1-a)/2) * se_3
print( paste("MC binary call price =", mean_3) )
```

```
## [1] "MC binary call price = 58.2251962242492"
```

```r
print( paste( c("MC binary call",a*100,'% CI =', CI_3), collapse = "  " ) )
```

```
## [1] "MC binary call  95  % CI =  56.984595876404  59.4657965720943"
```

- `ifelse( ST>K, ST, 0 )` : This is equivalent to `(ST>=K)* pmax(0,ST)`

7.Rainbow Option (Multi-Assets) - Monte Carlo Pricing

A rainbow option is a derivative exposed to two or more underlying assets. 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 multivariate Geometric Brownian Motion:

$dS(t) = rS(t)dt + \sigma \circ S(t) \circ d\mathbf{W}(t)$ for $\sigma = [.1\ .2\ .3\ .4\ .5]$ and Corr $(W_i(1), W_j(1)) = .3, \forall i \neq j$

where $\circ$ is the Hadamard or element-wise matrix product.

(Note: there is no general analytical solution for rainbow options.)

```
set.seed(1234567890)
n=10000; d=5; S0=100; M=1; r=.01 # - - |

Rho= matrix(.3,5,5); 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
Z=matrix( rnorm(n*d) , n, d)
ST=S0*exp( Drft*M+ Z%*%L*sqrt(M) )


maxT=apply(ST,1,max)
avgT=apply(ST,1,mean)
payoff_4=exp(-r*M)*(maxT-avgT) # Payoff - - |

mean_4=mean(payoff_4)
se_4=sd(payoff_4)/sqrt(n)

a=0.95;CI_4=mean_4+c(-1,1)*qnorm(1-(1-a)/2)*se_4
print( paste("rainbow option price =", mean_4) )
```

```
## [1] "rainbow option price = 36.2473636425247"
```

```
print(  paste( c("rainbow option",a*100,'% CI =',CI_4), collapse = "  " ) )
```

```
## [1] "rainbow option  95  % CI =  35.7261581492228  36.7685691358267"
```