Assigment 2

Shunwen Tan, Fernando Hernandez, Shreyas Purohit, Zheng Mao

March 13, 2017

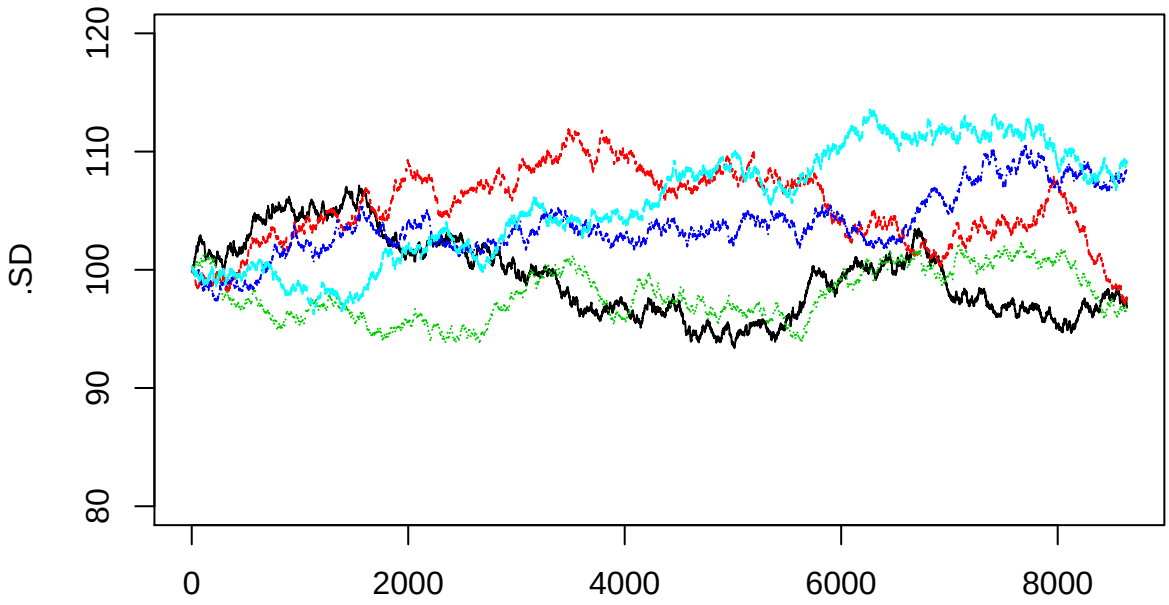
Question 1. Black-Scholes: Closed Form Solution vs. Monte-Carlo Simulation

The purpose of this exercise is to price a vanilla call option with Monte-Carlo simulation and compare the result with the closed-form formula. The simulation will also provide a confidence interval for the option price. We will understand how thelength of this confidence interval changes as we increase the number of simulated paths. The price of a stock today is S0 = 100. Consider a European call option with maturity 3 months (T = 1=4) and strike price K = 100. We make the following parametric assumptions: r = 0.05, σ = 0.2, and δ = 0.

For this exercise, reset the random number generator before each simulation run by randn('seed',0).

a. Simulate and plot 5 paths for the stock price under the risk-neutral measure. Use 5 minutes time-increments (there are 8 x12 = 96 increments each day and 90 days until maturity).

```
paths = matrix(0, n, m)
paths = data.table(paths)
paths[1, ] = s0
v = r - 0.5 * sigma^2
set.seed(0)
for (i in 2:n){
  sim = rnorm(m)
  paths[i, ] = paths[i - 1, ] * exp(v * h + sigma * sqrt(h) * sim)
}
paths[, matplot(.SD, type = "l", ylim = c(80,120))]
```




```
## NULL
```

b. Find the Black-Scholes call option price.

```
black.scholes = function(k, s0, r, T, sigma) {
  values = c(2)

  d1 = (log(s0/k)+(r+sigma^2/2)*T)/(sigma*sqrt(T))
  d2 = d1 - sigma * sqrt(T)

  values[1] = s0*pnorm(d1) - k*exp(-r*T)*pnorm(d2)
  values[2] = k*exp(-r*T) * pnorm(-d2) - s0*pnorm(-d1)

  values
}
black.scholes(k, s0, r, t, sigma)[1]
```

```
## [1] 4.614997
```

c. Find the option price and its 95% confidence interval by Monte-Carlo simulation. You do not have to simulate the entire paths here|since the is European, you will have to simulate only the final price ST . Do this for 100; 1,000; 1,000,000; and 100,000,000 simulations. Discuss how the length of the confidence interval changes with the number of simulations. Compare the Monte-Carlo price with the Black-Scholes price.

```
set.seed(0)
sim.bs = function(m, u, d, n, p, s0,k){
  f.path = c()
  f.path = s0 * exp((r - delta - .5 * sigma^2) * t + sigma * sqrt(t) * rnorm(m)) - k
  price = exp(-r*t)*mean(pmax(f.path, 0))
  sd = sd(f.path)
  conf.int = c(price - 1.96 * sd / sqrt(m), price +  1.96 * sd / sqrt(m))
  result = c(price, conf.int)
  names(result) = c("Price", "Lower", "Upper")
  result
}
repetitions = c(100, 1000, 1000000, 100000000)
sim2 = sapply(repetitions, sim.bs, u = u, d = d, n = n, p = p, s0 = s0, k = k)
colnames(sim2) = c("100", "1,000", "1,000,000", "100,000,000")
kable(sim2)
```

	100	1,000	1,000,000	100,000,000
Price	4.197527	4.532937	4.612806	4.614317
Lower	2.423057	3.893891	4.592913	4.612328
Upper	5.971998	5.171983	4.632699	4.616307

The value tends to the Black-Scholes price of 4.614, and with 100 million simulations its is correct up to three decimal points.

Question 2 Down-And-Out Put Option Price by Monte-Carlo Simulation

The purpose of this exercise is to price a down-and-out put option with Monte-Carlo simulation and to understand how the volatility of the underlying aspects the proportion of crossed paths and consequently the price of the option. The price of a stock today is S0 = 100. Consider a down-and-out put option (an option that ceases to exist when the price of the underlying security hits a specific barrier price level) with maturity 3 months (T = 1=4), strike price K = 95, and a barrier level of Sb = 75. We make the following parametric assumptions: r = 0.05, $\sigma = 0.2$, and $\delta = 0$. For this exercise, reset the random number generator before each simulation run by randn('seed',0).

```
set.seed(0)
mc.downandout = function(n, m, r, delta, h, sigma, s0, sb){
  paths = matrix(0, n, m)
  paths[1, ] = s0
  v = r - 0.5 * sigma^2
  for (i in 2:n){
    paths[i, ] = paths[i - 1, ] * exp(v * h + sigma * sqrt(h) * rnorm(m))
```

```
    }
  # What is the number of paths that cease to exist.
  break.barrier = function (x){if(any(x<sb)){1} else {0}}
  paths.out = data.table(paths)[,lapply(.SD, break.barrier),][,sum(.SD)]

  # Fill in with 95 the path that crossed the
  paths.updated = paths
  for (j in 1:m){
    if (any(paths.updated[, j] < sb)) {
      paths.updated[, j] = k
    }
  }

  payoffs = sapply(k - paths.updated[n, ], max, 0)
  price = exp(-r*t)*mean(payoffs)
  sd = sd(payoffs)
  conf.int = c(price - sd*1.96/sqrt(m), price + sd*1.96/sqrt(m))
  result = c(price, conf.int, paths.out)
  names(result) = c("Price", "Lower", "Upper", "Paths Out")
  result
}
```

a. Simulate (under the risk-neutral measure) 1,000 paths for the stock price at 5 minute intervals (there are 8 x 12 = 96 increments each day and 90 days until maturity). You have to simulate the entire paths here, since the option is now path-dependent. How many of the paths are "out"," i.e., when the underlying price crossed the barrier Sb? What is the price of the put option and its 95% confidence interval? How does this price compare with the Black-Scholes put option price? Explain the diference.

```
set.seed(0)
r.2 = mc.downandout(n, m, r, delta, h, sigma, s0, sb)
kable(r.2)
```

Price	1.542262
Lower	1.334935
Upper	1.749589
Paths Out	2.000000

```
bs.2 = black.scholes(k, s0, r, t, sigma)[2]
kable(bs.2, col.names = "BS Price")
```

BS Price

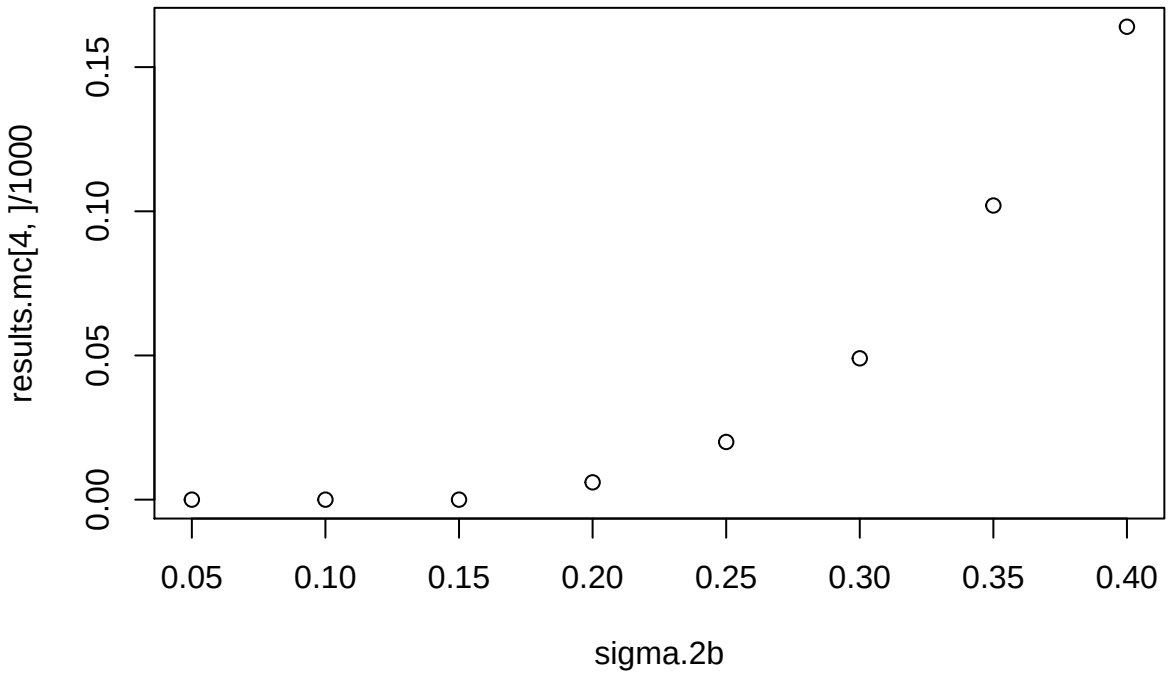
1.534261

We can se that the price is lower than the Black Scholes price for a put option.

b. Assume the following levels of volatility for the underlying: 5%, 10%, 15%, 20%, 25%, 30%, 35%, 40%. Simulate 1,000 paths for the stock price for each level of volatility. Plot a graph that has the volatility on the X axis and the proportion of "out" paths on the Y axis. What do you observe? Explain/interpret your findings.

```
set.seed(0)
sigma.2b = c(.05, .1, .15, .2, .25, .30, .35, .4)
```

```
results.mc = sapply(sigma.2b, mc.downandout, n = n, m = m, r = r, delta = delta, h = h, s0 = s0, sb = sb
plot(sigma.2b, results.mc[4,]/1000)
```



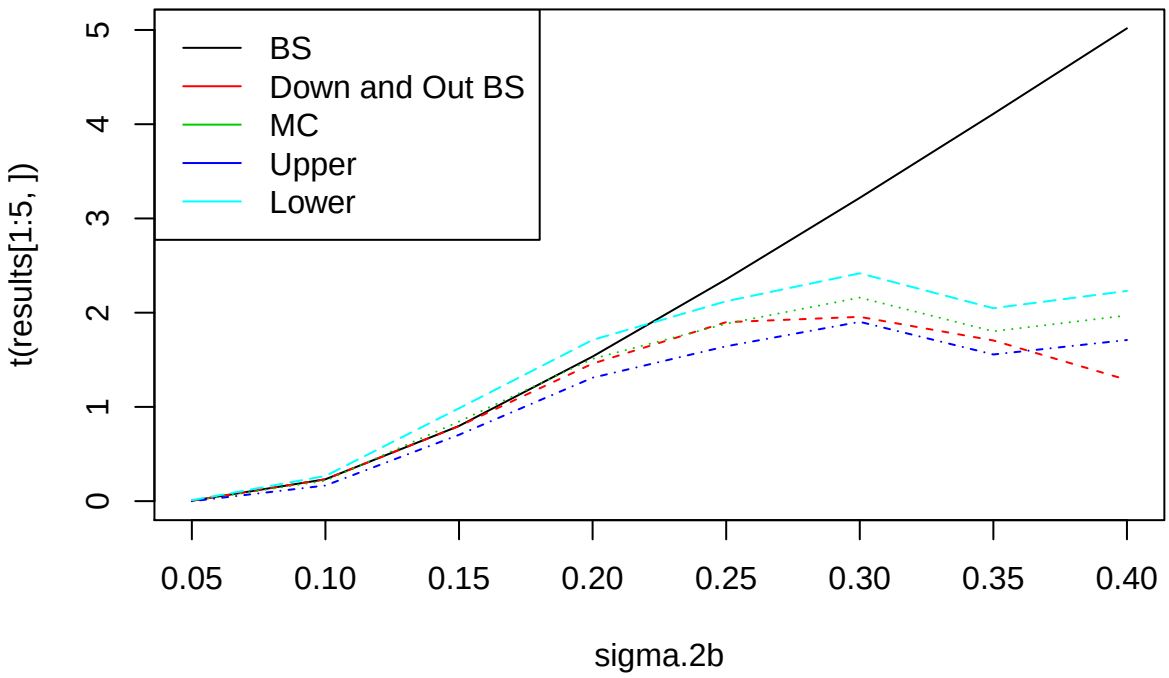
As the volatility grows, the number of paths that touch the barrier grows. This type of option is better when you don't expect volatility in the future.

c. For the same levels of the volatility, plot a graph that has the same X axis as above and the price of the down-and-out put on the Y axis, together with its confidence interval. Also, plot on the same graph the Black-Scholes price for each level of volatility. What do you observe? Explain/interpret your findings.

```
results.bs = sapply(sigma.2b, black.scholes, k = k, s0 = s0, r = r, T = t)[2,]
results.bs.do = sapply(sigma.2b, black.scholes.downandout, k = k, s0 = s0, r = r, t = t, sb =sb)
results = rbind(results.bs, results.bs.do, results.mc)
kable(results)
```

results.bs	0.0041293	0.2320021	0.7977289	1.534261	2.354056	3.218755	4.109813	5.017302
results.bs.do	0.0041293	0.2320021	0.7959787	1.459446	1.899291	1.955709	1.704934	1.287324
Price	0.0043487	0.2164974	0.8445851	1.510230	1.882268	2.160311	1.802019	1.970115
Lower	-0.0010669	0.1659342	0.7037874	1.309883	1.643278	1.902828	1.556560	1.709740
Upper	0.0097643	0.2670606	0.9853828	1.710577	2.121258	2.417794	2.047478	2.230490
Paths Out	0.0000000	0.0000000	0.0000000	6.000000	20.000000	49.000000	102.000000	164.000000

```
colnames(results) = sigma.2b
matplot(sigma.2b, t(results[1:5,]), type = "l", col = c(1,2,3,4,5))
legend("topleft", legend = c("BS", "Down and Out BS", "MC","Upper","Lower"), lty = c(1,1,1), col = c(1,
```



We see that the price of the option separates from the Black-Sholes price as the volatility grows. This makes sense since probability of hitting the barrier grows with the volatility.

3.Pricing Exotic Options in Complicated Market Structures

a.Discretize the SDEs using the Euler discretization scheme.Using $1{,}000$ **simulations and a step size of** $1/250$ **approximate the distribution (draw the histogram) of (**$r_T|r_0 = 0.05$**), for** $T = 1$**.**

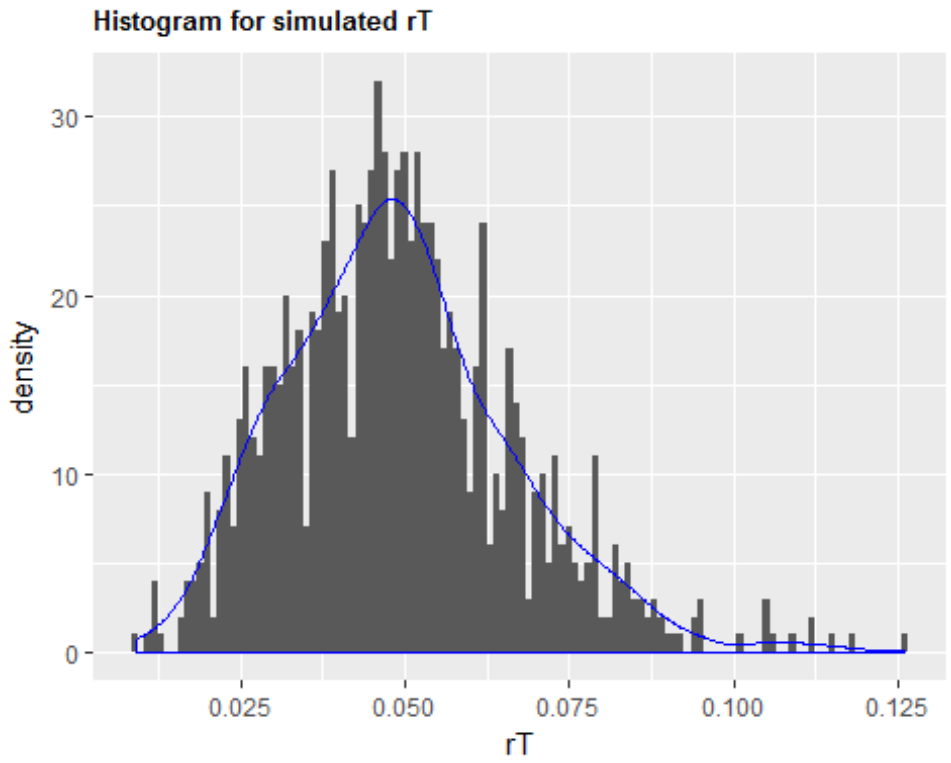
$$S_{1t+dt} = S_{1t} + r_t S_{1t} dt + \sigma_{11}\sqrt{S_{1t}}\sqrt{dt}Z_1 + \sigma_{12}S_{1t}\sqrt{dt}Z_2$$

$$S_{2t+dt} = S_{2t} + r_t S_{2t} dt + \sigma_{21}(S_{1t} - S_{2t})\sqrt{dt}Z_1$$

$$r_{t+dt} = r_t + \alpha(\beta - r_t)dt + \delta\sqrt{r_t}\sqrt{dt}Z_1$$

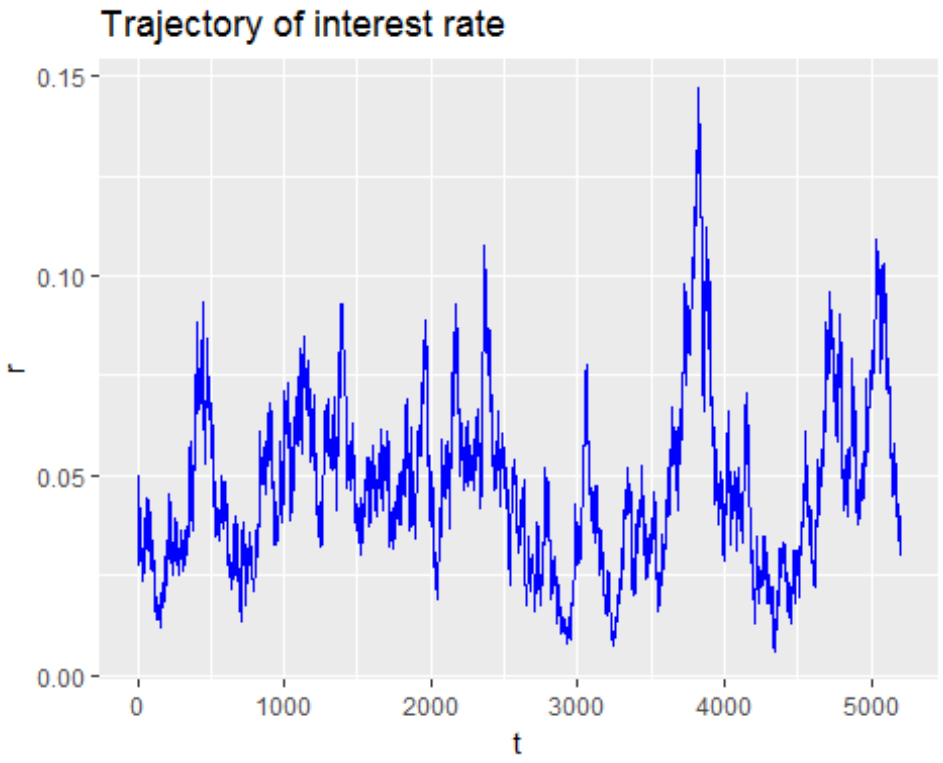
```
#parameters
r0<-0.05
alpha<-0.6
beta<-0.05
sigma11<-0.1
sigma12<-0.2
sigma21<-0.3
S10<-10
S20<-10
delta<-0.1
dt=1/250
NSteps<-1/dt
NS<-1000

#simulate paths of r
rPaths = matrix(nrow=NS, ncol=NSteps+1)
rPaths[,1] = r0
for (i in 1:NS)
   for (j in 1:NSteps)
      rPaths[i,j+1]=rPaths[i,j]+alpha*(beta-rPaths[i,j])*dt+
                  delta*sqrt(rPaths[i,j])*sqrt(dt)*rnorm(1,0,1)

# rPaths_test<-matrix(nrow=NS,ncol=1)
#for (i in 1:NS)
 #     rPaths_test[i]=r0+alpha*(beta-r0)+
 #                   delta*sqrt(r0)*rnorm(1,0,1)
# rT_test<-data.frame(rT=rPaths_test)
# ggplot(rT_test, aes(x=rT))+ geom_histogram(aes(y =..density..),binwidth =
0.001)

#Plot the histogram of rT
library(ggplot2)
rT<-data.frame(rT=rPaths[,NSteps+1])
 ggplot(rT, aes(x=rT))+ geom_histogram(aes(y =..density..),binwidth = 0.001)+
  geom_density(col="blue")  +
  labs(title="Histogram for simulated rT") +
```

```
  labs(x="rT", y="density")+
  theme(plot.title = element_text(size=10, face="bold"))
```



b.Simulate one trajectory of the interest rate from $t = 0$ to $T = 100$, with a step size of $1/52$. Draw a plot of the trajectory.

```
#parameters
T=100
dt1<-1/52
NSteps1<-T/dt1

#Simulate trajectory of the interest rate
rPaths1 = matrix(nrow=1, ncol=NSteps1+1)
rPaths1[1] = r0
   for (j in 1:NSteps1)
      rPaths1[j+1]=rPaths1[j]+alpha*(beta-rPaths1[j])*dt1+
                   delta*sqrt(rPaths1[j])*sqrt(dt1)*rnorm(1,0,1)

# plot the trajectory
r1<-data.frame(t=c(1:(NSteps1+1)),r=t(rPaths1))
ggplot(r1,aes( x=t,y=r)) + geom_line(colour="blue")+
   labs(title="Trajectory of interest rate")
```



c.Using the Euler discretization and the initial parameters, price a Call option on asset 1 with strike price equal to $K = 10$ and maturity $T = 0.5$ using Monte-Carlo simulation. Use $10{,}000$ simulations and a time step of $1/250$.

```
#parameters
K.c <- 10
T.c <- 0.5
dt.c<-1/250
NS.c<-10000
NSteps.c<-T.c/dt.c

#Simulate paths of asset 1 and interest rate
SPaths.c = matrix(nrow=NS.c, ncol=NSteps.c+1)
SPaths.c[,1] = S10

rPaths.c = matrix(nrow=NS.c, ncol=NSteps.c+1)
rPaths.c[,1] = r0

for (i in 1:NS.c)
   for (j in 1:NSteps.c)
    {
     z1<-rnorm(1,0,1)
     z2<-rnorm(1,0,1)
     rPaths.c[i,j+1]=rPaths.c[i,j]+alpha*(beta-rPaths.c[i,j])*dt.c+
```

```
                  delta*sqrt(rPaths.c[i,j])*sqrt(dt.c)*z1

      SPaths.c[i,j+1]=SPaths.c[i,j]+rPaths.c[i,j]*SPaths.c[i,j]*dt.c+
                      sigma11*sqrt(SPaths.c[i,j]*dt.c)*z1+
                      sigma12*SPaths.c[i,j]*sqrt(dt.c)*z2
   }

#calc payoff of call and price of call
Payoff.c = exp(-rowSums(rPaths.c*dt.c))*pmax(SPaths.c[,NSteps.c+1]-K.c,0)
Call.c = mean(Payoff.c)
cat("Price of Call option on asset 1 is:",Call.c)

## Price of Call option on asset 1 is: 0.6968621
```

d.Using Monte-Carlo simulation price an option with payoff:

$$f(T) = max[max(max_{0\le t\le T} S_{1t}, max_{0\le t\le T} S_{2t}) - K, 0]$$

maturity $T = 0.5$ **and** $K = 10$**. Again use** $10{,}000$ **simulations.**

```
#parameters
K.d <- 10
T.d <- 0.5
dt.d<-1/250
NS.d<-10000
NSteps.d<-T.d/dt.d

#Simulate paths of interest rate, asset 1 and asset 2
SPaths.1 = matrix(nrow=NS.d, ncol=NSteps.d+1)
SPaths.1[,1] = S10
SPaths.2 = matrix(nrow=NS.d, ncol=NSteps.d+1)
SPaths.2[,1] = S20

rPaths.d = matrix(nrow=NS.d, ncol=NSteps.d+1)
rPaths.d[,1] = r0

for (i in 1:NS.d)
   for (j in 1:NSteps.d)
    {
     z1<-rnorm(1,0,1)
     z2<-rnorm(1,0,1)
     rPaths.d[i,j+1]=rPaths.d[i,j]+alpha*(beta-rPaths.d[i,j])*dt.d+
                     delta*sqrt(rPaths.d[i,j])*sqrt(dt.d)*z1

      SPaths.1[i,j+1]=SPaths.1[i,j]+rPaths.d[i,j]*SPaths.1[i,j]*dt.d+
                      sigma11*sqrt(SPaths.1[i,j]*dt.d)*z1+
                      sigma12*SPaths.1[i,j]*sqrt(dt.d)*z2

      SPaths.2[i,j+1]=SPaths.2[i,j]+rPaths.d[i,j]*SPaths.2[i,j]*dt.d+
                       sigma21*(SPaths.1[i,j]-SPaths.2[i,j])*sqrt(dt.d)*z1
```

```
   }

#calc payoff and price of the derivative
SPaths.1_max<-apply(SPaths.1,1,max)
SPaths.2_max<-apply(SPaths.2,1,max)
Payoff.d = exp(-rowSums(rPaths.d*dt.d))*pmax(pmax(SPaths.1_max,SPaths.2_max)-
K.d,0)
Call.d = mean(Payoff.d)
cat("Price of option with above payoff is:",Call.d)

## Price of option with above payoff is: 1.288713
```

4. Hedging, Large Price Movements, and Transaction Costs

a. Simulate a price path for 30 days at five minutes intervals.

```
library(ggplot2)
library(OptionPricing)
library(fOptions)
```

```
## Loading required package: timeDate
```

```
## Loading required package: timeSeries
```

```
## Loading required package: fBasics
```

```
##
```

```
## Rmetrics Package fBasics
```

```
## Analysing Markets and calculating Basic Statistics
```

```
## Copyright (C) 2005-2014 Rmetrics Association Zurich
```

```
## Educational Software for Financial Engineering and Computational Science
```

```
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
```

```
## https://www.rmetrics.org --- Mail to: info@rmetrics.org
```

```
##
```

```
## Rmetrics Package fOptions
```

```
## Pricing and Evaluating Basic Options
```

```
## Copyright (C) 2005-2014 Rmetrics Association Zurich
```

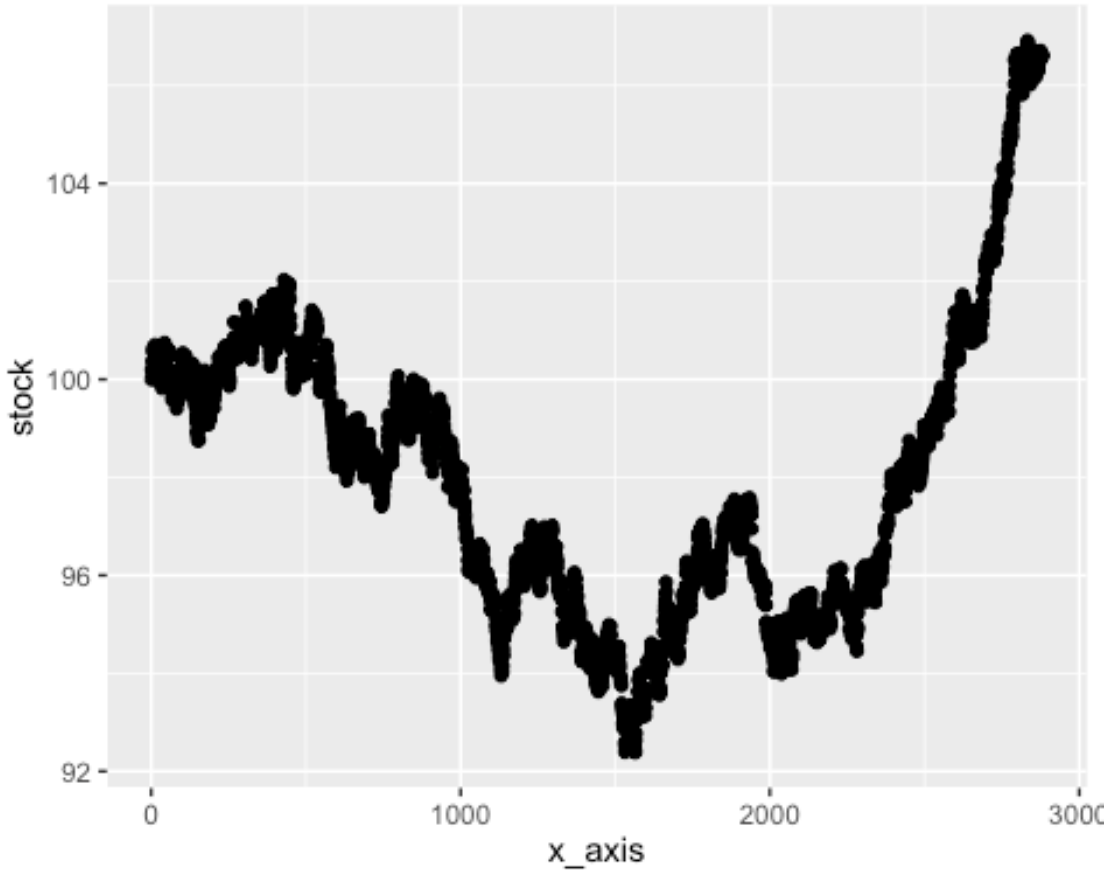
```
## Educational Software for Financial Engineering and Computational Science
```

```
## Rmetrics is free software and comes with ABSOLUTELY NO WARRANTY.
```

```
## https://www.rmetrics.org --- Mail to: info@rmetrics.org
```

```
set.seed(0)
r = 0.05
sigma = 0.3
S0 = 100
u = 0.2
K = 100
Period = 30/365
N = 30*96
h = Period/(96*30)
```

```
stock_path = function(){
  stock_price = c()
  stock_price[1] = S0
  for(j in 1:(N)){
    #Simulating N stock path
    dw=rnorm(1,0 ,1)
    ds = stock_price[j]*u*h + sigma*stock_price[j]*h^0.5*dw
    stock_price[j+1] = stock_price[j]+ds
  }
  return(stock_price)
}
stock=stock_path()
x_axis = seq(length(stock))
qplot(x_axis,stock,geom='point')
```



b. For every simulated price, compute the Black-Scholes price of the option.

The Black – Scholes price of the option has been computed for every simulated price.

```
T = 60/365
call_option = c()
delta=c()
```

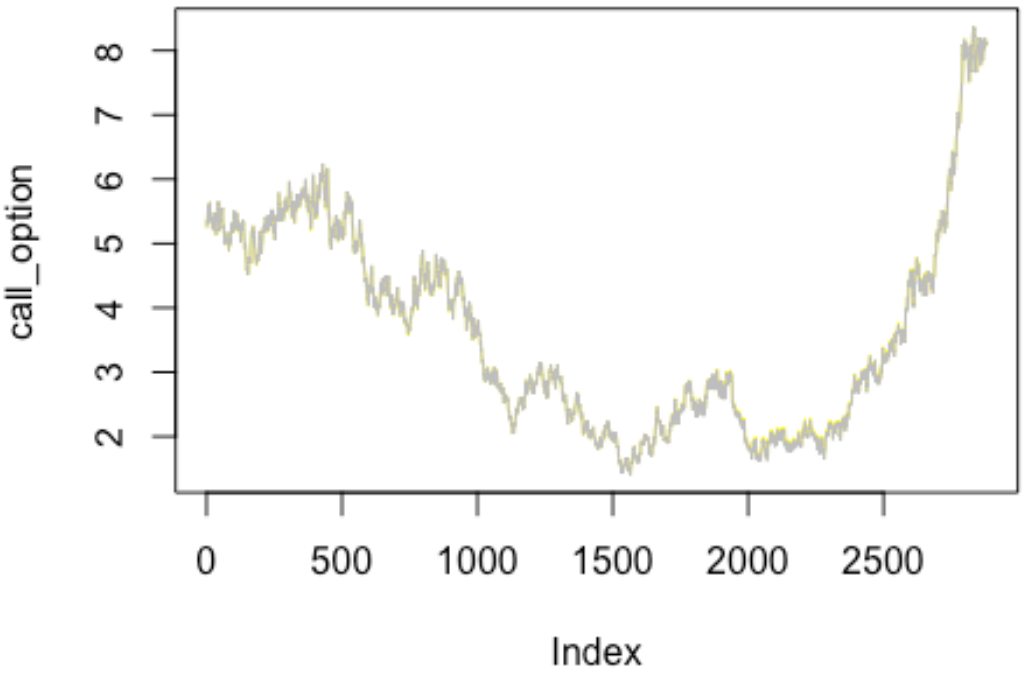
```
for(i in 1:length(stock)){
  bs_b = BS_EC((T-(i-1)*h),K,r,sigma,stock[i])
  call_option[i]=bs_b[1]
  delta[i] = bs_b[2]
}
```

c. At time 0, the value of the hedging portfolio is exactly the value of the portfolio replicating the option, i.e., $C_0 = \_S_0 + B_0$. Then, at each new data point, perform the following changes in the hedging portfolio:

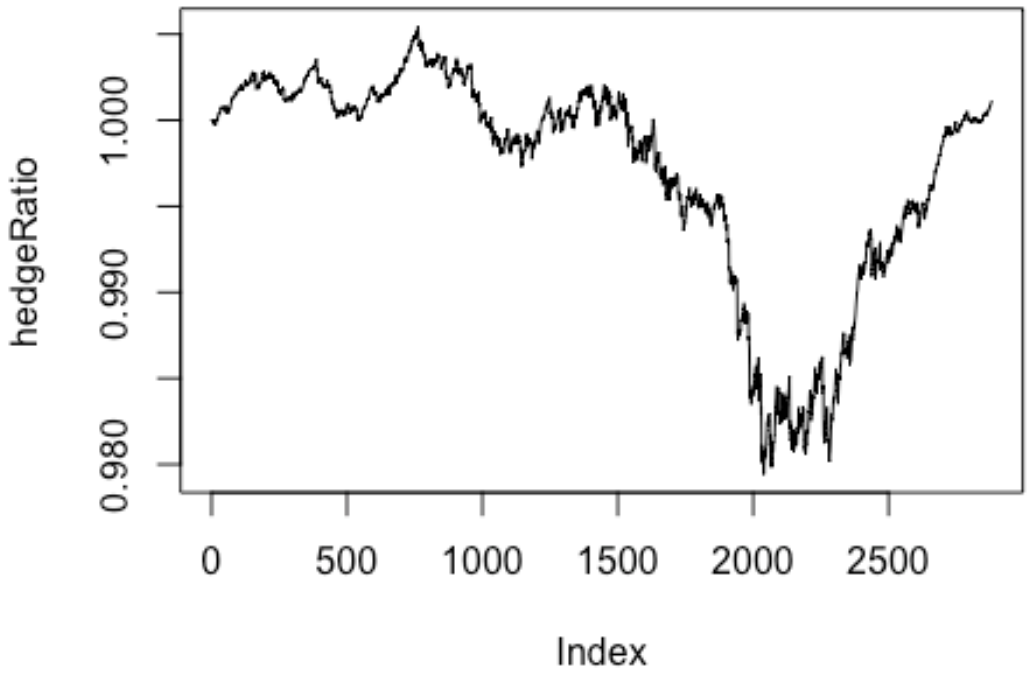
$$V_t = \Delta_{t-dt} S_t + B_{t-dt} e^{r \times dt}$$
$$B_t = V_t - \Delta_t S_t$$

```
bond_price = c()
portfolio_value = c()
portfolio_value[1] = call_option[1]
bond_price[1] = portfolio_value[1] - delta[1]*stock[1]

for(i in 1:(N)){
  portfolio_value[i+1] = delta[i]*stock[i+1] + bond_price[i]* exp(r*h)
  bond_price[i+1] = portfolio_value[i+1] - delta[i+1]*stock[i+1]
}

#Graph
plot(stock,type="l")
```


```
plot(call_option,type="l",col="yellow")
lines(portfolio_value,col="grey")
```




```
#hedgeRatio
hedgeRatio = portfolio_value/call_option
plot(hedgeRatio,type = "l")
```

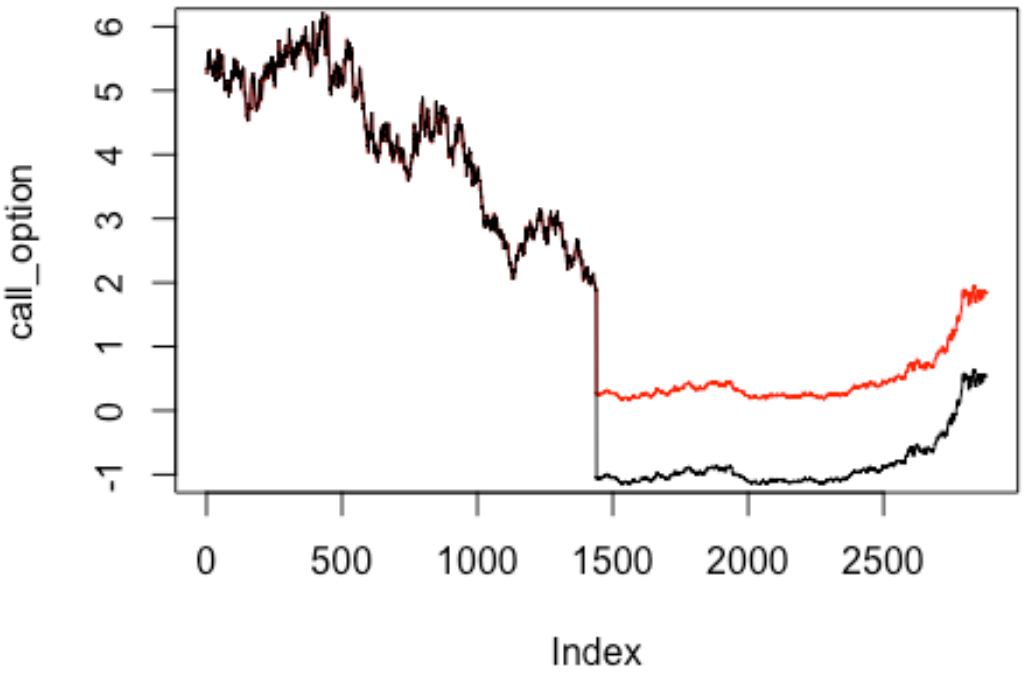


d. Assume now that exactly in the middle of the simulation there is an unexpected downward jump of 10% in the value of the asset. Plot the same graphs as before. What do you observe?

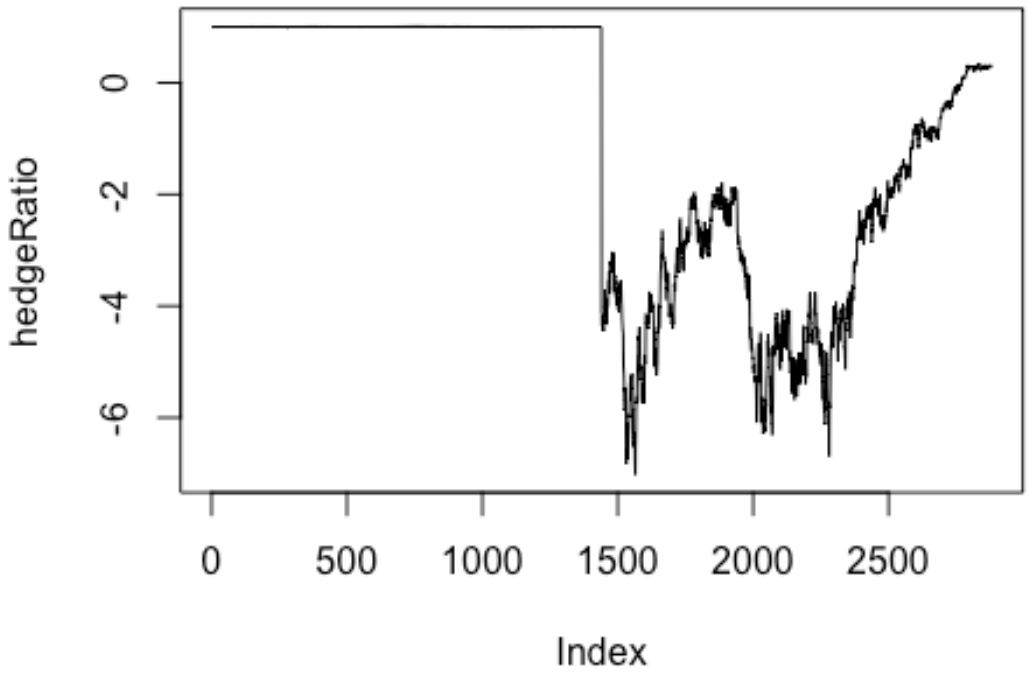
```
set.seed(0)
stock_path = function(){
  stock_price = c()
  stock_price[1] = S0
    for(j in 1:(N)){
      if(j == N/2){
        stock_price[j] = 0.9*stock_price[j]
      }
    #Simulating N stock paths
      dw=rnorm(1,0 ,1)
      ds = stock_price[j]*u*h + sigma*stock_price[j]*h^0.5*dw
      stock_price[j+1] = stock_price[j]+ds
    }
  return(stock_price)
}
```

```
stock=stock_path()

T = 60/365
call_option = c()
delta=c()
for(i in 1:length(stock)){
  bs_d = BS_EC((T-(i-1)*h),K,r,sigma,stock[i])
  call_option[i]=bs_d[1]
  delta[i] = bs_d[2]
}


bond_price = c()
portfolio_value = c()
portfolio_value[1] = call_option[1]
bond_price[1] = portfolio_value[1] - delta[1]*stock[1]
for(i in 1:(N)){
  portfolio_value[i+1] = delta[i]*stock[i+1] + bond_price[i]* exp(r*h)
  bond_price[i+1] = portfolio_value[i+1] - delta[i+1]*stock[i+1]
}


#Graph
plot(stock,type="l")
```


```
plot(call_option,type="l",col="red",ylim=c(-1,6))
lines(portfolio_value)
```




```
hedgeRatio = portfolio_value/call_option
plot(hedgeRatio,type = "l")
```



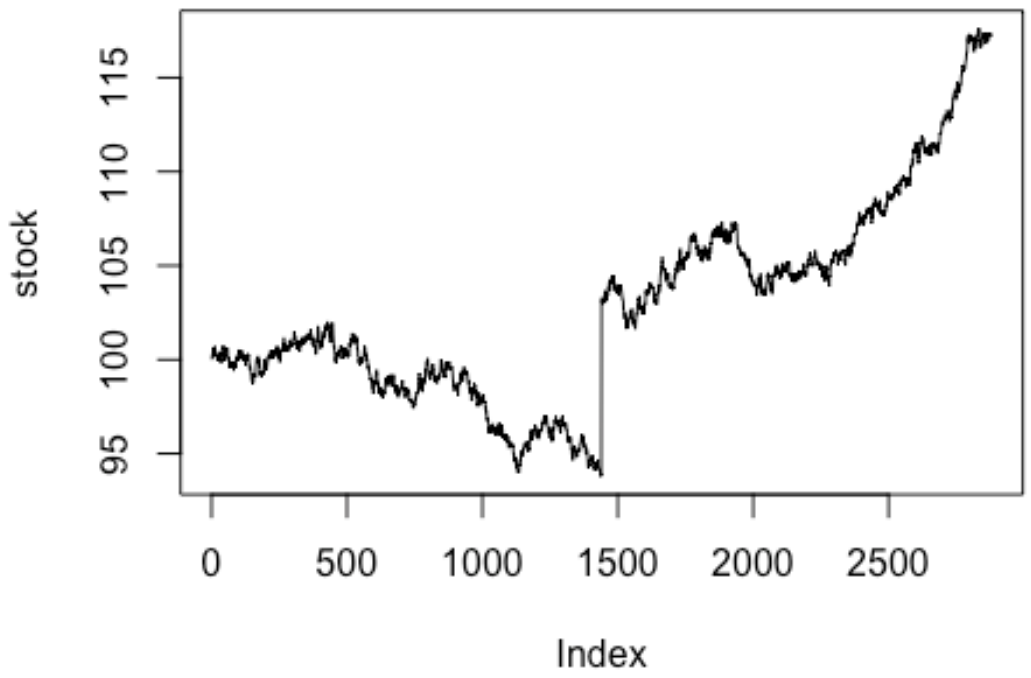
Observations:

From the graphs, we observe that the replicating portfolio and the call price deviate after the jump. Thus we can infer that delta hedging is effective only for small changes in prices and its efficiency somewhat reduces with large changes in prices.

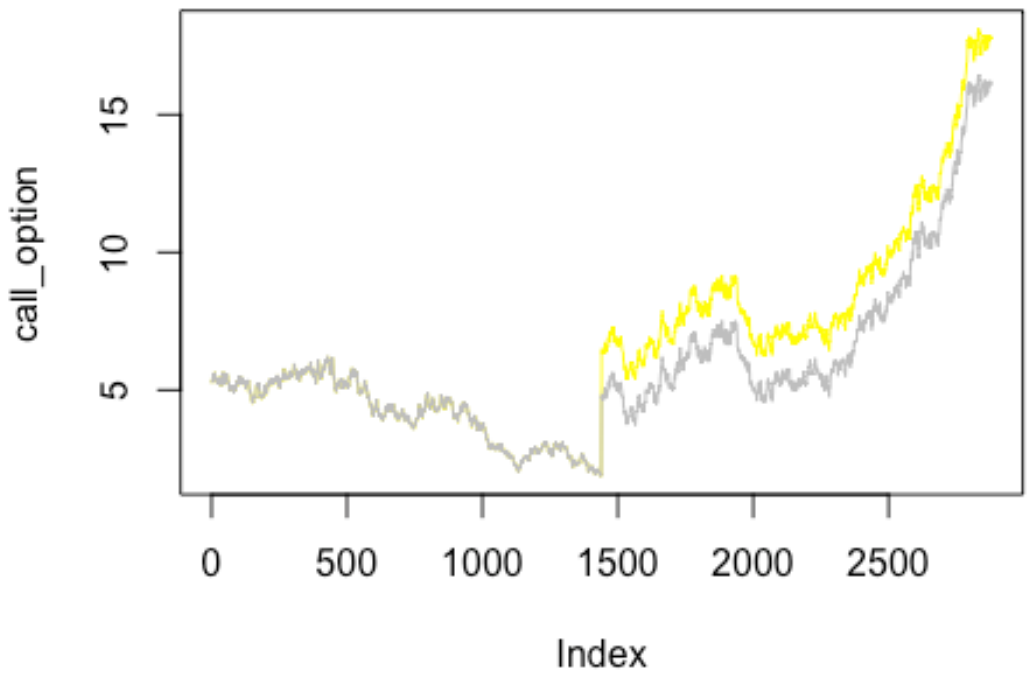
e. Perform the same exercise but assume an upward jump of 10%. Interpret your findings.

```
set.seed(0)
stock_path = function(){
  stock_price = c()
  stock_price[1] = S0
    for(j in 1:(N)){
      if(j == N/2){
        stock_price[j] = 1.1*stock_price[j]
      }
    #Simulating N stock paths
      dw=rnorm(1,0 ,1)
      ds = stock_price[j]*u*h + sigma*stock_price[j]*h^0.5*dw
      stock_price[j+1] = stock_price[j]+ds
    }
```

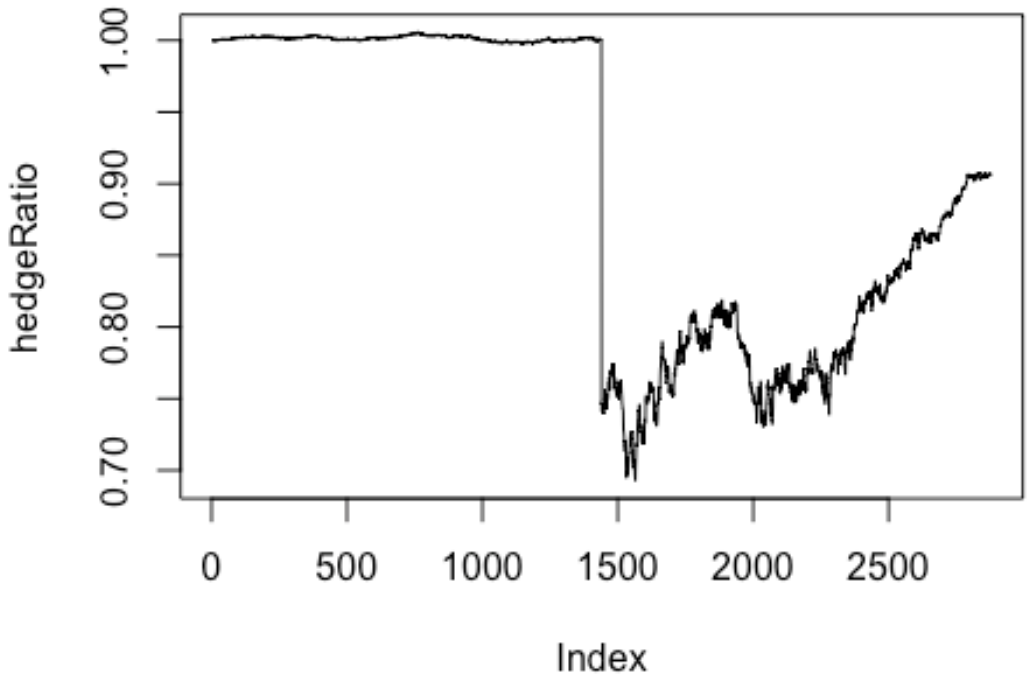
```
  return(stock_price)
}
stock=stock_path()

library(OptionPricing)
T = 60/365
call_option = c()
delta=c()
for(i in 1:length(stock)){
  bs_e = BS_EC((T-(i-1)*h),K,r,sigma,stock[i])
  call_option[i]=bs_e[1]
  delta[i] = bs_e[2]
}


bond_price = c()
portfolio_value = c()
portfolio_value[1] = call_option[1]
bond_price[1] = portfolio_value[1] - delta[1]*stock[1]
for(i in 1:(N)){
  portfolio_value[i+1] = delta[i]*stock[i+1] + bond_price[i]* exp(r*h)
  bond_price[i+1] = portfolio_value[i+1] - delta[i+1]*stock[i+1]
}

#Graph
plot(stock,type="l")
```




```
plot(call_option,type="l",col="yellow")
lines(portfolio_value,col="grey")
```




```
hedgeRatio = portfolio_value/call_option
plot(hedgeRatio,type = "l")
```



Observations:

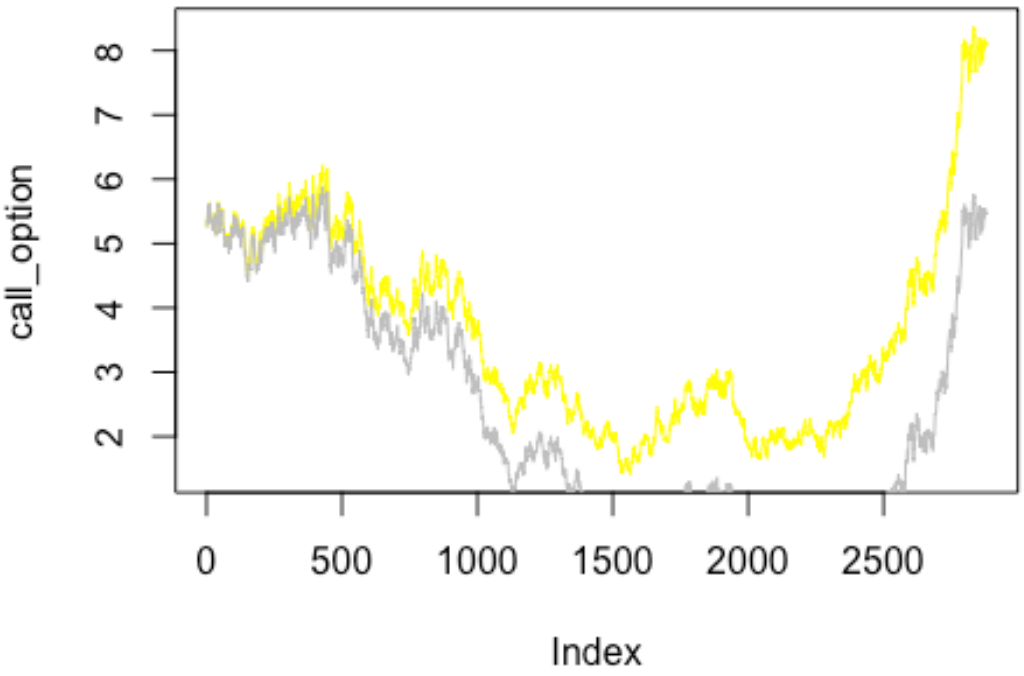
Providing further support to what was observed earlier, we can observe that delta hedges are ineffective for large changes in prices (10% in this case) and is most effective only for small changes in prices.

f. Get back to point (c) and assume that there are transaction fees of 20 basis points at each period. Plot the same graphs as requested at point (c). What do you observe?

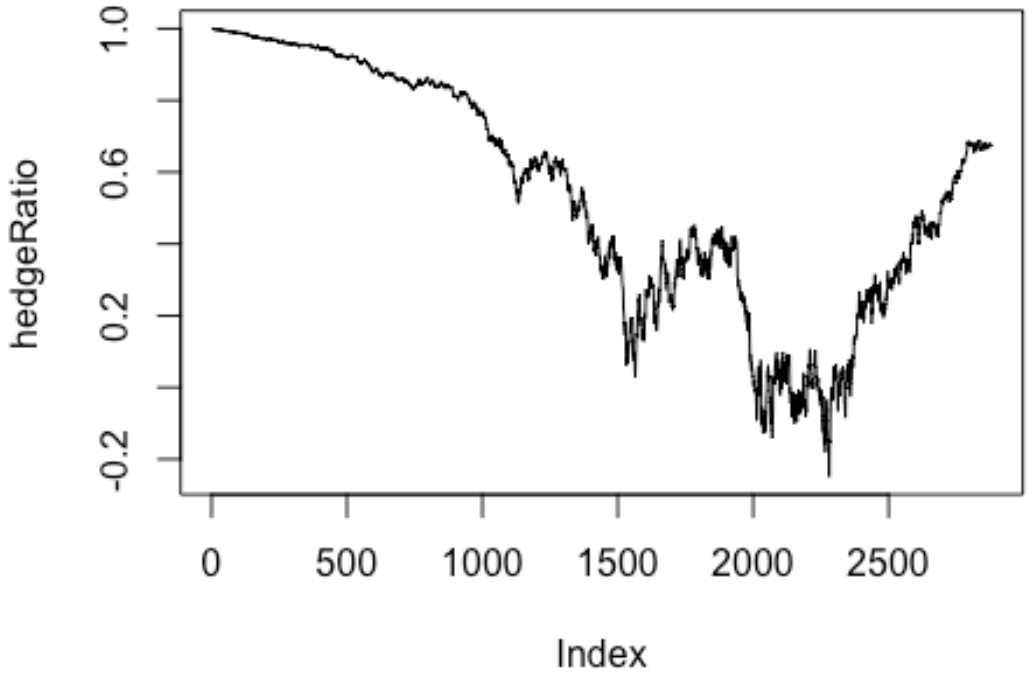
```
set.seed(0)
stock_path = function(){
  stock_price = c()
  stock_price[1] = S0
    for(j in 1:(N)){
    #Simulating N stock paths
      dw=rnorm(1,0 ,1)
      ds = stock_price[j]*u*h + sigma*stock_price[j]*h^0.5*dw
      stock_price[j+1] = stock_price[j]+ds
    }
  return(stock_price)
```

```
}
stock=stock_path()

library(OptionPricing)
T = 60/365
call_option = c()
delta=c()
for(i in 1:length(stock)){
  bs_f = BS_EC((T-(i-1)*h),K,r,sigma,stock[i])
  call_option[i]=bs_f[1]
  delta[i] = bs_f[2]
}


bond_price = c()
portfolio_value = c()
portfolio_value[1] = call_option[1]
bond_price[1] = portfolio_value[1] - delta[1]*stock[1]
for(i in 1:(N)){
  portfolio_value[i+1] = delta[i]*stock[i+1] + bond_price[i]* exp(r*h)
  bond_price[i+1] = portfolio_value[i+1] - delta[i+1]*stock[i+1]- abs(delta[i
+1]-delta[i])*stock[i+1]*0.002
}

#Graph
plot(stock,type="l")
```


```
plot(call_option,type="l",col="yellow")
lines(portfolio_value,col="grey")
```




```
hedgeRatio = portfolio_value/call_option
plot(hedgeRatio,type = "l")
```



Observations:

The Black-Scholes model does not account for transaction charges. With a 20bps transaction charge levied for delta-hedging of the portfolio, the delta hedge becomes ineffective. We are assuming re-balancing of the portfolio every five minutes, implying that we incur transaction costs every five minutes. These transaction charges accumulate over time and we can observe that the deviation of the delta-hedge increases as time passes.

5. Heston Model

a. What is the Black-Scholes prices of the option? What is the Heston price of the option?

```
K = 100
r = 0.04
T=0.5
t = 0
sigma = 0.3
p = -0.5
vt = 0.01
k = 6
theta = 0.02
```

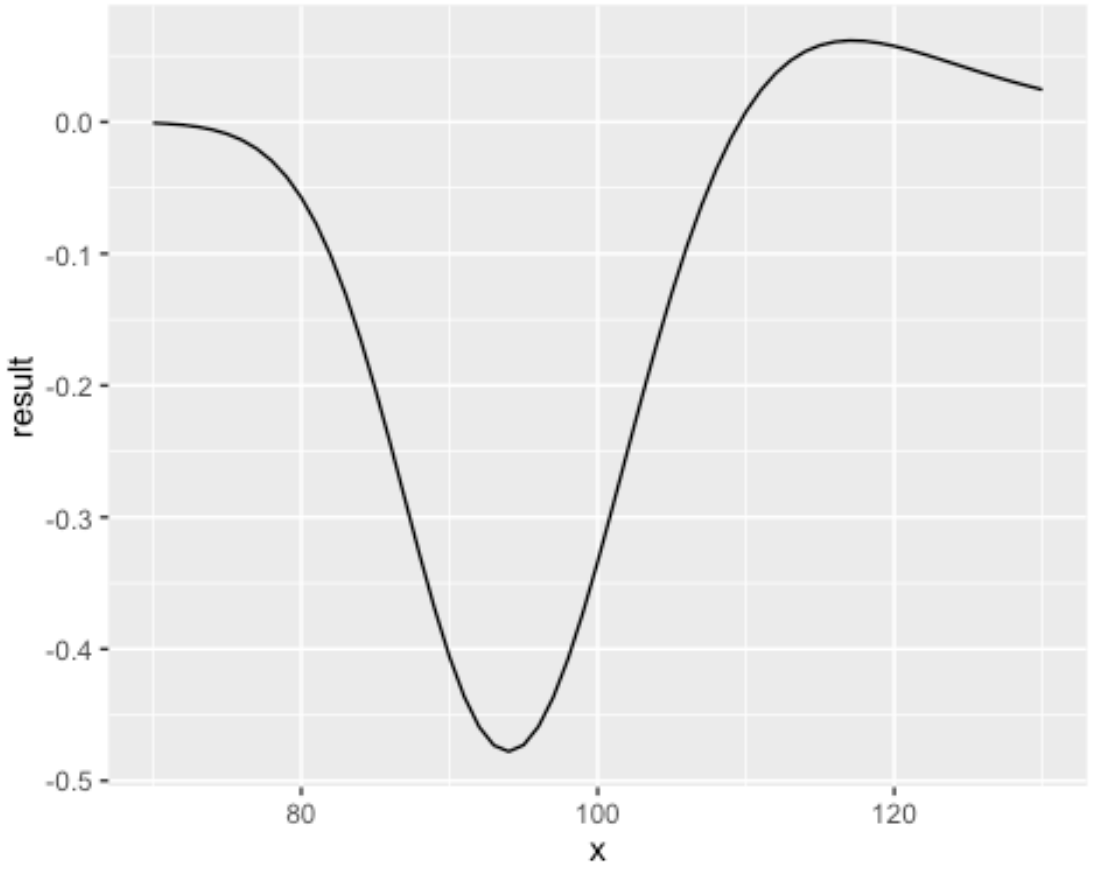
```
St = 100
lambda = 0
Xt = log(St)
maturity = T-t


Integration_Heston = function(K,r,maturity,sigma,p,vt,k,theta,St,lambda){
  indicator = c(1,0)
  Xt = log(St)
  b = k+lambda -p*sigma*indicator
  uVal =c(0.5,-0.5)
  d1 = function(u){
    return(sqrt((p*sigma*u*1i-b[1])**2-sigma^2*(2*uVal[1]*u*1i-u**2)))
  }
  d2 = function(u){
    return(sqrt((p*sigma*u*1i-b[2])**2-sigma^2*(2*uVal[2]*u*1i-u**2)))
  }
  g1 = function(u){
    return((b[1]-p*sigma*u*1i+d1(u))/(b[1]-p*sigma*u*1i-d1(u)))
  }
  g2 = function(u){
    return((b[2]-p*sigma*u*1i+d2(u))/(b[2]-p*sigma*u*1i-d2(u)))
  }
  A1 = function(maturity,u){
    tem = r*u*1i*maturity+k*theta/sigma^2*((b[1]-p*sigma*u*1i+d1(u))*maturity
-2*log((1-g1(u)*exp(d1(u)*maturity))/(1-g1(u))))
    return(tem)
  }
  A2 = function(maturity,u){
    tem = r*u*1i*maturity+k*theta/sigma^2*((b[2]-p*sigma*u*1i+d2(u))*maturity
-2*log((1-g2(u)*exp(d2(u)*maturity))/(1-g2(u))))
    return(tem)
  }
  B1 = function(maturity,u){
    return((b[1]-p*sigma*u*1i+d1(u))/sigma^2*(1-exp(d1(u)*maturity))/(1-g1(u)
*exp(d1(u)*maturity)))
  }
  B2 = function(maturity,u){
    return((b[2]-p*sigma*u*1i+d2(u))/sigma^2*(1-exp(d2(u)*maturity))/(1-g2(u)
*exp(d2(u)*maturity)))
  }
  phi1 = function(u){
    return(exp(A1(maturity,u)+B1(maturity,u)*vt+1i*u*Xt))
  }
  phi2= function(u){
    return(exp(A2(maturity,u)+B2(maturity,u)*vt+1i*u*Xt))
  }
  Function_integrated1 = function(u){
    return(Re(exp(-1i*u*log(K))*phi1(u)/(1i*u)))
  }
```

```
  Function_integrated2 = function(u){
    return(Re(exp(-1i*u*log(K))*phi2(u)/(1i*u)))
  }

  P = c()
  P[1] = 0.5+1/pi*integrate(Function_integrated1,0,500)$value
  P[2] = 0.5+1/pi*integrate(Function_integrated2,0,500)$value
  return(P)
}
Method_Heston = function(K,r,T,t,sigma,p,vt,k,theta,St,lambda){
  maturity = T-t
  P = Integration_Heston(K,r,maturity,sigma,p,vt,k,theta,St,lambda)
  C = St *P[1]-K*P[2]*exp(-r*maturity)
  return(C)
}

library(OptionPricing)
Method_Heston(K,r,T,t,sigma,p,vt,k,theta,St,lambda)

## [1] 4.683304

BS_EC(T,K,r,sqrt(theta),St)

##      price      delta      gamma
## 5.01698061 0.59870633 0.05987063
```

The Black-Scholes price of the option is **5.0170**.
The Heston price of the option is **4.6833**.

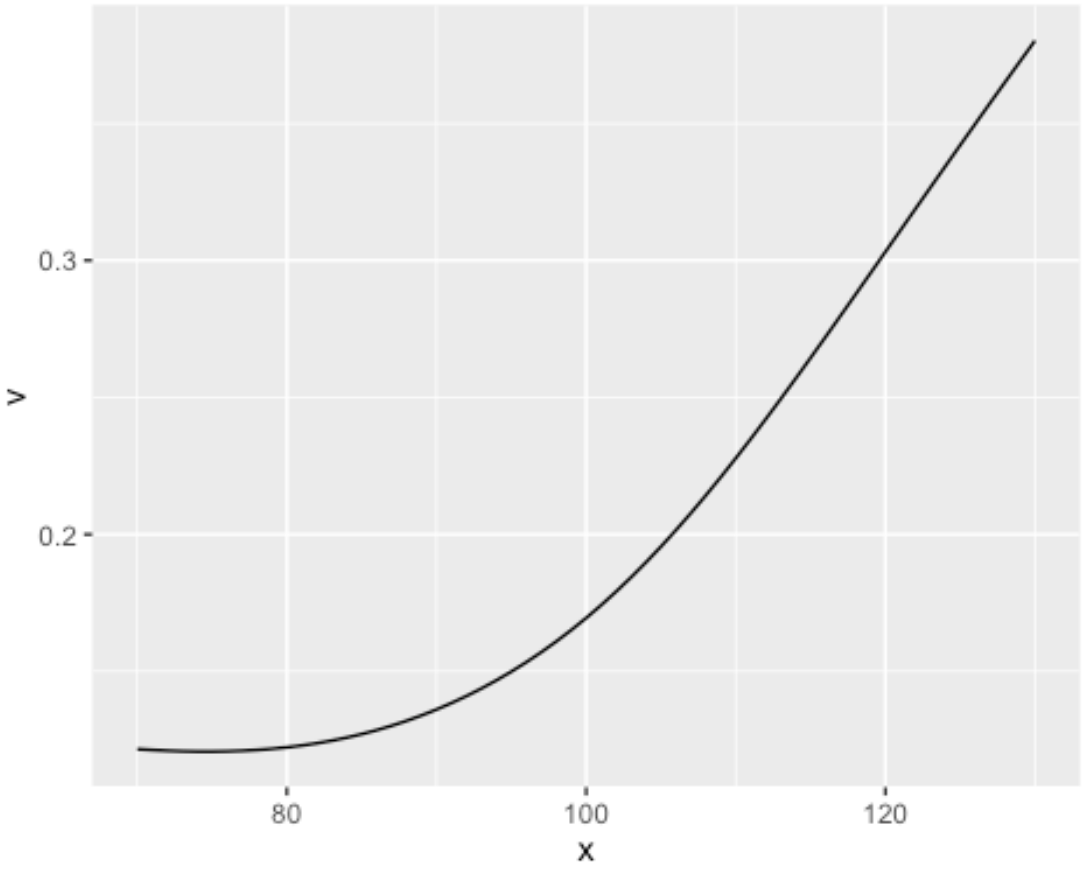
b. Plot the difference (C_{Heston} - $C_{Black\text{-}Scholes}$) for different underlying prices, ranging from $S_t = 70$ to $S_t = 130$.

```
result = c()
hmcall_option = c()
for(St in 70:130){
  bscall_option=BS_EC(T,K,r,sqrt(theta),St)[1]
  hmcall_option[St-69] = Method_Heston(K,r,T,t,sigma,p,vt,k,theta,St,lambda)
  result[St-69] = hmcall_option[St-69]-bscall_option
}
x = 70:130
qplot(x,result,geom="line")
```



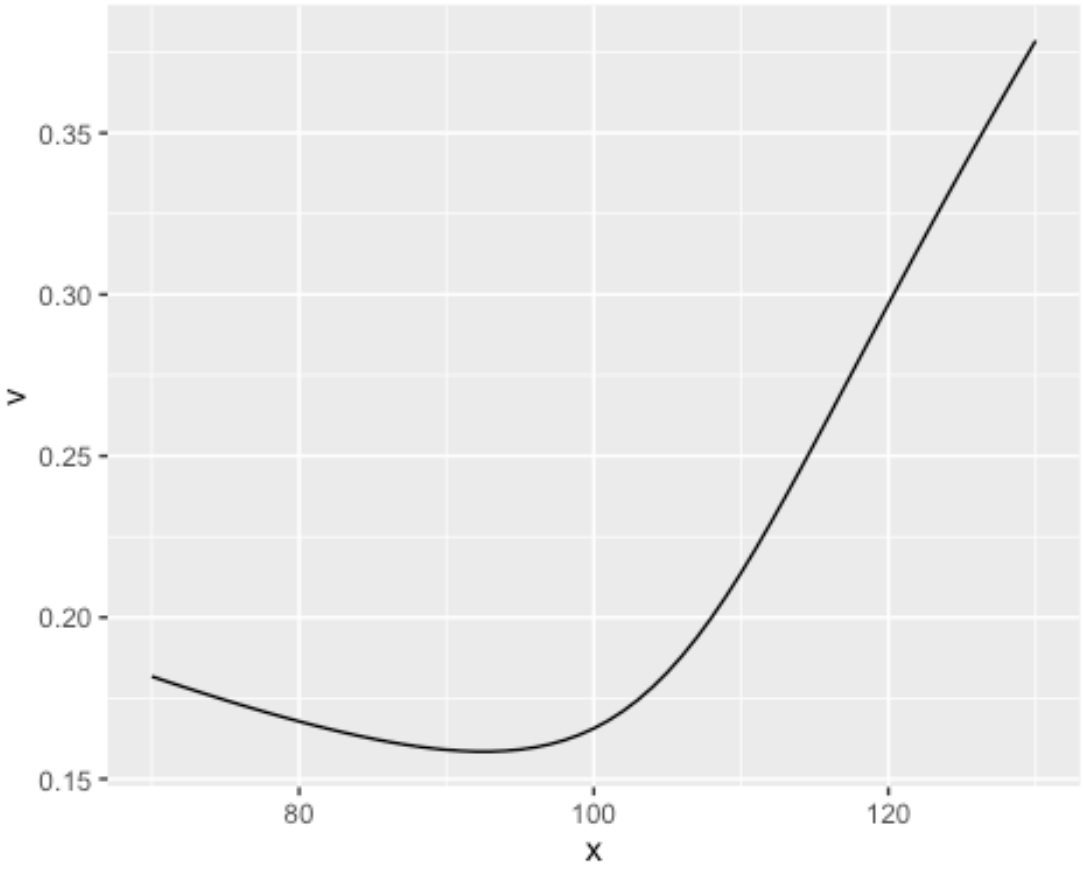
c. For each Heston price obtained at point (b), compute the Black-Scholes implied volatility. Plot the implied volatility as a function of the underlying, from $S_t = 70$ to $S_t = 130$.

```
v=c()
for(St in 70:130){
  v[St-69]=GBSVolatility(price=hmcall_option[St-69],S=St,X=K,Time=T,r=r,b=0)
}
qplot(x,v,geom="line")
```



d. Perform the same exercise as in (c), but now with _ = 0:5. What do you observe?

```
result = c()
hmcall_option = c()
p=0.5
for(St in 70:130){
  hmcall_option[St-69] = Method_Heston(K,r,T,t,sigma,p,vt,k,theta,St,lambda)
  v[St-69]=GBSVolatility(price=hmcall_option[St-69],S=St,X=K,Time=T,r=r,b=0)
}
x = 70:130
qplot(x,v,geom="line")
```



Observations:

If *rho* is positive, we would have a fat right tail and a thin left tail. If *rho* is negative, we would have fat left tails and thin right tails. In reality, we have fatter left tails. A change in skewness impacts the implied volatility curve.