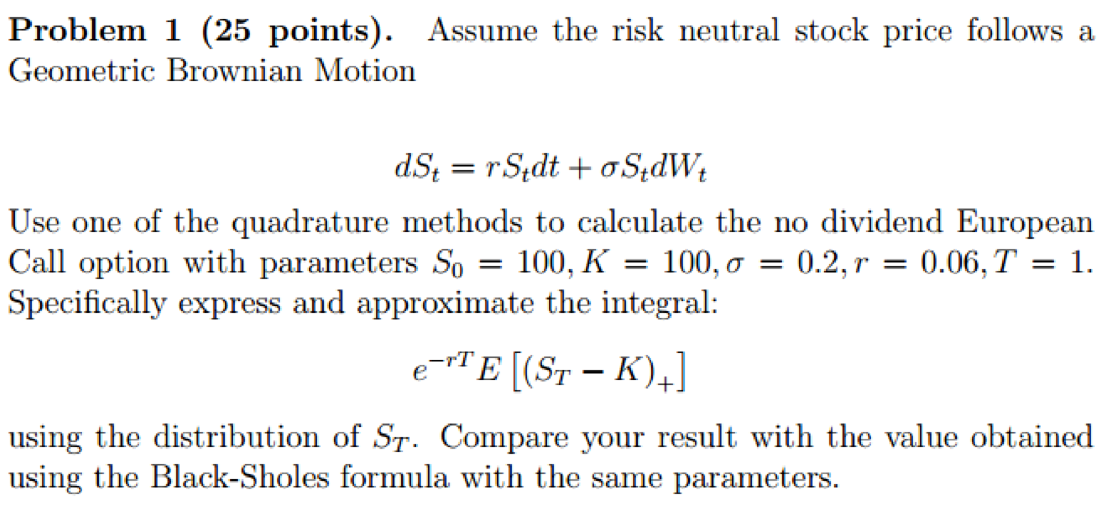
FE621 Computational Methods in Finance

Take Home Midterm

Shuchang Shi

Problem 1 (25 points)



Problem 1 (25 points). Assume the risk neutral stock price follows a Geometric Brownian Motion

$$dS_t = rS_t dt + \sigma S_t dW_t$$

Use one of the quadrature methods to calculate the no dividend European Call option with parameters $S_0 = 100, K = 100, \sigma = 0.2, r = 0.06, T = 1$. Specifically express and approximate the integral:

$$e^{-rT} E\left[(S_T - K)_+\right]$$

using the distribution of S_T. Compare your result with the value obtained using the Black-Sholes formula with the same parameters.

As for the quadrature methods, we have done the two kinds of quadrature methods in the Homework1 Part3 to build the functions of Simpson's quadrature rules and Trapezoidal

Quadrature Rules, So I use the two functions in this question.

```
# Trapezoidal Quadrature Rules
TrapRule = function(a, b, m, f){
  h = (b-a)/(m-1)
  sum = 0.5 * h * (f(a)-f(b))
  for (i in 1:(m-2))
  {
    ai = a + i*h
    sum = sum + h*f(ai)
  }
  sum
}

# Simpson's Quadrature Rules
SimpsonRule = function(a, b, m, f){
  m = m-1
  h = (b-a)/m
  x = seq(from=a, to=b, by=h/2)
  y = f(x)
  ix1 = seq(from=3, by=2, to=2*m+1)
  ix2 = seq(from=2, by=2, to=2*m  )
  h/6 * (y[1] + 2*sum(y[ix1]) + 4*sum(y[ix2]) + y[2*m+1])
}
```

For the Geometric Brownian Motion,

$$dS_t = rS_t dt + \sigma S_t dW_t$$

By the knowledge of FE610,

The St is

$$S_t = S(0)\exp\left[\left(r - \frac{\sigma^2}{2}\right)T + \sigma W\right]$$

And the option price is

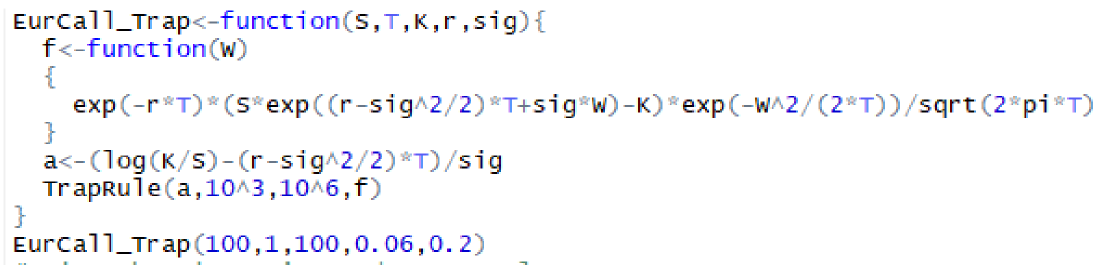
$$e^{-rT}E\left[(S_T - K)_+\right]$$

Which can be written as

$$e^{-rT}E[(S_T - K)_+] = e^{-rT}\int_d^\infty \left(S(0)\exp\left[\left(r - \frac{\sigma^2}{2}\right)T + \sigma W\right] - K\right)\frac{1}{\sqrt{2\pi T}}\exp\left(-\frac{W^2}{2T}\right)dW$$

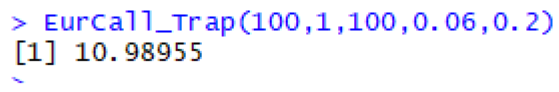
$$d = \frac{\log\frac{K}{S(0)} - \left(r - \frac{\sigma^2}{2}\right)T}{\sigma}$$

Using the Trapezoidal Quadrature Rules:



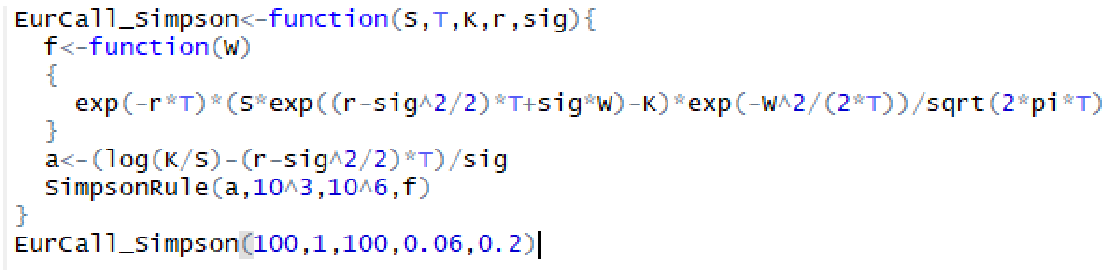
```
EurCall_Trap<-function(S,T,K,r,sig){
  f<-function(W)
  {
    exp(-r*T)*(S*exp((r-sig^2/2)*T+sig*W)-K)*exp(-W^2/(2*T))/sqrt(2*pi*T)
  }
  a<-(log(K/S)-(r-sig^2/2)*T)/sig
  TrapRule(a,10^3,10^6,f)
}
EurCall_Trap(100,1,100,0.06,0.2)
```

Result:

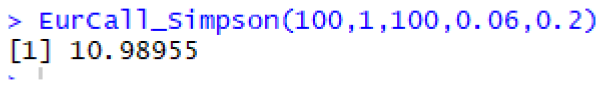


```
> EurCall_Trap(100,1,100,0.06,0.2)
[1] 10.98955
```

Using the Simpson's Quadrature Rules:



```
EurCall_Simpson<-function(S,T,K,r,sig){
  f<-function(W)
  {
    exp(-r*T)*(S*exp((r-sig^2/2)*T+sig*W)-K)*exp(-W^2/(2*T))/sqrt(2*pi*T)
  }
  a<-(log(K/S)-(r-sig^2/2)*T)/sig
  SimpsonRule(a,10^3,10^6,f)
}
EurCall_Simpson(100,1,100,0.06,0.2)
```

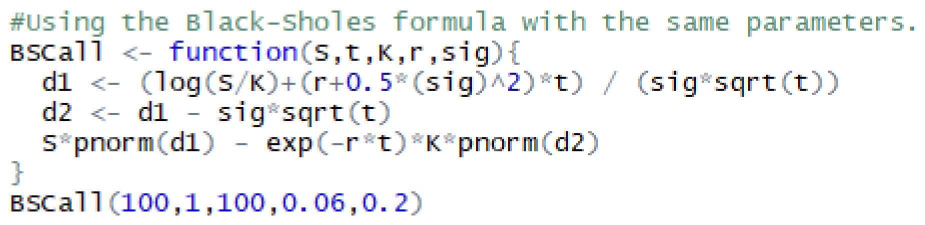


```
> EurCall_Simpson(100,1,100,0.06,0.2)
[1] 10.98955
```

The two quadrature methods' results are same.

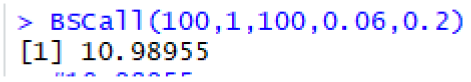
So the European Call option price is 10.98955.

To compare the result with the value obtained using the Black-Sholes formula with the same parameters. we have done the functions of Black-Sholes formula in the Homework1 Part2 to calculate the price of the European Call option, I use the code of the HW1:



```
#Using the Black-Sholes formula with the same parameters.
BSCall <- function(S,t,K,r,sig){
  d1 <- (log(S/K)+(r+0.5*(sig)^2)*t) / (sig*sqrt(t))
  d2 <- d1 - sig*sqrt(t)
  S*pnorm(d1) - exp(-r*t)*K*pnorm(d2)
}
BSCall(100,1,100,0.06,0.2)
```

The result is



```
> BSCall(100,1,100,0.06,0.2)
[1] 10.98955
```

So , the results are same.

Problem 2 (10 points).

Problem 2 (10 points). Consider the Heston model:

$$dS_t = rS_t dt + \sqrt{Y_t} S_t dW_t \tag{1}$$

$$dY_t = \kappa(\bar{Y} - Y_t)dt + \nu\sqrt{Y_t} dZ_t \tag{2}$$

where $r = 0.01$, $\kappa = 2$, $\bar{Y} = 0.16$, $\nu = 1$ are parameters, $Y_0 = 0.17$, $S_0 = 40$. Please answer the following questions:

(a) What is the main reason why the Heston model can produce an implied volatility curve (smile)? Give parameter choices so that the Heston model reduces to the geometric Brownian motion.

(b) Apply expectations to the volatility process Y_t to obtain a deterministic equation in $y_t = \mathbf{E}[Y_t]$. Either solve this equation or approximate the solution and give a numerical answer to the value of $y_{\frac{1}{250}}$

(a)

Since the x-axis is spot price, there would be a point where spot price equals to strike price on the x-axis. We know that vol will be the lowest when spot price equals to strike price. Therefore, the curve will have the smallest value. As the difference spot price and strike price increase, vol will also increase. Therefore, the curve will up rise, and a smile implied volatility curve will be produced.

(b)

$$dY_t = k(\bar{Y} - Y_t)dt + \nu\sqrt{Y_t}\, dZ_t$$

$$= (0.32 - 2Y_t)dt + \sqrt{Y_t}\, dZ_t$$

Set $f(t,x) = e^{2t}x$

$f_x = e^{2t}$ $\quad f_{xx} = 0$ $\quad f_t = 2xe^{2t}$

$$\therefore d(e^{2t}x) = f_t dt + f_x dx + 0$$

$$= 2xe^{2t}dt + e^{2t}dx$$

$X = dY_t \to \; = 2Y_t e^{2t} dt + e^{2t} dY_t \leftarrow dY_t = (0.32 - 2Y(t))dt + \sqrt{Y_t}\, dZ_t$

$= 2Y_t e^{2t} dt + e^{2t}(0.32 - 2Y_t)dt + e^{2t}\sqrt{Y_t}\, dZ_t$

$= 0.32 \cdot e^{2t} dt + e^{2t}\sqrt{Y_t}\, dZ_t. = d(e^{2t} Y_t)$

Integration both side

$\therefore \; e^{2t} Y_t = Y(0) + 0.32 \cdot \int_0^t e^{2t} du + \int_0^t e^{2u}\sqrt{Y(u)}\, dY(u)$

$= Y(0) + 0.16(e^{2t} - 1) + \underline{\int_0^t e^{2u}\sqrt{Y_u}\, dY(u)} \to$

$\therefore E(e^{2t} Y_t) = e^{2t} E(Y_t) = Y(0) + 0.16(e^{2t} - 1)$ is a Ito Integral, so equal 0

and integrating gives:

$$e^{\alpha t} u(t) - u(0) = \mu(e^{\alpha t} - 1)$$

Now using that $u(0) = \mathbb{E}[r_0] = r_0$ after rearranging terms we finally obtain:

$$\mathbb{E}[r_t] = u(t) = \mu + (r_0 - \mu)e^{-\alpha t} \qquad (6)$$

$E(Y_t) = Y_t = e^{-2t} \cdot Y(0) + 0.16(1 - e^{-2t}) \quad \because Y(0) = 0.17$

$\therefore \; Y_{\frac{1}{250}} = E(Y_{\frac{1}{250}}) = (e^{-\frac{2}{250}}) \times 0.17 + 0.16 \times (1 - e^{-\frac{2}{250}})$

$\approx 0.99203 \times 0.17 + 0.16 \times 7.9681 \times 10^{-3}$

$\approx 0.168645 + 1.2749 \times 10^{-3}$

≈ 0.16992

The result is 0.16992

Problem 3 (10 points)

Problem 3 (10 points). Here is the pseudo-code for the valuation of an European Call option using a multiplicative binomial tree. Please identify all the mistakes in the code below. Please indicate how to fix the mistakes.

```
Data: Parameters $K, T, S, r, N, u, d$
dt = T/N
p = (exp(r*dt)-d)/(u-d)
disc = exp(r*dt)
St[0] = S*d
for j = 1 to N do
  | St[j] = St[j-1]*u/d
end
for j = 0 to N do
  | C[j] = max (K-St[j], 0.0)
end
for i = (N-1) down to 0 do
  | for j = 0 to i do
  |   | C[j] = disc * ( (1-p) * C[j] + p * C[j+1])
  | end
end
European_call = C[0]
```

There are 3 errors in this question, mainly refer the Figure 2.3 below

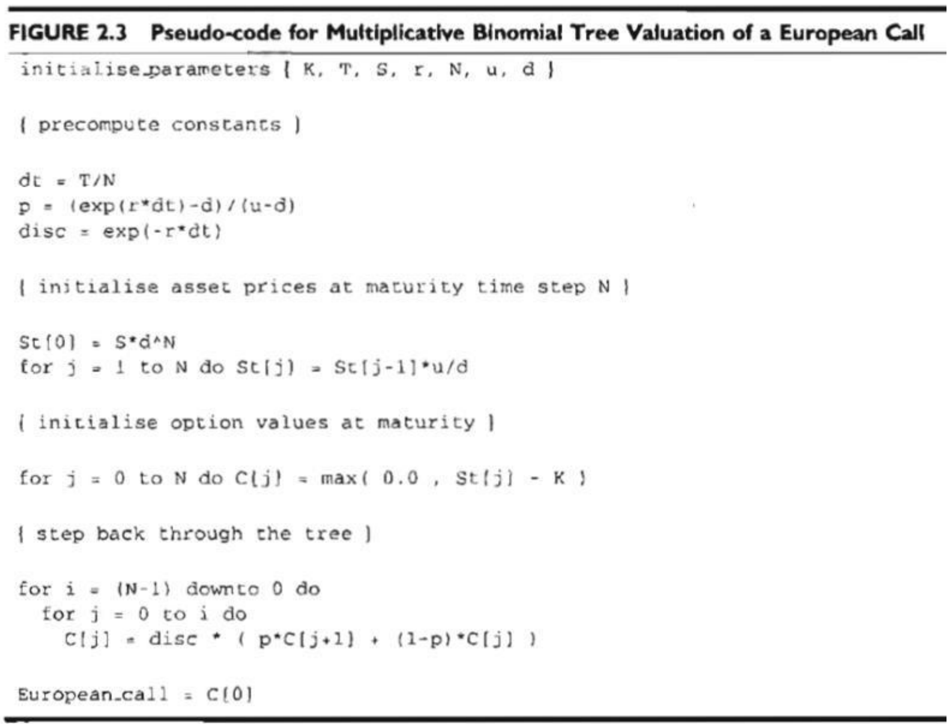


FIGURE 2.3 Pseudo-code for Multiplicative Binomial Tree Valuation of a European Call

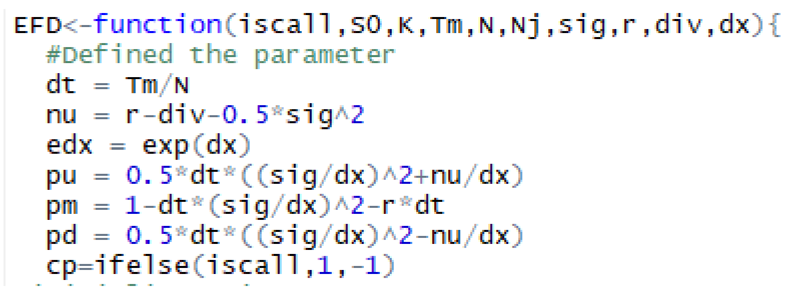
```
initialise_parameters { K, T, S, r, N, u, d }

{ precompute constants }

dt = T/N
p = (exp(r*dt)-d)/(u-d)
disc = exp(-r*dt)

{ initialise asset prices at maturity time step N }

St[0] = S*d^N
for j = 1 to N do St[j] = St[j-1]*u/d

{ initialise option values at maturity }

for j = 0 to N do C[j] = max( 0.0 , St[j] - K )

{ step back through the tree }

for i = (N-1) downto 0 do
  for j = 0 to i do
    C[j] = disc * ( p*C[j+1] + (1-p)*C[j] )

European_call = C[0]
```

```
Data: Parameters K; T; S; r; N; u; d
dt = T/N
p = (exp(r*dt)-d)/(u-d)
disc = exp(-r*dt)
St[0] = S*d^N
for j = 1 to N do
    St[j] = St[j-1]*u/d
end
for j = 0 to N do
    C[j] = max (St[j]-K, 0.0)
end
for i = (N-1) down to 0 do
    for j = 0 to i do
        C[j] = disc * ( (1-p) * C[j] + p * C[j+1])
    end
end
European call = C[0]
```

Problem 4 (40 points)

(a) the Explicit Finite Difference method

At first, I defined the function of the Explicit Finite Difference method, and defined the parameters of the function



```
EFD<-function(iscall,S0,K,Tm,N,Nj,sig,r,div,dx){
  #Defined the parameter
  dt = Tm/N
  nu = r-div-0.5*sig^2
  edx = exp(dx)
  pu = 0.5*dt*((sig/dx)^2+nu/dx)
  pm = 1-dt*(sig/dx)^2-r*dt
  pd = 0.5*dt*((sig/dx)^2-nu/dx)
  cp=ifelse(iscall,1,-1)
```

And then, defined and initialize the option price in a matrix

```
  St = matrix(0,2*Nj+1,Nj+1)
  St[Nj+1] = S0
  for(j in 1:Nj){
    for(i in (Nj-j+2):(Nj+j)){
      St[i-1,j+1]=St[i,j]*exp(dx)
      St[i,j+1]=St[i,j]
      St[i+1,j+1]=St[i,j]*exp(-dx)
    }
  }
```

price the option and step back

```
  V = matrix(0,2*Nj+1,N+1)
  for(i in 1:(2*Nj+1)){
    V[i,N+1]=max( 0 , cp*(St[i,Nj+1]-K) )
  }
  for(j in N:1){
    for(i in 2:(2*Nj)){
      V[i,j]=pu*V[i-1,j+1]+pm*V[i,j+1]+pd*V[i+1,j+1]
    }
```

Finally, Defined the boundary and return the result

```
    bou<-ifelse(iscall,St[1,Nj+1]-St[2,Nj+1],St[2*Nj,Nj+1]-St[2*Nj+1,Nj+1])
    V[1,j]=V[2,j]+ifelse(iscall,bou,0)
    V[2*Nj+1,j]=V[2*Nj,j]+ifelse(iscall,0,bou)
  }
  V[Nj+1,1]
```

Make a test with S0 = 100; K = 100; σ = 0.2; r = 0.06; T = 1, N=5, Nj=5,div=0.02,dx=0.2 for both call and put option.

```
EFD(iscall = TRUE,100,100,1,5,5,0.2,0.06,0.02,0.2)
EFD(iscall = FALSE,100,100,1,5,5,0.2,0.06,0.02,0.2)
```

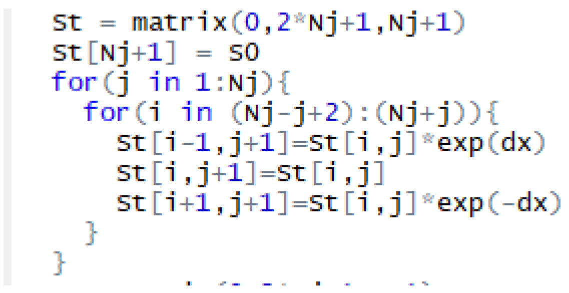
```
> EFD(iscall = TRUE,100,100,1,5,5,0.2,0.06,0.02,0.2)
[1] 8.909771
> EFD(iscall = FALSE,100,100,1,5,5,0.2,0.06,0.02,0.2)
[1] 5.016398
```

(b) Implicit Finite Difference method

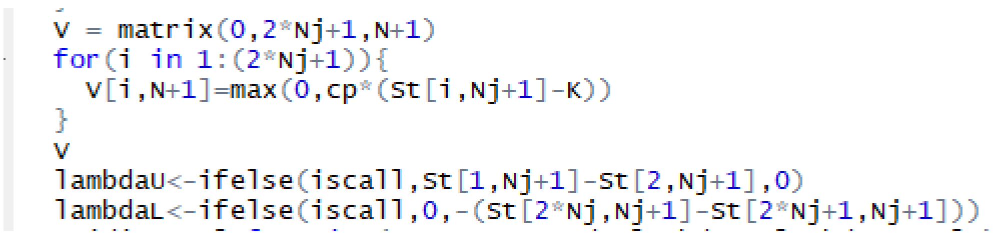
At first, I defined the function of the Finite Difference method, and defined the parameters of the function

```
FD<-function(iscall,S0,K,Tm,N,Nj,sig,r,div,dx){
#Defined the parameter
  nu = r-div-0.5*sig^2
  pu = -0.5 * dt * ( (sig/dx)^2 + nu/dx )
  pm =  1.0 + dt *   (sig/dx)^2 + r*dt
  pd = -0.5 * dt * ( (sig/dx)^2 - nu/dx)
  cp = ifelse(iscall, 1, -1)
```

And then, defined and initialize the option price in a matrix

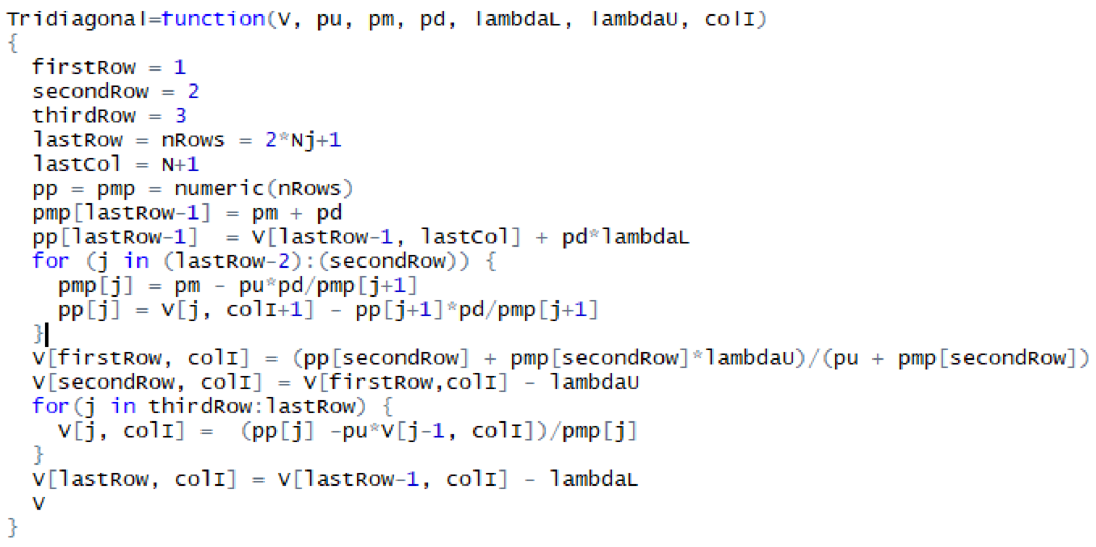


```
St = matrix(0,2*Nj+1,Nj+1)
St[Nj+1] = S0
for(j in 1:Nj){
  for(i in (Nj-j+2):(Nj+j)){
    St[i-1,j+1]=St[i,j]*exp(dx)
    St[i,j+1]=St[i,j]
    St[i+1,j+1]=St[i,j]*exp(-dx)
  }
}
```



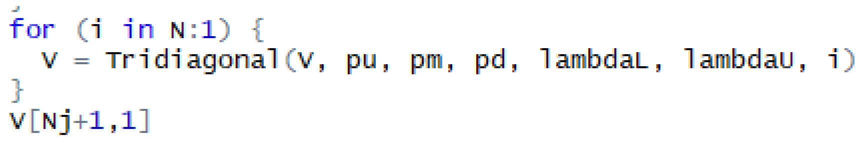
```
V = matrix(0,2*Nj+1,N+1)
for(i in 1:(2*Nj+1)){
  V[i,N+1]=max(0,cp*(St[i,Nj+1]-K))
}
V
lambdaU<-ifelse(iscall,St[1,Nj+1]-St[2,Nj+1],0)
lambdaL<-ifelse(iscall,0,-(St[2*Nj,Nj+1]-St[2*Nj+1,Nj+1]))
```

We should build another function to solve the Tridiagonal



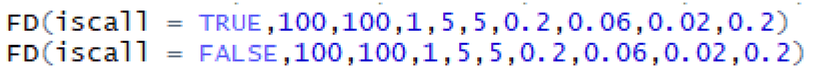
```
Tridiagonal=function(V, pu, pm, pd, lambdaL, lambdaU, colI)
{
  firstRow = 1
  secondRow = 2
  thirdRow = 3
  lastRow = nRows = 2*Nj+1
  lastCol = N+1
  pp = pmp = numeric(nRows)
  pmp[lastRow-1] = pm + pd
  pp[lastRow-1]  = V[lastRow-1, lastCol] + pd*lambdaL
  for (j in (lastRow-2):(secondRow)) {
    pmp[j] = pm - pu*pd/pmp[j+1]
    pp[j] = V[j, colI+1] - pp[j+1]*pd/pmp[j+1]
  }
  V[firstRow, colI] = (pp[secondRow] + pmp[secondRow]*lambdaU)/(pu + pmp[secondRow])
  V[secondRow, colI] = V[firstRow,colI] - lambdaU
  for(j in thirdRow:lastRow) {
    V[j, colI] =  (pp[j] -pu*V[j-1, colI])/pmp[j]
  }
  V[lastRow, colI] = V[lastRow-1, colI] - lambdaL
  V
}
```

And calculate the option price



```
for (i in N:1) {
  V = Tridiagonal(V, pu, pm, pd, lambdaL, lambdaU, i)
}
V[Nj+1,1]
```

Make a test with S0 = 100; K = 100; σ = 0.2; r = 0.06; T = 1, N=5, Nj=5,div=0.02,dx=0.2 for both call and put option.



```
FD(iscall = TRUE,100,100,1,5,5,0.2,0.06,0.02,0.2)
FD(iscall = FALSE,100,100,1,5,5,0.2,0.06,0.02,0.2)
```

The Result:

```
> FD(iscall = TRUE,100,100,1,5,5,0.2,0.06,0.02,0.2)
[1] 8.326235
> FD(iscall = FALSE,100,100,1,5,5,0.2,0.06,0.02,0.2)
[1] 4.493932
```

(c)

```
Points<-function(sig,Tm,N,e)
{
  for(i in 1:N)
  {
    dt<-Tm/i
    dx<-sig*sqrt(3*dt)
    if(dx^2+dt<e)
      break
  }
  point<-c(i,dx,dt)
  point
}
```

Make a test σ = 0.25; Tm=1, N=10^6,e=0.001

Result:

```
> Points(0.25,1,10^6,0.001)
[1] 1.188000e+03 1.256297e-02 8.417508e-04
```

(d)

In the C, the number of steps is 1188

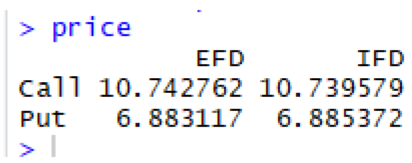
And $S_0 = 100$; $K = 100$, $T = 1$ year, $\sigma = 25\%$; $r = 6\%$; $\delta = 0{:}03$.Nj=5

```
EFD.call<-EFD(iscall = TRUE,100,100,1,1188,5,0.25,0.06,0.02,0.2)
EFD.put<-EFD(iscall = FALSE,100,100,1,1188,5,0.25,0.06,0.02,0.2)
IFD.call<-IFD(iscall = TRUE,100,100,1,1188,5,0.25,0.06,0.02,0.2)
IFD.put<-IFD(iscall = FALSE,100,100,1,1188,5,0.25,0.06,0.02,0.2)
price<-matrix(c(EFD.call,EFD.put,IFD.call,IFD.put),ncol=2,nrow=2)
colnames(price)<-c("EFD", "IFD")
rownames(price)<-c("Call","Put")
price
```

And put the result into a matrix

Result:

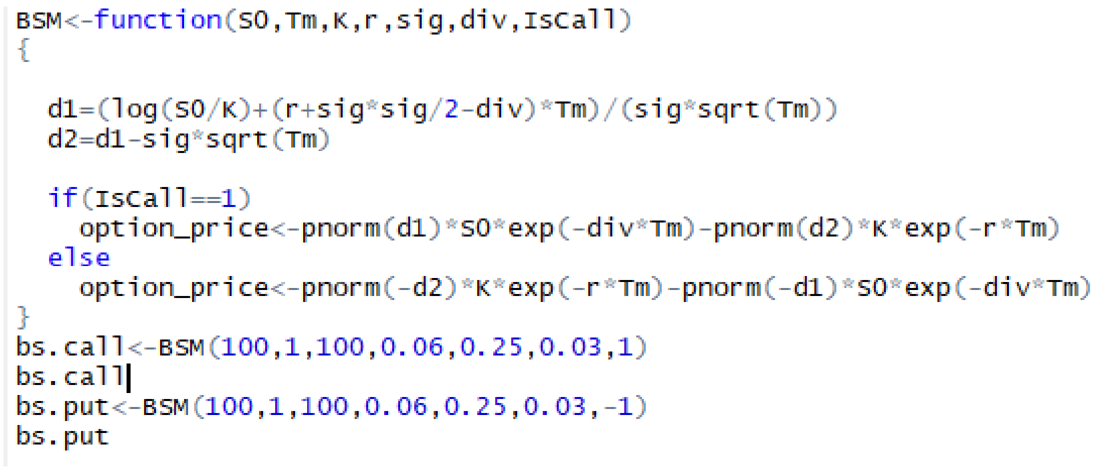
```
      Explicit  Implicit
Call 10.742762 10.739579
Put   6.883117  6.885372
```



```
> price
          EFD       IFD
Call 10.742762 10.739579
Put   6.883117  6.885372
>
```

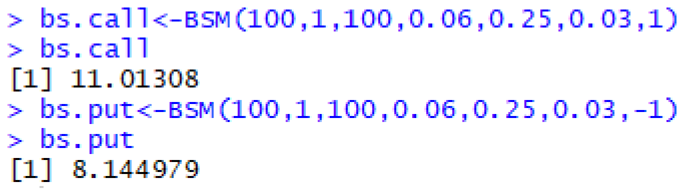
(e)

The Black-Sholes formula with the the parameters $S_0 = 100; K = 100, T = 1$ year, $\sigma =$ 25%; r = 6%; $\delta = 0{:}03$.. we have done the functions of Black-Sholes formula in the Homework1 Part2 to calculate the price of the European Call option, I use the code of the HW1:

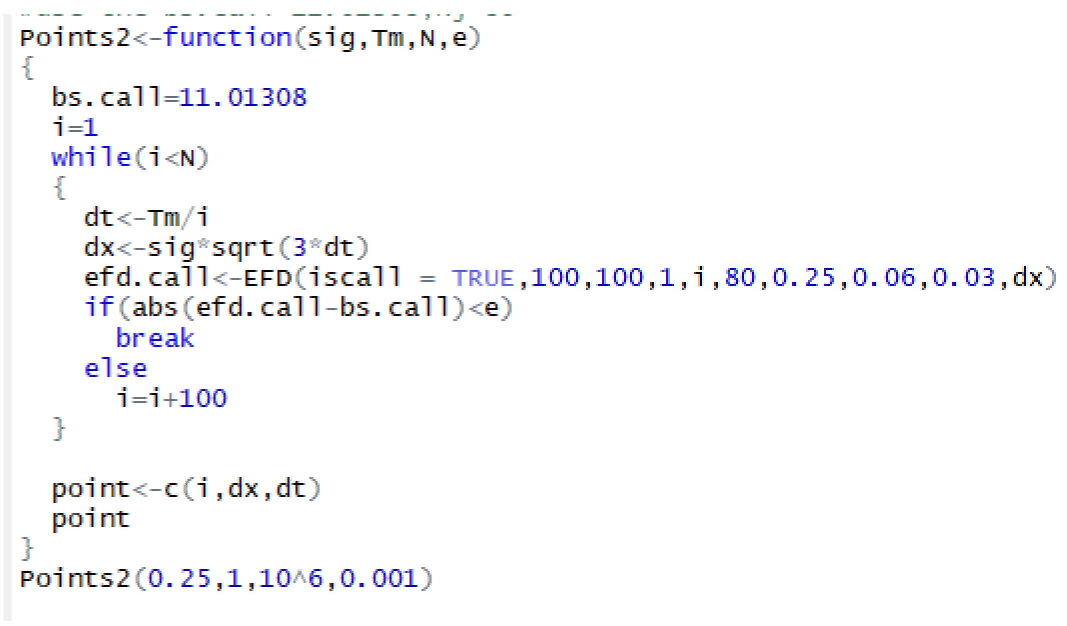


```
BSM<-function(S0,Tm,K,r,sig,div,IsCall)
{

  d1=(log(S0/K)+(r+sig*sig/2-div)*Tm)/(sig*sqrt(Tm))
  d2=d1-sig*sqrt(Tm)

  if(IsCall==1)
    option_price<-pnorm(d1)*S0*exp(-div*Tm)-pnorm(d2)*K*exp(-r*Tm)
  else
    option_price<-pnorm(-d2)*K*exp(-r*Tm)-pnorm(-d1)*S0*exp(-div*Tm)
}
bs.call<-BSM(100,1,100,0.06,0.25,0.03,1)
bs.call
bs.put<-BSM(100,1,100,0.06,0.25,0.03,-1)
bs.put
```

Result:

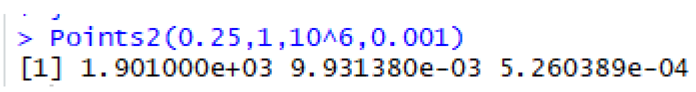


```
> bs.call<-BSM(100,1,100,0.06,0.25,0.03,1)
> bs.call
[1] 11.01308
> bs.put<-BSM(100,1,100,0.06,0.25,0.03,-1)
> bs.put
[1] 8.144979
```



```
Points2<-function(sig,Tm,N,e)
{
  bs.call=11.01308
  i=1
  while(i<N)
  {
    dt<-Tm/i
    dx<-sig*sqrt(3*dt)
    efd.call<-EFD(iscall = TRUE,100,100,1,i,80,0.25,0.06,0.03,dx)
    if(abs(efd.call-bs.call)<e)
      break
    else
      i=i+100
  }

  point<-c(i,dx,dt)
  point
}
Points2(0.25,1,10^6,0.001)
```

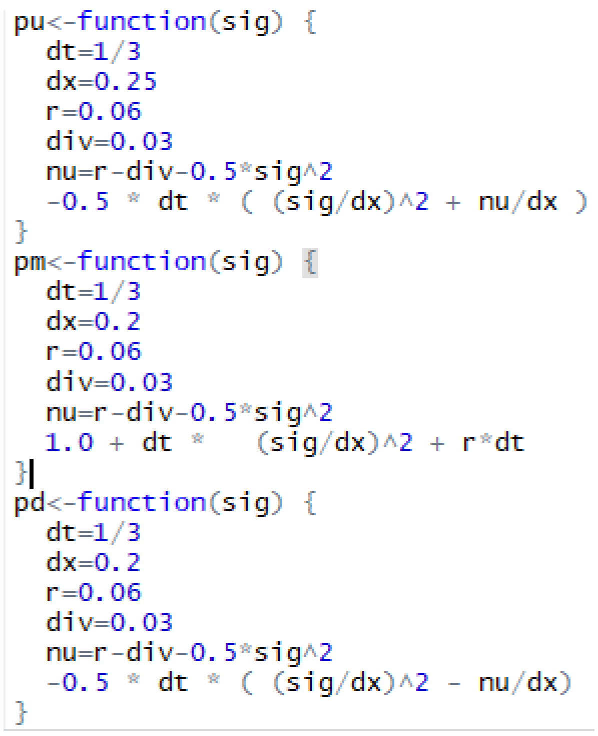
Result:



```
> Points2(0.25,1,10^6,0.001)
[1] 1.901000e+03 9.931380e-03 5.260389e-04
```

(f)

Defined the function of Pu, Pm and Pd,and use the K = 100, T = 1.0, S =100, r = 0.06, div=0.03,dx=0.25



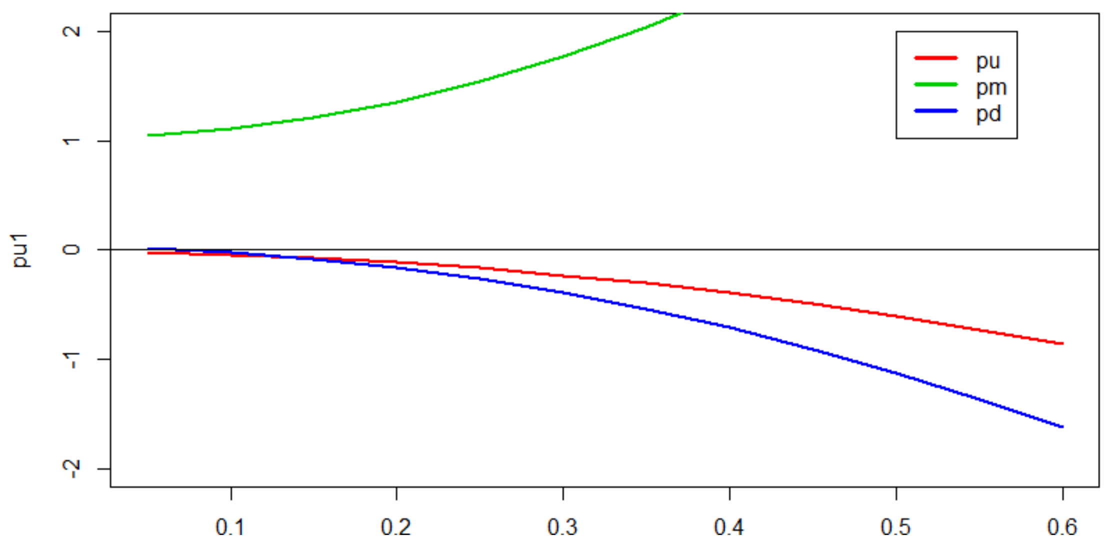
```
pu<-function(sig) {
  dt=1/3
  dx=0.25
  r=0.06
  div=0.03
  nu=r-div-0.5*sig^2
  -0.5 * dt * ( (sig/dx)^2 + nu/dx )
}
pm<-function(sig) {
  dt=1/3
  dx=0.2
  r=0.06
  div=0.03
  nu=r-div-0.5*sig^2
  1.0 + dt *    (sig/dx)^2 + r*dt
}
pd<-function(sig) {
  dt=1/3
  dx=0.2
  r=0.06
  div=0.03
  nu=r-div-0.5*sig^2
  -0.5 * dt * ( (sig/dx)^2 - nu/dx)
}
```

Set the sigma is $\{0.05, 0.1, 0.15, ..., 0.6\}$

```
s = seq(from=0.05, to=0.6, by=0.05)
pu1 = pu(s)
pm1 = pm(s)
pd1 = pd(s)
```

And plot the result:

```
plot(s, pu1, ylim=c(-2,2), type="n")
title(theTitle, cex=3)
mtext(text=xlab1, side=1, line=2, cex=1.25)
mtext(text=xlab2, side=1, line=3, cex=1.25)
lines(s, pu1, lwd=2, col=2)
lines(s, pm1, lwd=2, col=3)
lines(s, pd1, lwd=2, col=4)
abline(h=0)
legend(0.5, 2, legend=c('pu', 'pm', 'pd'), col=2:4, lty=1, lwd=3)
```

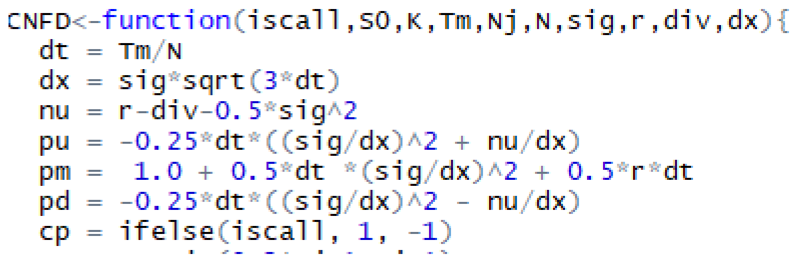


From the figure, we can observe that when the Sigma(volatility)goes up from 0.05 to 0.6, Pm becomes larger. But Pu and Pd become smaller. So the implicit finite method is opposite to the explicit finite method

(g)

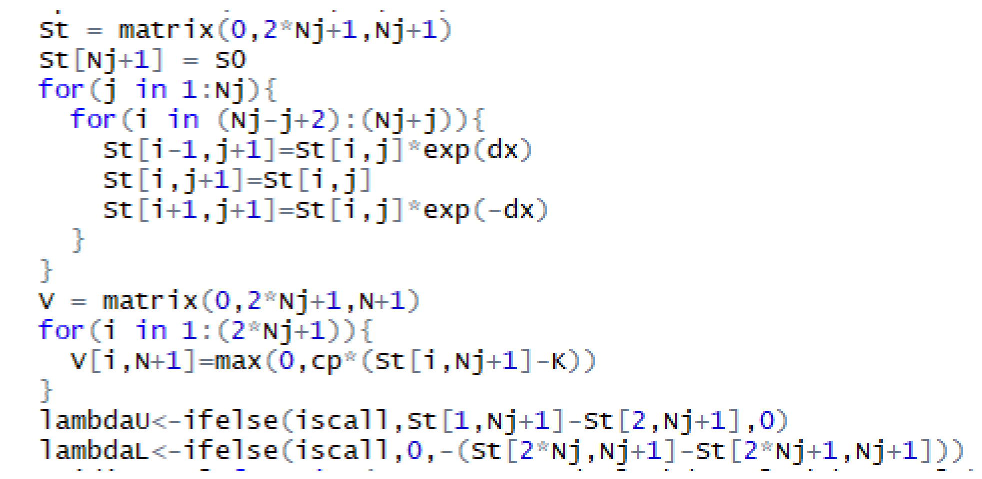
Just like the Implicit Finite Difference method,

At first, I defined the function of the Finite Difference method, and defined the parameters of the function



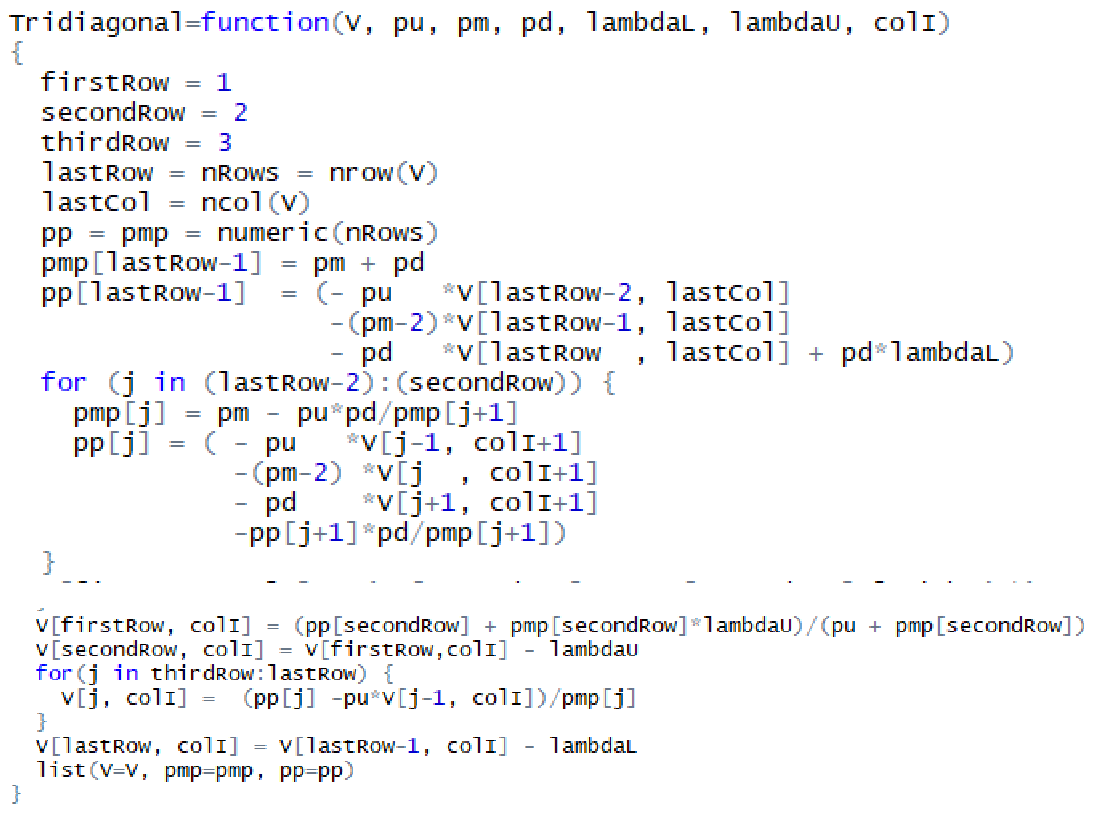
```
CNFD<-function(iscall,S0,K,Tm,Nj,N,sig,r,div,dx){
  dt = Tm/N
  dx = sig*sqrt(3*dt)
  nu = r-div-0.5*sig^2
  pu = -0.25*dt*((sig/dx)^2 + nu/dx)
  pm =  1.0 + 0.5*dt *(sig/dx)^2 + 0.5*r*dt
  pd = -0.25*dt*((sig/dx)^2 - nu/dx)
  cp = ifelse(iscall, 1, -1)
```

And then, defined and initialize the option price in a matrix



```
  St = matrix(0,2*Nj+1,Nj+1)
  St[Nj+1] = S0
  for(j in 1:Nj){
    for(i in (Nj-j+2):(Nj+j)){
      St[i-1,j+1]=St[i,j]*exp(dx)
      St[i,j+1]=St[i,j]
      St[i+1,j+1]=St[i,j]*exp(-dx)
    }
  }
  V = matrix(0,2*Nj+1,N+1)
  for(i in 1:(2*Nj+1)){
    V[i,N+1]=max(0,cp*(St[i,Nj+1]-K))
  }
  lambdaU<-ifelse(iscall,St[1,Nj+1]-St[2,Nj+1],0)
  lambdaL<-ifelse(iscall,0,-(St[2*Nj,Nj+1]-St[2*Nj+1,Nj+1]))
```

We should build another function to solve the Tridiagonal



```
Tridiagonal=function(V, pu, pm, pd, lambdaL, lambdaU, colI)
{
  firstRow = 1
  secondRow = 2
  thirdRow = 3
  lastRow = nRows = nrow(V)
  lastCol = ncol(V)
  pp = pmp = numeric(nRows)
  pmp[lastRow-1] = pm + pd
  pp[lastRow-1]  = (- pu   *V[lastRow-2, lastCol]
                    -(pm-2)*V[lastRow-1, lastCol]
                    - pd   *V[lastRow  , lastCol] + pd*lambdaL)
  for (j in (lastRow-2):(secondRow)) {
    pmp[j] = pm - pu*pd/pmp[j+1]
    pp[j] = ( - pu    *V[j-1, colI+1]
              -(pm-2) *V[j  , colI+1]
              - pd    *V[j+1, colI+1]
              -pp[j+1]*pd/pmp[j+1])
  }

  V[firstRow, colI] = (pp[secondRow] + pmp[secondRow]*lambdaU)/(pu + pmp[secondRow])
  V[secondRow, colI] = V[firstRow,colI] - lambdaU
  for(j in thirdRow:lastRow) {
    V[j, colI] =  (pp[j] -pu*V[j-1, colI])/pmp[j]
  }
  V[lastRow, colI] = V[lastRow-1, colI] - lambdaL
  list(V=V, pmp=pmp, pp=pp)
}
```

And calculate the option price and return the result

```
  for (i in N:1) {
    h = Tridiagonal(V, pu, pm, pd, lambdaL, lambdaU, i)
    V = h$V
  }
V[Nj+1,1]
}
```

Use the same parameters as in part(d) and the same number of steps in the grid. Put the results of the3 methods (EFD, IFD, CNFD) side by side in a table

```
EFD.call<-EFD(iscall = TRUE,100,100,1,200,200,0.25,0.06,0.02,0.2)
EFD.put<-EFD(iscall = FALSE,100,100,1,200,200,0.25,0.06,0.02,0.2)
IFD.call<-IFD(iscall = TRUE,100,100,1,200,200,0.25,0.06,0.02,0.2)
IFD.put<-IFD(iscall = FALSE,100,100,1,200,200,0.25,0.06,0.02,0.2)
CNFD.call<-CNFD(iscall = TRUE,100,100,1,200,200,0.25,0.06,0.02,0.2)
CNFD.put<-CNFD(iscall = FALSE,100,100,1,200,200,0.25,0.06,0.02,0.2)
CNFD.call
CNFD.put
price2<-matrix(c(EFD.call,EFD.put,IFD.call,IFD.put,CNFD.call,CNFD.put),ncol=3,nrow=2)
colnames(price2)<-c("EFD", "IFD", "CNFD")
rownames(price2)<-c("Call","Put")
price2
```

Result:

```
> price2
           EFD       IFD      CNFD
Call 10.750464 10.731552 11.585743
Put   6.890345  6.872934  7.741956
>
```

From the result we can observe that the results of these three methods are almost same.But the results of CNFD is a little larger than the others

(h)

Use the parameters as S0 = 100; K = 100; σ = 0.25; r = 0.06; T = 1, N=200, Nj=200,div=0.02

delta<-(EFD2(iscall = TRUE,100,1,100+0.001*100,0.06,0.25,200,200,0.03)-EFD2(iscall = TRUE,100,1,100-0.001*100,0.06,0.25,200,200,0.03))/(2*0.001*100)

delta

gamma<-((EFD2(iscall = TRUE,100,1,100+0.001*100,0.06,0.25,200,200,0.03)-EFD2(iscall = TRUE,100,1,100,0.06,0.25,200,200,0.03))/(0.001*100)-(EFD2(iscall = TRUE,100,1,100,0.06,0.25,200,200,0.03)-EFD2(iscall = TRUE,100,1,100-0.001*100,0.06,0.25,200,200,0.03))/(0.001*100))/0.5*(2*0.001*100)

gamma

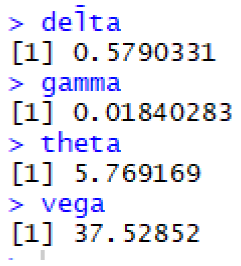
theta<-(EFD2(iscall = TRUE,100,1+0.001*1,100,0.06,0.25,200,200,0.03)-EFD2(iscall = TRUE,100,1,100,0.06,0.25,200,200,0.03))/(0.001*1)

theta

vega<-(EFD2(iscall = TRUE,100,1,100,0.06,0.25+0.001*0.25,200,200,0.03)-EFD2(iscall = TRUE,100,1,100,0.06,0.25-0.001*0.25,200,200,0.03))/(2*0.001*0.25)

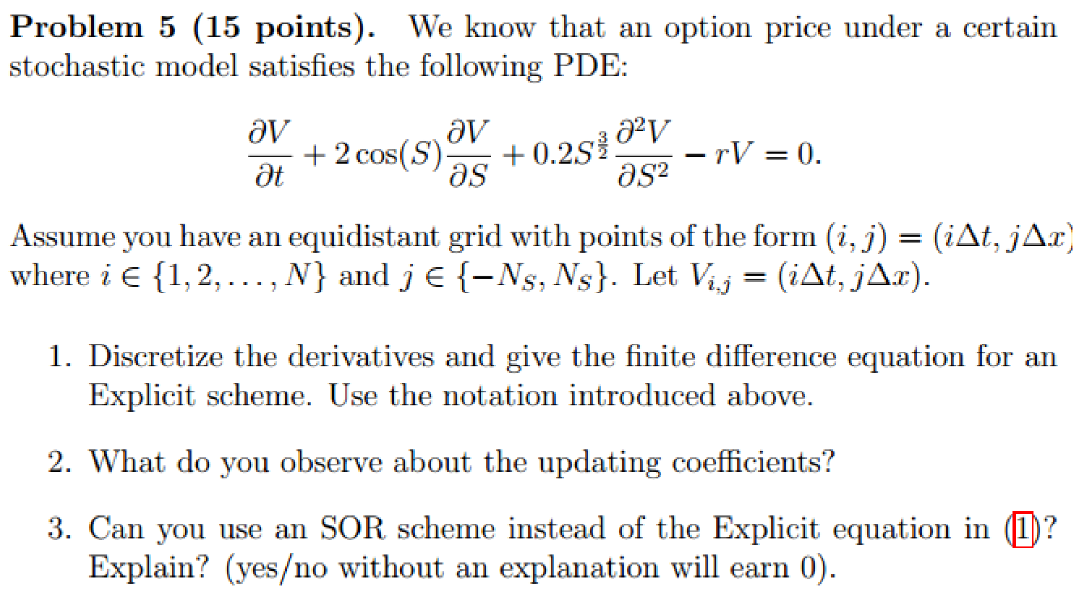
vega

result:



```
> delta
[1] 0.5790331
> gamma
[1] 0.01840283
> theta
[1] 5.769169
> vega
[1] 37.52852
```

Problem 5 (15 points).



Problem 5 (15 points). We know that an option price under a certain stochastic model satisfies the following PDE:

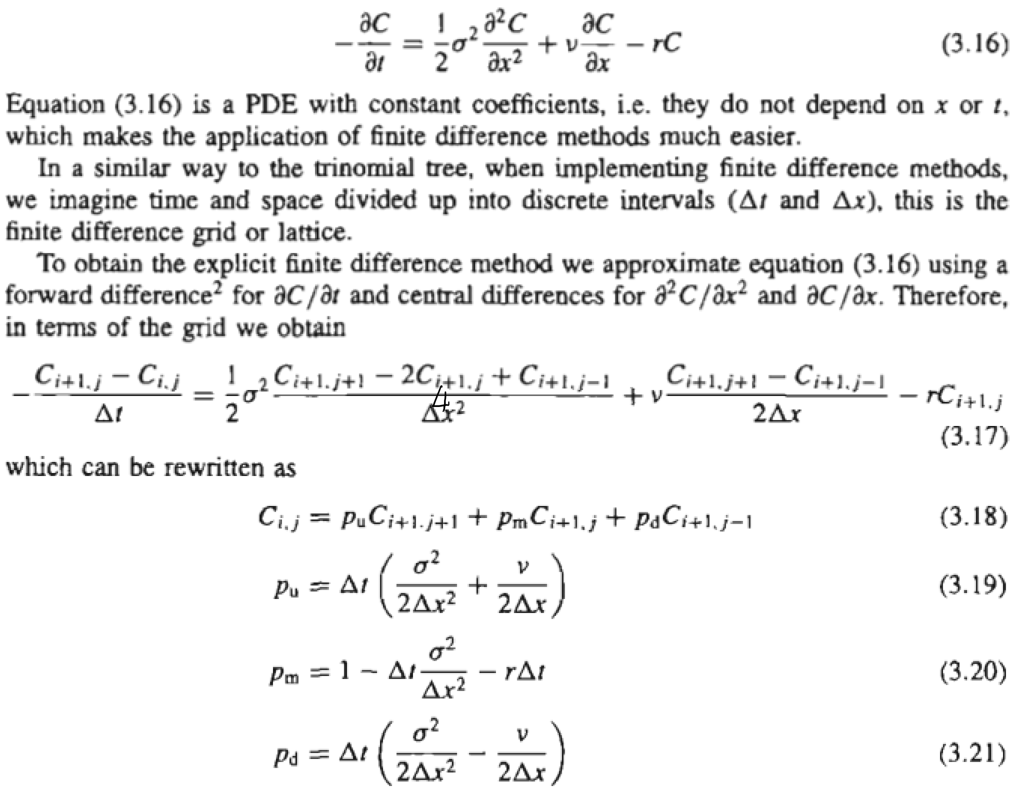
$$\frac{\partial V}{\partial t} + 2\cos(S)\frac{\partial V}{\partial S} + 0.2S^{\frac{3}{2}}\frac{\partial^2 V}{\partial S^2} - rV = 0.$$

Assume you have an equidistant grid with points of the form $(i, j) = (i\Delta t, j\Delta x)$ where $i \in \{1, 2, \ldots, N\}$ and $j \in \{-N_S, N_S\}$. Let $V_{i,j} = (i\Delta t, j\Delta x)$.

1. Discretize the derivatives and give the finite difference equation for an Explicit scheme. Use the notation introduced above.

2. What do you observe about the updating coefficients?

3. Can you use an SOR scheme instead of the Explicit equation in (1)? Explain? (yes/no without an explanation will earn 0).



$$-\frac{\partial C}{\partial t} = \frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial x^2} + \nu\frac{\partial C}{\partial x} - rC \tag{3.16}$$

Equation (3.16) is a PDE with constant coefficients, i.e. they do not depend on x or t, which makes the application of finite difference methods much easier.

In a similar way to the trinomial tree, when implementing finite difference methods, we imagine time and space divided up into discrete intervals (Δt and Δx), this is the finite difference grid or lattice.

To obtain the explicit finite difference method we approximate equation (3.16) using a forward difference[2] for $\partial C/\partial t$ and central differences for $\partial^2 C/\partial x^2$ and $\partial C/\partial x$. Therefore, in terms of the grid we obtain

$$-\frac{C_{i+1,j} - C_{i,j}}{\Delta t} = \frac{1}{2}\sigma^2\frac{C_{i+1,j+1} - 2C_{i+1,j} + C_{i+1,j-1}}{\Delta x^2} + \nu\frac{C_{i+1,j+1} - C_{i+1,j-1}}{2\Delta x} - rC_{i+1,j} \tag{3.17}$$

which can be rewritten as

$$C_{i,j} = p_u C_{i+1,j+1} + p_m C_{i+1,j} + p_d C_{i+1,j-1} \tag{3.18}$$

$$p_u = \Delta t\left(\frac{\sigma^2}{2\Delta x^2} + \frac{\nu}{2\Delta x}\right) \tag{3.19}$$

$$p_m = 1 - \Delta t\frac{\sigma^2}{\Delta x^2} - r\Delta t \tag{3.20}$$

$$p_d = \Delta t\left(\frac{\sigma^2}{2\Delta x^2} - \frac{\nu}{2\Delta x}\right) \tag{3.21}$$

According to the (3.16) to (3.21) in the book ***CS-Implementing Derivatives Models page 58***

the (3.16).

$$-\frac{\partial C}{\partial t} = \frac{1}{2}\sigma^2\frac{\partial^2 C}{\partial x^2} + \nu\frac{\partial C}{\partial x} - rC$$

Lee $x = \ln(S)$

∴ we get

$$-\frac{\partial V}{\partial t} = 0.2S^{-\frac{1}{2}}\frac{\partial^2 V}{\partial S^2} + \frac{2\cos(S)}{S}\frac{\partial V}{\partial S} - rV$$

and $p_u = \Delta t\left(\frac{0.2S^{-\frac{1}{2}}}{\Delta x^2} + \frac{\cos(S)}{S\Delta x}\right)$

$p_m = 1 - \Delta t\left(\frac{0.4S^{-\frac{1}{2}}}{\Delta x^2} + r\right)$

$p_d = \Delta t\left(\frac{0.2S^{-\frac{1}{2}}}{\Delta x^2} - \frac{\cos(S)}{S\Delta x}\right)$

(2)

(2) (2) $P_u + P_m + P_d = 1 - r\delta t$, which should

$P_u + P_m + P_d = 1$

$\therefore$ $r\delta t$ should equal to 0

that means $\delta t \longrightarrow 0$

(3) Mainly according to the April 4's FE621-B Note of Professor Bozdog.

$$\begin{cases} Au_{i+1} \geq b_i \\ u_{i+1} \geq g_{i+1} \\ (u_{i+1} - g_{i+1})(Au_{i+1}) - b_i = 0 \end{cases}$$

write $Au_{i+1} - b_i$ equation as

$$J_{i+1,j} = \beta b_{i,j} + \gamma u_{i+1,j+1} + \delta u_{i+1,j+1}$$

$\because$ SOR will give solution of $Au_{i+1} = b_i$

$\therefore$ we need $u_{i+1} \geq g_{i+1}$

In order to enforce the condition

$$y^k_{i+1} = \beta b_{i,j} + \gamma u^{k-1}_{i+1,j+1} + \delta u^{k-1}_{i+1,j-1}$$

$$u^k_{i+1,j} = \max[v^{k-1}_{i+1,j} + w(y^k_{i+1,j} - u^{k-1}_{i+1,j}), g_{i+1,j})$$

Iterate over k until solution converges to certain Σ

After we solve for all u_{ij} points

$\Rightarrow$ we find the efficient frontier.

by going back and extracting at every $i\delta t$ step the value of S for which $h_{ij} = g_{ij}$ at point $i\delta t$

So, we can use an SOR scheme instead of the Explicit equation.