FE621 - Midterm

Napat Loychindarat

April 9, 2018

Problem 1

Using quadrature method below to calculate the no dividend European Call option with parameters $S_0 = 100, K = 100, \sigma = 0.2, r = 0.06, T = 1$.

$$c = \mathrm{e}^{-rt} \int_{In(K)}^{ln(400)} (e^{\xi} - K)\left(\frac{1}{\sqrt{2\pi\sigma^2\tau}} exp\left(-\frac{1}{2\sigma^2\tau}\left(\xi - z - \left(r - \frac{\sigma^2}{2}\right)\tau\right)^2\right)\right) d\xi$$

With different number of subinterval, Simpson method converges to Black-Scholes faster than Trapezoid method.

```
##     m Trapezoid  Simpson BlackScholes
## 1   9  10.51166 11.13406     10.98955
## 2  17  10.87336 10.99392     10.98955
## 3  33  10.96069 10.98980     10.98955
## 4  65  10.98235 10.98956     10.98955
## 5 129  10.98775 10.98955     10.98955
## 6 257  10.98910 10.98955     10.98955
## 7 513  10.98944 10.98955     10.98955
```

Code:

```
T = 1
S0 = 100
K = 100
r = 0.06
sig = 0.2
D = 0

z <- log(S0)
zmax <- log(400)
xmax <- exp(zmax)


f <- function(xj, K, tao, z, sig, r){

    fx <- (exp(xj) - K)*(1/sqrt(2*pi*(sig^2)*tao)*exp((-0.5/(sig^2*tao))*(xj-z-((r-(0.5*(sig^2
)))*tao))^2))
    return(fx)
}

a <- log(K)
b <- zmax

EuroCalltrapezoid <- function(f, a, b, n, K, tao, z, sig, r) {
    if (is.function(f) == FALSE) {
```

```
        stop('f must be a function with one parameter (variable)')
    }
    h <- (b - a) / (n-1)
    j <- 1:(n - 1)
    xj <- a + j * h
    approx <- (h / 2) * (f(a, K, tao, z, sig, r) + 2*sum(f(xj, K, tao, z, sig, r)) + f(b, K, t
ao, z, sig, r))
    return(exp(-r*T)*approx)
}

EuroCallsimpson <- function(f, a, b, n, K, tao, z, sig, r) {
    if (is.function(f) == FALSE) {
        stop('f must be a function with one parameter (variable)')
    }
    h <- (b - a) / (n - 1)
    xj <- seq.int(a, b, length.out = n)
    xj <- xj[-1]
    xj <- xj[-length(xj)]
    approx <- (h / 3) * (f(a, K, tao, z, sig, r) + 2 * sum(f(xj[seq.int(2, length(xj), 2)], K,
tao, z, sig, r)) + 4 * sum(f(xj[seq.int(1, length(xj), 2)], K, tao, z, sig, r)) + f(b, K, tao,
z, sig, r))
    return(exp(-r*T)*approx)
}
BS <- function (K, S, t, r, v, type){

    d1 <- (log(S/K)+(r+(v^2/2)*t))/(v*sqrt(t))
    d2 <- d1 - v * sqrt(t)

    OptPrice <- NULL

    if(tolower(type) == "calls"){
        OptPrice <- S*pnorm(d1) - K*exp(-r*t)*pnorm(d2)
    }else{
        OptPrice <- K*exp(-r*t) * pnorm(-d2) - S*pnorm(-d1)
    }

    return(OptPrice)
}

table <- matrix(0, nrow = 7, ncol = 4)
j = 1
for(i in c(9,17,33,65,129,257,513)){
    table[j,1] <- i
    table[j,2] <- EuroCalltrapezoid(f, a, b, i, K,T, z, sig, r)
    table[j,3] <- EuroCallsimpson(f, a, b, i, K,T, z, sig, r)
    table[j,4] <- BS(K,S0,T,r,sig,type = 'calls')
    j = j + 1
}
table <- data.frame(table)
colnames(table) <- c("m", "Trapezoid", "Simpson", "BlackScholes")
table
```

Problem 2

(a) Why can Heston model produce an implied volatility curve (smile)? Give parameter choices to reduce Heston model to Geometric Brownian motion.

Heston model is the underlying stochastic volatility (SV) model which is a square-root diffusion model for stochastic-variance process. Hence, Heston model can produce an implied volatility curve.

When we set kappa equals to 0 and volatility of variance equals to zero, then:

$$dY_t = 0 * (\bar{Y} - Y_t)dt + 0 * \sqrt{Y_t}dZ_t = 0$$

$$\int_0^t dY_t = Y_t - Y_0 = 0;\ Y_t = Y_0$$

Since Y_0 is a constant, we assume and set it as σ^2. Then substitute Y_0,

$$\mathrm{dS}_t = r\mathrm{S}_t dt + \sqrt{\sigma^2}dZ_t$$

$$= r\mathrm{S}_t dt + \sigma dZ_t$$

From $\mathrm{dS}_t = r\mathrm{S}_t dt + \sigma dZ_t$, it is there Geometric Brownian Motion with μ equal to r.

(b) Apply expectation to volatility process Y_t to obtain $y_t = \mathbb{E}[Y_t]$ and approximate $y_{\frac{1}{250}}$.

Set

$$f(t, Y_t) = e^{\kappa t}Y_t, f_t = ke^{\kappa t}Y_t, f_x = e^{\kappa t}, f_{xx} = 0$$

$$\mathrm{d}(e^{\kappa t}Y_t) = \kappa e^{\kappa t}Y_t dt + e^{\kappa t}dY_t$$

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

$$= \kappa e^{\kappa t}Y_t dt + e^{\kappa t}\left[\kappa(\bar{Y} - Y_t)dt + \nu\sqrt{Y_t}dZ_t\right]$$

$$= \kappa e^{\kappa t}Y_t dt + e^{\kappa t}\kappa(\bar{Y} - Y_t)dt + e^{\kappa t}\nu\sqrt{Y_t}dZ_t$$

$$= \kappa e^{\kappa t}Y_t dt + e^{\kappa t}\kappa\bar{Y}dt - \kappa e^{\kappa t}Y_t dt + e^{\kappa t}\nu\sqrt{Y_t}dZ_t$$

$$= e^{\kappa t}\kappa\bar{Y}dt + e^{\kappa t}\nu\sqrt{Y_t}dZ_t$$

$$\int_0^t \mathrm{d}(e^{\kappa t}Y_t) = \int_0^t e^{\kappa t}\kappa\bar{Y}\,dt + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

$$e^{\kappa t}Y_t - e^0 Y_0 = \int_0^t e^{\kappa t}\kappa\bar{Y}\,dt + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

$$e^{\kappa t}Y_t = Y_0 + \int_0^t e^{\kappa t}\kappa\bar{Y}\,dt + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

$$Y_t = e^{-\kappa t}Y_0 + e^{-\kappa t}\kappa\bar{Y}\int_0^t e^{\kappa t}\,dt + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

$$Y_t = e^{-\kappa t}Y_0 + e^{-\kappa t}\kappa\bar{Y}\frac{1}{\kappa}[e^{\kappa t} - 1] + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

$$Y_t = e^{-\kappa t}Y_0 + e^{-\kappa t}\bar{Y}[e^{\kappa t} - 1] + \int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t$$

Take expectation to Y_t.

$$\mathbb{E}[Y_t] = \mathbb{E}[e^{-\kappa t}Y_0] + \bar{Y}[1 - e^{-\kappa t}] + \mathbb{E}\left[\int_0^t e^{\kappa t}\nu\sqrt{Y_t}\,dZ_t\right]$$

From martingale property, expectation to Ito integral equal to zero, so

$$\mathbb{E}[Y_t] = \mathbb{E}[e^{-\kappa t}Y_0] + \bar{Y}[1 - e^{-\kappa t}] + 0 = e^{-\kappa t}Y_0 + \bar{Y}[1 - e^{-\kappa t}]$$

From $r = 0.01, \kappa = 2, \bar{Y} = 0.16, \nu = 1, t = \frac{1}{250}, Y_0 = 0.17$

$$\mathbb{E}\left[Y_{\frac{1}{250}}\right] = e^{-\frac{2}{250}}(0.17) + 0.16\left[1 - e^{-\frac{2}{250}}\right] = 0.1699203$$

Therefore, a numerical answer to the value of $\mathbb{E}\left[Y_{\frac{1}{250}}\right] = y_{\frac{1}{250}} = 0.1699203$

Problem 3

We have to fix the discount term disc = exp(-r*dt), the initial St[0] = S*d^N and change max function to max(St[j] – K, 0.0)

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

(a) Implement the Explicit Finite Different method to price both European Call and Put options

Code:

```
ExplicitMethod <- function(isCall, K, Tao, S0, r, sig, N, div, dx, Nj = 0, Greeks = FALSE){

    dt <- Tao/N
    nu <- r-div-0.5*sig^2
    edx <- exp(dx)

    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)

    fRow <- 1
    fCol <- 1

    if(Nj != 0){
        Greeks = FALSE
    }

    cp <- ifelse(isCall, 1, -1)
    if(Nj!=0){
        r <- 2*Nj+1
        nRow <- 2*Nj+1
        lRow <- 2*Nj+1
        mRow <- Nj+1

        nCol <- N+1
        lCol <- N+1

        V <- matrix(0, nrow=nRow, ncol=nCol)
        S <- matrix(0, nrow=nRow, ncol=nCol)

        S[mRow, fCol] <- S0
        S[lRow,lCol] <- S0*exp(-Nj*dx)

        for(j in (lRow-1):1){
            S[j,lCol] <- S[j+1,lCol] * edx
        }
    }
    else{
        Nj <- N
        r <- 2*Nj+1
        nRow <- 2*Nj+1
        lRow <- 2*Nj+1

        mRow <- N+1
        s <- N+1
        nCol <- N+1
        lCol <- N+1

        V <- matrix(0, nrow=nRow, ncol=nCol)
        S <- matrix(0, nrow=nRow, ncol=nCol)

        S[mRow, fCol] <- S0
```

```
        for (i in 1:(nCol-1)) {
            for(j in (mRow-i+1):(mRow+i-1)) {
                S[j-1,i+1] <- S[j, i]*exp(dx)
                S[j,i+1] <- S[j,i]
                S[j+1,i+1] <- S[j,i]*exp(-dx)
            }
        }
    }

    for (j in 1:lRow) {
        V[j,lCol] <- max(0, cp * (S[j, lCol]-K))
    }

    for (i in N:1) {
        for(j in (mRow+Nj-1):(mRow-Nj+1)) {
            V[j, i] <- pu*V[j-1,i+1] + pm*V[j, i+1] + pd*V[j+1,i+1]
        }
        boundary <- ifelse(isCall, S[1, lCol]-S[2,lCol], S[nRow-1,lCol]-S[nRow,lCol])
        V[lRow,  i] <- V[lRow-1,  i] + ifelse(isCall, 0, boundary)
        V[fRow, i] <- V[fRow+1, i] + ifelse(isCall, boundary, 0)
    }

    if(Greeks && isCall){
        delta <- (V[mRow-1,fCol+1]-V[mRow+1,fCol+1])/
            (S[mRow-1,fCol+1] - S[mRow+1,fCol+1])
        deltaf =(V[mRow-1,fCol+1]-V[mRow,fCol+1])/
            (S[mRow-1,fCol+1]-S[mRow,fCol+1])
        deltas =(V[mRow,fCol+1]-V[mRow+1,fCol+1])/
            (S[mRow,fCol+1]-S[mRow+1,fCol+1])
        gamma <- 2*(deltaf-deltas)/((S[mRow-1,fCol+1] - S[mRow+1,fCol+1]))
        theta <-((V[mRow,fCol+1] - V[mRow,fCol])/dt)

        return(list(Price = V[mRow,fCol], Delta = delta, Gamma = gamma, Theta = theta))
    }

    return(V[mRow,fCol])
}
```

(b) Implement the Implicit Finite Different method to price both European Call and Put options

Code:

```
ImplicitMethod <- function(isCall, K, Tao,S0, r, sig, N, div, dx, Nj=0){

    dt <- Tao/N
    nu <- r-div-0.5*sig^2
    edx <- exp(dx)

    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)
    fRow <- 1

    if(Nj!=0){
        r <- 2*Nj+1
        nRow <- 2*Nj+1
        lRow <- 2*Nj+1
        mRow <- Nj+1
```

```
        nCol <- lCol = N+1
    }
    else{
        Nj <- N
        r <- 2*Nj+1
        nRow <- 2*Nj+1
        lRow <- 2*Nj+1

        mRow <- N+1
        s <- N+1
        nCol <- N+1
        lCol <- N+1
    }
    fCol <- 1
    cp = ifelse(isCall, 1, -1)

    pp <- matrix(0, nrow=nRow, ncol=nCol)
    pmp <- matrix(0, nrow=nRow, ncol=nCol)
    V <- matrix(0, nrow=nRow, ncol=nCol)
    S <- matrix(0, nrow=nRow, ncol=nCol)

    S[mRow, fCol] <- S0
    S[lRow,lCol] <- S0*exp(-Nj*dx)

    for(j in (lRow-1):1){
        S[j,lCol] <- S[j+1,lCol] * edx
    }
    for (j in fRow:lRow) {
        V[j, lCol] <- max( 0, cp * (S[j, lCol]-K))
    }
    if(isCall){
        lambdaL <- 0
        lambdaU <-(S[1, lCol] - S[2, lCol])
    }else{
        lambdaL <- -1 * (S[lRow-1, lCol] - S[lRow,lCol])
        lambdaU <- 0
    }
    for (i in (lCol-1):fCol) {
        V <- solveImplicitTridiagonalSystem(V, pu, pm, pd, lambdaL, lambdaU, i)
    }
    return(V[mRow,fCol])
}

solveImplicitTridiagonalSystem <- function(V, pu, pm, pd, lambdaL, lambdaU, iCol)
{
    fRow <- 1
    sRow <- 2
    tRow <- 3
    lRow <- nrow(V)
    nRow <- nrow(V)
    lCol <- ncol(V)

    pp <- numeric(nRow)
    pmp <- numeric(nRow)
    pmp[lRow-1] <- pm + pd
    pp[lRow-1]  <- V[lRow-1, lCol] + pd*lambdaL

    for (j in (lRow-2):(sRow)) {
        pmp[j] <- pm - pu*pd/pmp[j+1]
        pp[j] <- V[j, iCol+1] - pp[j+1]*pd/pmp[j+1]
```

```
    }

    V[fRow, iCol] <- (pp[sRow] + pmp[sRow]*lambdaU)/(pu + pmp[sRow])
    V[sRow, iCol] <- V[fRow,iCol] - lambdaU

    for(j in tRow:lRow) {
        V[j, iCol] <-  (pp[j] -pu*V[j-1, iCol])/pmp[j]
    }
    V[lRow, iCol] <- V[lRow-1, iCol] - lambdaL

    return(V)
}
```

(c) Estimate the numbers $\Delta t, \Delta x, Nj$ for both Explicit and Implicit Finite Difference schemes.

Using the accuracy of Explicit Finite Difference schemes which is $O(\Delta x^2 + \Delta t)$, we will increase the number of N in formula $\Delta t = \frac{T}{N}$ and put in the equations $N_j \leq \frac{n_{sd}\sqrt{T}}{2\sqrt{3\Delta t}} - \frac{1}{2}$ and $\Delta x = \frac{n_{sd}\sigma\sqrt{T}}{2N_j+1}$, respectively. Run the loop until $\Delta x^2 + \Delta t \leq \varepsilon$.

```
## $dt
## [1] 0.0008410429
##
## $Nj
## [1] 59
##
## $dx
## [1] 0.01260504
##
## $N
## [1] 1189
```

Code:

```
parametersEstimate <- function(N=3, sig, Tao, nsd, epsilon = 0.001){
    while(T){
        dt <- Tao/N
        Nj <- (sqrt(N)*nsd)/(2*sqrt(3))-0.5
        Nj <- floor(Nj)
        dx <- (nsd*sig*sqrt(Tao))/(2*Nj+1)

        if(dx^2+dt <= epsilon){
            break
        }
        N <- N+1
    }

    return(list(dt = dt, Nj = Nj,dx = dx,N = N))
}

param <- parametersEstimate(N=2,sig=0.25,Tao=1,nsd=6)
param
```

(d) Consider $S_0 = 100, K = 100, T = 1, \sigma = 25\%, r = 6\%, \delta = 0.03$. Report the prices.

The difference between Explicit Finite Difference and Implicit Finite Difference methods start at 0.0X for Call Option but 0.00X for Put Option as pictures shown before.

```
##   EFD.call  EFD.put
## 1 11.01139 8.143147

##   IFD.call  IFD.put
## 1 11.00896 8.140925
```

Code:

```
EFD <- data.frame(EFD.call = ExplicitMethod(isCall=T, K=100, Tao=1, S0=100, r=0.06, sig=0.25,
N=param$N, div=0.03, dx=param$dx,Greeks = F),
                  EFD.put = ExplicitMethod(isCall=F, K=100, Tao=1, S0=100, r=0.06, sig=0.25, N
=param$N, div=0.03, dx=param$dx,Greeks = F))

IFD <- data.frame(IFD.call = ImplicitMethod(isCall=T, K=100, Tao=1, S0=100, r=0.06, sig=0.25,
N=param$N, div=0.03, dx=param$dx),
                  IFD.put = ImplicitMethod(isCall=F, K=100, Tao=1, S0=100, r=0.06, sig=0.25, N
=param$N, div=0.03, dx=param$dx))
```

(e) Repeat part (c) to get the empirical number of iterations. Obtain Black-Scholes price and do an iterative procedure to figure out $\Delta t, \Delta x, N, Nj$.

Using the condition that the difference between Black-Scholes and Finite Difference methods of Call Option to find the parameters that make Finite Difference price converge to Black-Scholes price, we obtain as follows:

- Explicit: dt = 0.0005170631, Nj = 75.67086, dx = 0.009846285, N = 1934
- Implicit: dt = 0.0002157963, Nj = 117.4067, dx = 0.006360959, N = 4634

It's clear that we need to spend a lot of number to make FD method converge to Black-Scholes.

```
parametersEstimateBS(N=1932, K=100, S0=100, r=0.06, sig=0.25, Tao=1, div=0, nsd=6, explicit =
T)

## $dt
## [1] 0.0005170631
##
## $Nj
## [1] 75.67086
##
## $dx
## [1] 0.009846285
##
## $N
## [1] 1934

parametersEstimateBS(N=4634, K=100, S0=100, r=0.06, sig=0.25, Tao=1, div=0, nsd=6, explicit =
F)

## $dt
## [1] 0.0002157963
##
## $Nj
## [1] 117.4067
##
```

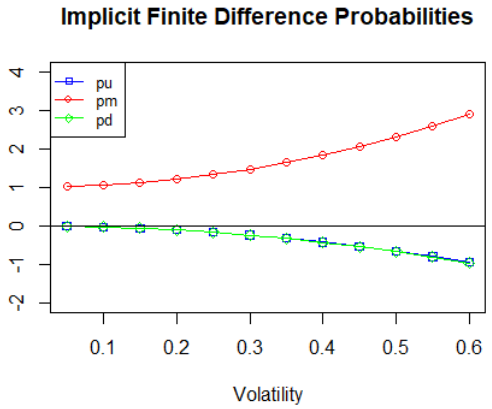
```
## $dx
## [1] 0.006360959
##
## $N
## [1] 4634
```

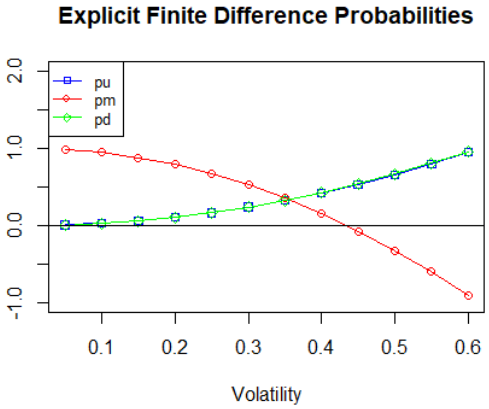
Code:

```
parametersEstimateBS <- function(N, K, S0, r, sig, Tao, div, nsd, explicit = T, epsilon = 0.00
1){

    BS <- BSM(K, S0, Tao, r, sig, type="calls")

    while(T){
        dt <- Tao/N
        Nj <- (sqrt(N)*nsd)/(2*sqrt(3))-0.5
        dx <- (nsd*sig*sqrt(Tao))/(2*Nj+1)

        if(explicit == F){
            IFD.price <- ImplicitMethod(isCall=T, K=K, Tao=Tao, S0=S0, r=r, sig=sig, N=N, div=
div, dx=dx)
            if(abs(BS - IFD.price) <= epsilon){
                break
            }
        }else{
            EFD.price <- ExplicitMethod(isCall=T, K=K, Tao=Tao, S0=S0, r=r, sig=sig, N=N, div=
div, dx=dx,Greeks = F)
            if(abs(BS - EFD.price) <= epsilon){
                break
            }
        }
        N <- N+1
    }
    return(list(dt = dt, Nj = Nj,dx = dx,N = N))
}
BSM <- function (K, S, t, r, v, type){

    d1 <- (log(S/K)+(r+(v^2/2)*t))/(v*sqrt(t))
    d2 <- d1 - v * sqrt(t)

    OptPrice <- NULL

    if(tolower(type) == "calls"){
        OptPrice <- S*pnorm(d1) - K*exp(-r*t)*pnorm(d2)
    }else{
        OptPrice <- K*exp(-r*t) * pnorm(-d2) - S*pnorm(-d1)
    }

    return(OptPrice)
}
```

(f) Plot the graph for the Implicit Finite Difference Probabilities P_u, P_m, P_d as a function of $\sigma, \sigma \in \{0.05, 0.1, 0.15, \ldots, 0.6\}$.

With parameters from part (d), P_u *and* P_d are moving down and getting very close to each other but P_m is moving up when the volatility is increasing. It's obvious that P_u, P_m, P_d of Implicit method have opposite direction to Explicit method.





Code:

```
ImplicitProb <- function(sig, dt=0.0008928571, dx=0.01035098, r=0.06, div=0.03){
    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)

    return(list(pu = pu, pm = pm , pd = pd))
}
x = seq(from=0.05, to=0.6, by=0.05)
probs <- matrix(0, nrow = length(x), ncol = 3)
for(i in 1:length(x)){
    prob <- ImplicitProb(x[i])
    probs[i,1] <- prob$pu
    probs[i,2] <- prob$pm
    probs[i,3] <- prob$pd
}
probs <- data.frame(pu = probs[,1], pm = probs[,2], pd = probs[,3])
plot(x,probs$pu, type = "o", col="blue", ylim = c(-2.00001, 4),
     xlab = "Volatility", ylab = "", pch=22, main = "Implicit Finite Difference Probabilities"
)
lines(x,probs$pm, col="red", type = "o", pch=21)
lines(x,probs$pd, col="green", type = "o", pch=23)
legend("topleft", legend=c("pu", "pm", "pd"),
       col=c("blue", "red", "green"), lty=1, pch = c(22,21,23), cex=0.8)
abline(h=0)

ExplicitProb <- function(sig, dt=param$dt, dx=param$dx, r=0.06, div=0.03){
    nu <- r-div-0.5*sig^2

    pu <- 0.5*dt*((sig/dx)^2 + nu/dx )
    pm <- 1.0-dt*(sig/dx)^2
    pd <- 0.5*dt*((sig/dx)^2 - nu/dx)
```

```
    return(list(pu = pu, pm = pm , pd = pd))
}
x = seq(from=0.05, to=0.6, by=0.05)

probs2 <- matrix(0, nrow = length(x), ncol = 3)

for(i in 1:length(x)){
    prob2 <- ExplicitProb(x[i])
    probs2[i,1] <- prob2$pu
    probs2[i,2] <- prob2$pm
    probs2[i,3] <- prob2$pd
}
probs2 <- data.frame(pu = probs2[,1], pm = probs2[,2], pd = probs2[,3])

plot(x,probs2$pu, type = "o", col="blue", ylim = c(-1, 2),
     xlab = "Volatility", ylab = "", pch=22, main = "Explicit Finite Difference Probabilities"
)
lines(x,probs2$pm, col="red", type = "o", pch=21)
lines(x,probs2$pd, col="green", type = "o", pch=23)
legend("topleft", legend=c("pu", "pm", "pd"),
       col=c("blue", "red", "green"), lty=1, pch = c(22,21,23), cex=0.8)
abline(h=0)
```

(g) Implement the Crank-Nicolson Finite Difference method for European Call and Put Options. Put the results of the 3 methods (EFD, IFD, CNFD).

The prices from 3 methods are very close. The difference between Explicit Finite Difference and Implicit Finite Difference methods start at 0.0X for Call Option but 0.00X for Put Option as pictures shown before. As we can see, it's obvious that the CNFD is the average between EFD and IFD.

```
##            EFD       IFD      CNFD
## call 11.011390 11.008956 11.010173
## put   8.143147  8.140925  8.142036
```

Code:

```
CrankNicholsonMethod <- function(isCall, K, tao, S0, r, sig, N, div, dx, Nj=0){

    dt <- tao/N
    nu <- r-div-0.5*sig^2
    edx <- exp(dx)
    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)
    fRow = 1
    fCol = 1
    if(Nj!=0){
        r <- nRow <- lRow <- 2*Nj+1
        mRow <- Nj+1
        nCol <- lCol <- N+1
    }
    else{
        Nj <- N
        r <- nRow <- lRow <- 2*Nj+1
        mRow <- s <- nCols <- lCol <- N+1
    }
```

```
    mRow <- Nj+1
    nCol <- Nj+1
    lCol <- Nj+1

    cp <- ifelse(isCall, 1, -1)

    pp <- matrix(0, nrow=nRow, ncol=nCol)
    pmp <- matrix(0, nrow=nRow, ncol=nCol)
    V <- matrix(0, nrow=nRow, ncol=nCol)
    S <- matrix(0, nrow=nRow, ncol=nCol)

    S[mRow, fCol] <- S0
    S[lRow,lCol] <- S0*exp(-Nj*dx)

    for(j in (lRow-1):1){
        S[j,lCol] <- S[j+1,lCol]*edx
    }
    for (j in fRow:lRow) {
        V[j, lCol] <- max(0,cp*(S[j,lCol]-K))
    }
    if(isCall){
        lambdaL <- 0
        lambdaU <- (S[1,lCol]-S[2,lCol])
    }else{
        lambdaL <- -1 * (S[lRow-1, lCol]-S[lRow,lCol])
        lambdaU <- 0
    }
    for (i in (lCol-1):fCol) {

        V <- solveCrankNicholsonTridiagonalSystem(V, pu, pm, pd, lambdaL, lambdaU, i)
    }
    return(V[mRow,fCol])
}

solveCrankNicholsonTridiagonalSystem <- function(V, pu, pm, pd, lambdaL, lambdaU, iCol)
{
    fRow <- 1
    sRow <- 2
    tRow <- 3
    lRow <- nrow(V)
    nRow <- nrow(V)
    lCol <- ncol(V)

    pp <- numeric(nRow)
    pmp <- numeric(nRow)
    pmp[lRow-1] <- pm + pd
    pp[lRow-1]  <- (- pu   *V[lRow-2, lCol]-(pm-2)*V[lRow-1, lCol]
                    - pd   *V[lRow  , lCol] + pd*lambdaL)
    for (j in (lRow-2):(sRow)) {
        pmp[j] <- pm - pu*pd/pmp[j+1]
        pp[j] <- ( - pu   *V[j-1, iCol+1]-(pm-2) *V[j  , iCol+1]
                   - pd    *V[j+1, iCol+1]-pp[j+1]*pd/pmp[j+1])
    }
    V[fRow, iCol] <- (pp[sRow] + pmp[sRow]*lambdaU)/(pu + pmp[sRow])
    V[sRow, iCol] <- V[fRow,iCol] - lambdaU
    for(j in tRow:lRow) {
        V[j, iCol] <-  (pp[j] -pu*V[j-1, iCol])/pmp[j]
    }
    V[lRow, iCol] <- V[lRow-1, iCol] - lambdaL
```

```r
    return(V)
}

CNFD <- data.frame(CNFD.call = CrankNicholsonMethod(isCall=T, K=100, tao=1, S0=100, r=0.06, si
g=0.25, N=param$N, div=0.03, dx=param$dx),
                   CNFD.put = CrankNicholsonMethod(isCall=F, K=100, tao=1, S0=100, r=0.06, sig
=0.25, N=param$N, div=0.03, dx=param$dx))

FD.table <- cbind(cbind(t(EFD), t(IFD)),t(CNFD))
colnames(FD.table) <- c("EFD", "IFD", "CNFD")
row.names(FD.table) <- c("call", "put")
FD.table
```

(h) Calculate the hedge sensitivities for European Call Option using Explicit Finite Difference method.

Using same parameters from (d), we have that Delta = 0.5791419, Gamma = 0.01503321, Theta = -5.774924 and Vega = 37.58323.

```
HedgeSensitivities(isCall=TRUE, K=100, Tao=1, S0=100, r=0.06, sig=0.25, N=param$N, div=0.03, d
x=param$dx,Greeks = T)

## $delta
## [1] 0.5791529
##
## $gamma
## [1] 0.01503452
##
## $theta
## [1] -5.775438
##
## $vega
## [1] 37.5866
```

Code:

```
HedgeSensitivities <- function(isCall, K, Tao, S0, r, sig, N, div, dx, Nj = 0, Greeks = TRUE){

    Greeks <- ExplicitMethod(isCall=TRUE, K=K, Tao=Tao, S0=S0, r=r, sig=sig, N=N, div=div, dx=
dx,Greeks = Greeks)
    delta <- Greeks$delta
    gamma <- Greeks$gamma
    theta <- Greeks$theta

    dsig <- 0.001*sig
    vega <- (ExplicitMethod(isCall=TRUE, K=K, Tao=Tao, S0=S0, r=r, sig=sig+dsig, N=N, div=div,
dx=dx,Greeks = F) - ExplicitMethod(isCall=TRUE, K=K, Tao=Tao, S0=S0, r=r, sig=sig-dsig, N=N, d
iv=div, dx=dx,Greeks = F))/(2*dsig)

    return(list(delta = delta, gamma = gamma, theta = theta, vega = vega))
}
```

Problem 5

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

1) Discretize the derivatives and give the finite difference equation for Explicit schema.

From the result of Taylor's series,

$$\frac{\partial V}{\partial S}(S,t) = \frac{V(S+\Delta S,t) - V(S-\Delta S,t)}{\Delta S} + O((\Delta S)^2) \approx \frac{V_{j+1}^{i+1} - V_{j-1}^{i+1}}{2\Delta S}$$

$$\frac{\partial^2 V}{\partial S^2}(S,t) = \frac{V(S+\Delta S,t) - 2V(S,t) + V(S-\Delta S,t)}{(\Delta S)^2} + O((\Delta S)^2) \approx \frac{V_{j+1}^{i+1} - 2V_j^{i+1} + V_{j-1}^{i+1}}{(\Delta S)^2}$$

$$\frac{\partial V}{\partial t}(S,t) = \frac{V(S,t+\Delta t) - V(S,t)}{\Delta t} + O(\Delta t) \approx \frac{V_j^{i+1} - V_j^i}{\Delta t}$$

Substitute in the question's equation.

$$\frac{V_j^{i+1} - V_j^i}{\Delta t} + 2cos(j\Delta S)\left[\frac{V_{j+1}^{i+1} - V_{j-1}^{i+1}}{2\Delta S}\right] + 0.2(j\Delta S)^{\frac{3}{2}}\left[\frac{V_{j+1}^{i+1} - 2V_j^{i+1} + V_{j-1}^{i+1}}{(\Delta S)^2}\right] - rV_j^i = 0$$

$$\frac{V_j^{i+1} - V_j^i}{\Delta t} + cos(j\Delta S)\left[\frac{V_{j+1}^{i+1} - V_{j-1}^{i+1}}{\Delta S}\right] + 0.2(j\Delta S)^{\frac{3}{2}}\left[\frac{V_{j+1}^{i+1} - 2V_j^{i+1} + V_{j-1}^{i+1}}{(\Delta S)^2}\right] = rV_j^i$$

$$\frac{V_j^{i+1}}{\Delta t} - \frac{V_j^i}{\Delta t} + \frac{cos(j\Delta S)V_{j+1}^{i+1}}{\Delta S} - \frac{cos(j\Delta S)V_{j-1}^{i+1}}{\Delta S} + \frac{0.2(j\Delta S)^{\frac{3}{2}}V_{j+1}^{i+1}}{(\Delta S)^2} - \frac{0.4(j\Delta S)^{\frac{3}{2}}V_j^{i+1}}{(\Delta S)^2} + \frac{0.2(j\Delta S)^{\frac{3}{2}}V_{j-1}^{i+1}}{(\Delta S)^2} = rV_j^i$$

$$\frac{V_j^{i+1}}{\Delta t} - \frac{0.4(j\Delta S)^{\frac{3}{2}}V_j^{i+1}}{(\Delta S)^2} + \frac{cos(j\Delta S)V_{j+1}^{i+1}}{\Delta S} + \frac{0.2(j\Delta S)^{\frac{3}{2}}V_{j+1}^{i+1}}{(\Delta S)^2} - \frac{cos(j\Delta S)V_{j-1}^{i+1}}{\Delta S} + \frac{0.2(j\Delta S)^{\frac{3}{2}}V_{j-1}^{i+1}}{(\Delta S)^2} = rV_j^i + \frac{V_j^i}{\Delta t}$$

$$\left[\frac{1}{\Delta t} - \frac{0.4(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_j^{i+1} + \left[\frac{cos(j\Delta S)}{\Delta S} + \frac{0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j+1}^{i+1} + \left[-\frac{cos(j\Delta S)}{\Delta S} + \frac{0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j-1}^{i+1} = \left[r + \frac{1}{\Delta t}\right]V_j^i$$

$$\left[\frac{(\Delta S)^2}{\Delta t(\Delta S)^2} - \frac{0.4\Delta t(j\Delta S)^{\frac{3}{2}}}{\Delta t(\Delta S)^2}\right]V_j^{i+1} + \left[\frac{cos(j\Delta S)(\Delta S)}{(\Delta S)^2} + \frac{0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j+1}^{i+1} + \left[-\frac{cos(j\Delta S)(\Delta S)}{(\Delta S)^2} + \frac{0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j-1}^{i+1}$$
$$= \left[\frac{1 + r\Delta t}{\Delta t}\right]V_j^i$$

$$\left[\frac{cos(j\Delta S)(\Delta S) + 0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j+1}^{i+1} + \left[\frac{(\Delta S)^2 - 0.4\Delta t(j\Delta S)^{\frac{3}{2}}}{\Delta t(\Delta S)^2}\right]V_j^{i+1} + \left[\frac{-cos(jS)(\Delta S) + 0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}\right]V_{j-1}^{i+1}$$
$$= \left[\frac{1 + r\Delta t}{\Delta t}\right]V_j^i$$

$$V_j^i$$
$$= \frac{1}{1+r\Delta t}\left[\Delta t\frac{cos(j\Delta S)(\Delta S)+0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}V_{j+1}^{i+1}+\Delta t\frac{(\Delta S)^2-0.4\Delta t(j\Delta S)^{\frac{3}{2}}}{\Delta t(\Delta S)^2}V_j^{i+1}\right.$$
$$\left.+\frac{-cos(j\Delta S)(\Delta S)+0.2(j\Delta S)^{\frac{3}{2}}}{(\Delta S)^2}V_{j-1}^{i+1}\right]$$

$$V_j^i = \frac{1}{1+r\Delta t}\left[a_jV_{j+1}^{i+1}+b_jV_j^{i+1}+c_jV_{j-1}^{i+1}\right]$$

$$a_j = \frac{cos(j\Delta S)+0.2(j)^{\frac{3}{2}}(\Delta S)^{\frac{1}{2}}}{\Delta S}\Delta t, b_j = 1-0.4\Delta t(j)^{\frac{3}{2}}(\Delta S)^{-\frac{1}{2}}, c_j = \frac{-cos(j\Delta S)+0.2(j)^{\frac{3}{2}}(\Delta S)^{\frac{1}{2}}}{\Delta S}\Delta t$$

2) What do you observe about updating coefficients?

For question PDE, we have coefficients as below.

$$a_j = \frac{cos(j\Delta S)+0.2(j)^{\frac{3}{2}}(\Delta S)^{\frac{1}{2}}}{\Delta S}\Delta t, b_j = 1-0.4\Delta t(j)^{\frac{3}{2}}(\Delta S)^{-\frac{1}{2}}, c_j = \frac{-cos(j\Delta S)+0.2(j)^{\frac{3}{2}}(\Delta S)^{\frac{1}{2}}}{\Delta S}\Delta t$$

As we can see, the coefficients of question PDE is variable since there are a lot of j term in the equations. Hence, the difference of coefficients between Black-Scholes PDE and question PDE is that Black-Scholes is constant coefficients but question PDE is variable coefficients which change along the grid.

3) Can you use an SOR scheme instead of Explicit equation?

We use the Explicit equation in case that there are multiple known but there is one unknown. However, for SOR scheme for the Implicit equation, there are multiple unknown. We therefore can use the iterative methods with matrix decomposition technique to solve this problem. One of them is SOR (Successive Over Relaxation). Moreover, SOR can be adapted to both European Option and American Option.

From a_j, b_j, c_j, in previous question and $d_j = -a_jV_{j+1}^{i+1}-b_jV_j^{i+1}-d_jV_{j-1}^{i+1}$, the SOR algorithm is as below.

$$y_j^{i,k+1} = \frac{1}{b_j}\left(d_j^i - a_jV_{j-1}^{i+1}-c_jV_{j+1}^{i+1}\right)$$

$$V_j^{i,k+1} = V_j^{i,k}+\omega\left(y_j^{i,k+1}+V_j^{i,k}\right), where\ \omega\ \epsilon\ (1,2)$$

The continuation value is:

$$CoV_j^i = \frac{1}{1+r\Delta t}\left[a_jV_{j+1}^{i+1}+b_jV_j^{i+1}+c_jV_{j-1}^{i+1}\right]$$

When we apply to American Option, then;

$V_j^i = max\left(j\Delta S-K, CoV_j^i\right)$ for Call Option and $V_j^i = max\left(K-j\Delta S, CoV_j^i\right)$ for Put Option.

Hence, there is no problem to use SOR over Explicit equation.