FE621. Assignment #3.

2019-10-28

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.

Name: Tai Nguyen Cong

CWID: 10442353

Question 1

Question 1.1

Solution:

```
# Explicit Finite Difference Method
# EU Call
explicitFDM.EC <- function(s0, k, t, r, div, sig, delx, N) {
  delt = t/N
  v = r - div - 0.5*sig^2

  pu = 0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1 - delt*(sig/delx)^2 -r*delt
  pd = 0.5*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  EC.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EC.res[1, i] = max(c(0, stock.pr.fin[i] - k))
  }

  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EC.res[i, j] = pd*EC.res[i-1, j-1] + pm*EC.res[i-1, j] + pu*EC.res[i-1, j+1]
      EC.res[i, 1] = EC.res[i, 2]
      EC.res[i, 2*N+1] = EC.res[i, 2*N] + stock.pr.fin[length(stock.pr.fin)] -
        stock.pr.fin[length(stock.pr.fin)-1]
    }
  }
  return(EC.res[N+1, ceiling((2*N+1)/2)])
}

# EU Put
explicitFDM.EP <- function(s0, k, t, r, div, sig, delx, N) {
  delt = t/N
  v = r - div - 0.5*sig^2

  pu = 0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1 - delt*(sig/delx)^2 -r*delt
  pd = 0.5*delt*((sig/delx)^2 - v/delx)
```

```
  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  EP.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EP.res[1, i] = max(c(0, k - stock.pr.fin[i]))
  }

  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EP.res[i, j] = pd*EP.res[i-1, j-1] + pm*EP.res[i-1, j] + pu*EP.res[i-1, j+1]
      EP.res[i, 2*N+1] = EP.res[i, 2*N]
      EP.res[i, 1] = EP.res[i, 2] + stock.pr.fin[2] - stock.pr.fin[1]
    }
  }
  return(EP.res[N+1, ceiling((2*N+1)/2)])
}
```

Question 1.2

Solution:

```
# Implicit Finite Difference Method
# Eu Call
implicitFDM.EC<-function(s0, k, t, r, div, sig, delx, N){
  delt = t/N
  v = r - div - 0.5*sig^2
  edx = exp(delx)

  pu = -0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1+delt*(sig/delx)^2 + r*delt
  pd = -0.5*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  lambda.u.c.eu = stock.pr.fin[length(stock.pr.fin)] - stock.pr.fin[length(stock.pr.fin)-1]
  lambda.l.c.eu = 0

  trid.mat <- matrix(ncol=2*N+1,nrow=2*N+1)
  trid.mat[1, ] = c(1,-1,rep(0,2*N-1))
  for(i in 2:(2*N)){
    trid.mat[i, ] = c(rep(0,i-2), pu, pm, pd, rep(0,2*N-i))
  }
  trid.mat[2*N+1, ] = c(rep(0,2*N-1), 1, -1)
  trid.mat = solve(trid.mat)

  EC.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EC.res[1, i] = max(c(0, stock.pr.fin[i] - k))
  }
  EC.res[1, ] = rev(EC.res[1, ])
```

```
  for(i in 2:(N+1)) {
    EC.res[i, ] = as.vector(trid.mat%*%
                            as.matrix( c(lambda.u.c.eu,
                                         EC.res[i-1, ][-c(1, length(EC.res[i-1, ]))],
                                         lambda.l.c.eu), ncol = 1))
  }
  return(EC.res[N+1, ceiling((2*N+1)/2)])
}

# Eu Put
implicitFDM.EP<-function(s0, k, t, r, div, sig, delx, N){
  delt = t/N
  v = r - div - 0.5*sig^2
  edx = exp(delx)

  pu = -0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1+delt*(sig/delx)^2 + r*delt
  pd = -0.5*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  lambda.l.c.eu = -1*(stock.pr.fin[2] - stock.pr.fin[1])
  lambda.u.c.eu = 0

  trid.mat <- matrix(ncol=2*N+1,nrow=2*N+1)
  trid.mat[1, ] = c(1,-1,rep(0,2*N-1))
  for(i in 2:(2*N)){
    trid.mat[i, ] = c(rep(0,i-2), pu, pm, pd, rep(0,2*N-i))
  }
  trid.mat[2*N+1, ] = c(rep(0,2*N-1), 1, -1)
  trid.mat = solve(trid.mat)

  EP.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EP.res[1, i] = max(c(0, k - stock.pr.fin[i]))
  }
  EP.res[1, ] = rev(EP.res[1, ])

  for(i in 2:(N+1)) {
    EP.res[i, ] = as.vector(trid.mat%*%
                            as.matrix( c(lambda.u.c.eu,
                                         EP.res[i-1, ][-c(1, length(EP.res[i-1, ]))],
                                         lambda.l.c.eu), ncol = 1))
  }
  return(EP.res[N+1, ceiling((2*N+1)/2)])
}
```

Question 1.3

Solution:

```r
# Crank - Nicolson Finite Difference Method
# EU Call
CN.FDM.EC<-function(s0, k, t, r, div, sig, delx, N){
  delt = t/N
  v = r - div - 0.5*sig^2
  edx = exp(delx)

  pu = -0.25*delt*((sig/delx)^2 + v/delx)
  pm = 1 + delt*0.5*(sig/delx)^2 + 0.5*r*delt
  pd = -0.25*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  lambda.u.c.eu = stock.pr.fin[length(stock.pr.fin)] - stock.pr.fin[length(stock.pr.fin)-1]
  lambda.l.c.eu = 0

  trid.mat <- matrix(ncol=2*N+1,nrow=2*N+1)
  trid.mat[1, ] = c(1,-1,rep(0,2*N-1))
  for(i in 2:(2*N)){
    trid.mat[i, ] = c(rep(0,i-2), pu, pm, pd, rep(0,2*N-i))
  }
  trid.mat[2*N+1, ] = c(rep(0,2*N-1), 1, -1)
  trid.mat = solve(trid.mat)

  EC.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EC.res[1, i] = max(c(0, stock.pr.fin[i] - k))
  }
  EC.res[1, ] = rev(EC.res[1, ])

  EC.mid.mat <- matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EC.mid.mat[i, j] = -1*pu*EC.res[i-1, j-1] - (pm-2)*EC.res[i-1, j] - pd*EC.res[i-1, j+1]
    }
    EC.res[i, ] = as.vector(trid.mat%*%
                              as.matrix( c(lambda.u.c.eu,
                                           EC.mid.mat[i, ][-c(1, length(EC.mid.mat[i, ]))],
                                           lambda.l.c.eu), ncol = 1))
  }
  return(EC.res[N+1, ceiling((2*N+1)/2)])
}

# EU Put
CN.FDM.EP<-function(s0, k, t, r, div, sig, delx, N){
  delt = t/N
  v = r - div - 0.5*sig^2
  edx = exp(delx)

  pu = -0.25*delt*((sig/delx)^2 + v/delx)
  pm = 1 + delt*0.5*(sig/delx)^2 + 0.5*r*delt
  pd = -0.25*delt*((sig/delx)^2 - v/delx)
```

```
  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  lambda.l.c.eu = -1*(stock.pr.fin[2] - stock.pr.fin[1])
  lambda.u.c.eu = 0

  trid.mat <- matrix(ncol=2*N+1,nrow=2*N+1)
  trid.mat[1, ] = c(1,-1,rep(0,2*N-1))
  for(i in 2:(2*N)){
    trid.mat[i, ] = c(rep(0,i-2), pu, pm, pd, rep(0,2*N-i))
  }
  trid.mat[2*N+1, ] = c(rep(0,2*N-1), 1, -1)
  trid.mat = solve(trid.mat)

  EP.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EP.res[1, i] = max(c(0, k - stock.pr.fin[i]))
  }
  EP.res[1, ] = rev(EP.res[1, ])

  EP.mid.mat <- matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EP.mid.mat[i, j] = -1*pu*EP.res[i-1, j-1] - (pm-2)*EP.res[i-1, j] - pd*EP.res[i-1, j+1]
    }
    EP.res[i, ] = as.vector(trid.mat%*%
                  as.matrix( c(lambda.u.c.eu, EP.mid.mat[i, ][-c(1, length(EP.mid.mat[i, ]))],
                                              lambda.l.c.eu), ncol = 1))
  }
  return(EP.res[N+1, ceiling((2*N+1)/2)])
}
```

Question 1.4

Solution:

For all 3 methods, I choose 675 time steps for calculation, as it has been proved in the book that this should be the number of steps to choose.

```
# Importing Data
s0 = 100
k = 100
t = 1
sig = 0.25
r = 0.06
div = 0.03
```

```
# Applying Explicit FDM
call.explicit = explicitFDM.EC(s0, k, t, r, div, sig, 0.2, 675)
put.explicit = explicitFDM.EP(s0, k, t, r, div, sig, 0.2, 3)
# Applying Implicit FDM
call.implicit = implicitFDM.EC(s0, k, t, r, div, sig, 0.2, 675)
put.implicit = implicitFDM.EP(s0, k, t, r, div, sig, 0.2, 675)
# Applying Crank-Nicolson FDM
```

```r
call.CN = CN.FDM.EC(s0, k, t, r, div, sig, 0.2, 675)
put.CN = CN.FDM.EP(s0, k, t, r, div, sig, 0.2, 675)

# Result table
call.tab.1 = matrix(c(call.explicit, call.implicit, call.CN), nrow = 1)
colnames(call.tab.1) <- c("Explicit FDM", "Implicit FDM", "Crank-Nicolson FDM")
put.tab.1 = matrix(c(put.explicit, put.implicit, put.CN), nrow = 1)
colnames(put.tab.1) <- c("Explicit FDM", "Implicit FDM", "Crank-Nicolson FDM")
call.tab.1
```

```
##      Explicit FDM Implicit FDM Crank-Nicolson FDM
## [1,]     10.14265     10.13699           10.13982
```

```r
put.tab.1
```

```
##      Explicit FDM Implicit FDM Crank-Nicolson FDM
## [1,]     7.872837     7.259764           7.262409
```

We can see that in all 3 methods, the result are really simillar, which suggest that the value of call option and put option for given data are around 10.14 USD and 7.26 USD, respectively. The Explicit FDM produces a slightly higher results in both cases for call and put options.

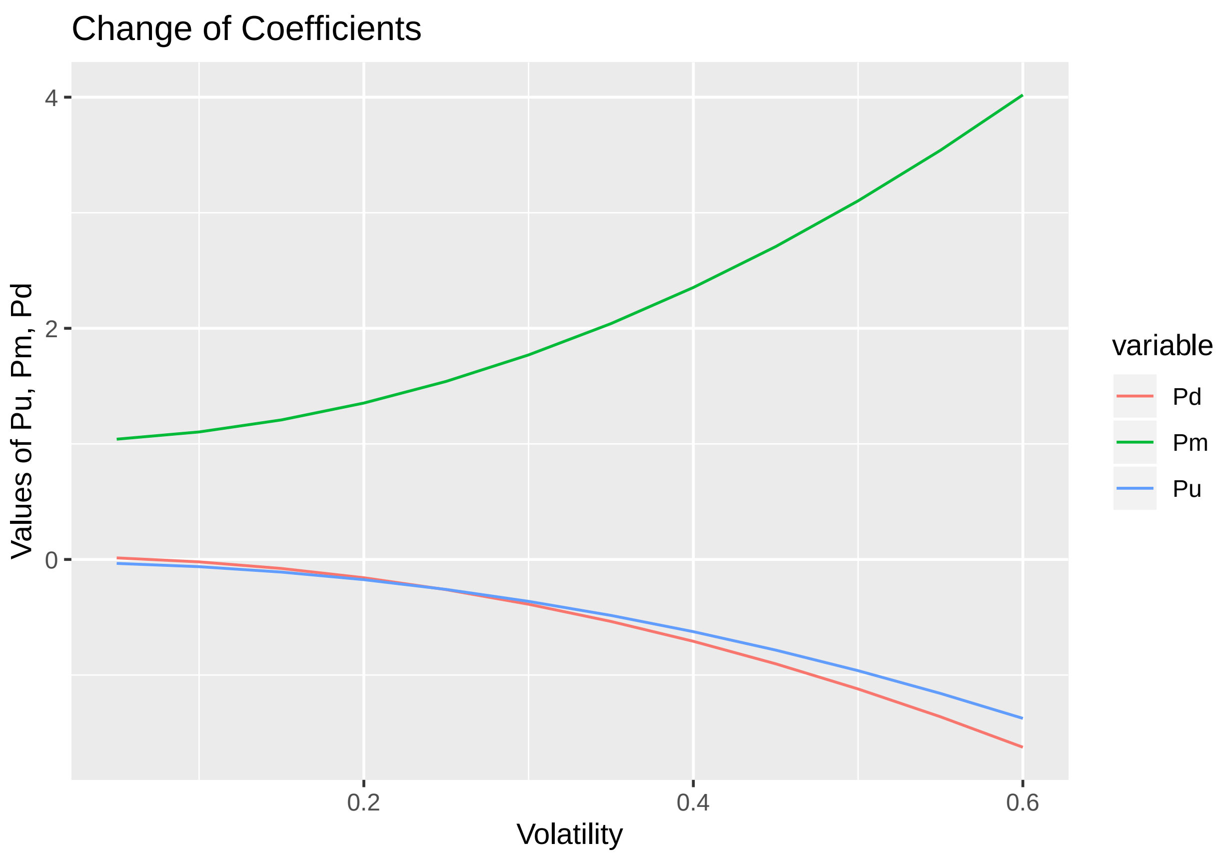
Question 1.5

Solution:

In this part, I choose time step equals 1/3 so that we can see clearly how Pu, Pm, Pd would diverge. If I choose large number of steps (i.e small time step) then it would create parallel lines.

```r
library(ggplot2)
delx.a = 0.2
delt.a = 1/3

sig.seq =seq(0.05, 0.6, 0.05)
v.seq = r - div - 0.5*sig.seq^2
pu.implicit = -0.5*delt.a*((sig.seq/delx.a)^2 + v.seq/delx.a)
pm.implicit = 1+delt.a*(sig.seq/delx.a)^2 + r*delt.a
pd.implicit = -0.5*delt.a*((sig.seq/delx.a)^2 - v.seq/delx.a)
df <- data.frame(x=c(pu.implicit, pm.implicit, pd.implicit), val=sig.seq,
                 variable=c(rep("Pu", length(sig.seq)),
                            rep("Pm", length(sig.seq)), rep("Pd", length(sig.seq)) ))
# plot
p <- ggplot(data = df, aes(x=val, y=x)) + geom_line(aes(colour=variable))
p + xlab("Volatility") + ylab("Values of Pu, Pm, Pd") + ggtitle("Change of Coefficients")
```



From the plot, we can see that, as we increase the value of σ, the values for Pu, Pm and Pd seems to diverge. Specifically, as σ increases, Pm tends to become larger, while Pu and Pd tends to converge together to the very small and negative amounts. This is because for the formula for Pu, Pm, Pd, we can see that coefficient for σ in Pm is positive, while those are negative for Pu and Pd, so when we increase σ, the result as in the plot occured.

Question 1.6

Solution:

```
#  Function computing Greeks of Call option
explicitFDM.EC.Greeks <- function(s0, k, t, r, div, sig, delx, N) {
  delt = t/N
  v = r - div - 0.5*sig^2

  pu = 0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1 - delt*(sig/delx)^2 -r*delt
  pd = 0.5*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  EC.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EC.res[1, i] = max(c(0, stock.pr.fin[i] - k))
  }
```

```
  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EC.res[i, j] = pd*EC.res[i-1, j-1] + pm*EC.res[i-1, j] + pu*EC.res[i-1, j+1]
      EC.res[i, 1] = EC.res[i, 2]
      EC.res[i, 2*N+1] = EC.res[i, 2*N] + stock.pr.fin[length(stock.pr.fin)] -
        stock.pr.fin[length(stock.pr.fin)-1]
    }
  }

  delta = (EC.res[N+1, ceiling((2*N+1)/2)+1] - EC.res[N+1, ceiling((2*N+1)/2)-1])/
    (stock.pr.fin[ceiling((2*N+1)/2)+1] - stock.pr.fin[ceiling((2*N+1)/2)-1])

  gamma = ((EC.res[N+1, ceiling((2*N+1)/2)+1] - EC.res[N+1, ceiling((2*N+1)/2)])/
    (stock.pr.fin[ceiling((2*N+1)/2)+1] - stock.pr.fin[ceiling((2*N+1)/2)]) -
    (EC.res[N+1, ceiling((2*N+1)/2)] - EC.res[N+1, ceiling((2*N+1)/2)-1])/
    (stock.pr.fin[ceiling((2*N+1)/2)] - stock.pr.fin[ceiling((2*N+1)/2)-1]))/
    (0.5*stock.pr.fin[ceiling((2*N+1)/2)+1] - 0.5*stock.pr.fin[ceiling((2*N+1)/2)-1])

  theta = (EC.res[N, ceiling((2*N+1)/2)+1] - EC.res[N+1, ceiling((2*N+1)/2)+1])/delt

  vega = (explicitFDM.EC(s0, k, t, r, div, sig*1.001, delx, N) -
    explicitFDM.EC(s0, k, t, r, div, sig*0.999, delx, N))/(2*0.001*sig)

  greeks.res <- matrix(c(delta, gamma, theta, vega), nrow = 1)
  colnames(greeks.res) <- c("Delta", "Gamma", "Theta", "Vega")
  return(greeks.res)
}

#  Function computing Greeks of Put option
explicitFDM.EP.Greeks <- function(s0, k, t, r, div, sig, delx, N) {
  delt = t/N
  v = r - div - 0.5*sig^2

  pu = 0.5*delt*((sig/delx)^2 + v/delx)
  pm = 1 - delt*(sig/delx)^2 -r*delt
  pd = 0.5*delt*((sig/delx)^2 - v/delx)

  num = seq(-N, N, 1)
  stock.pr.fin = s0*exp(num*delx)

  EP.res = matrix(nrow = N+1, ncol = 2*N+1)
  for(i in 1:length(stock.pr.fin)) {
    EP.res[1, i] = max(c(0, k - stock.pr.fin[i]))
  }

  for(i in 2:(N+1)) {
    for(j in 2:(2*N)) {
      EP.res[i, j] = pd*EP.res[i-1, j-1] + pm*EP.res[i-1, j] + pu*EP.res[i-1, j+1]
      EP.res[i, 2*N+1] = EP.res[i, 2*N]
      EP.res[i, 1] = EP.res[i, 2] + stock.pr.fin[2] - stock.pr.fin[1]
    }
  }
```

```r
  delta = (EP.res[N+1, ceiling((2*N+1)/2)+1] - EP.res[N+1, ceiling((2*N+1)/2)-1])/
    (stock.pr.fin[ceiling((2*N+1)/2)+1] - stock.pr.fin[ceiling((2*N+1)/2)-1])

  gamma = ((EP.res[N+1, ceiling((2*N+1)/2)+1] - EP.res[N+1, ceiling((2*N+1)/2)])/
             (stock.pr.fin[ceiling((2*N+1)/2)+1] - stock.pr.fin[ceiling((2*N+1)/2)]) -
             (EP.res[N+1, ceiling((2*N+1)/2)] - EP.res[N+1, ceiling((2*N+1)/2)-1])/
             (stock.pr.fin[ceiling((2*N+1)/2)] - stock.pr.fin[ceiling((2*N+1)/2)-1]))/
    (0.5*stock.pr.fin[ceiling((2*N+1)/2)+1] - 0.5*stock.pr.fin[ceiling((2*N+1)/2)-1])

  theta = (EP.res[N, ceiling((2*N+1)/2)+1] - EP.res[N+1, ceiling((2*N+1)/2)+1])/delt

  vega = (explicitFDM.EP(s0, k, t, r, div, sig*1.001, delx, N) -
            explicitFDM.EP(s0, k, t, r, div, sig*0.999, delx, N))/(2*0.001*sig)

  greeks.res <- matrix(c(delta, gamma, theta, vega), nrow = 1)
  colnames(greeks.res) <- c("Delta", "Gamma", "Theta", "Vega")
  return(greeks.res)
}
```

```r
# Greeks of Call option
explicitFDM.EC.Greeks(s0, k, t, r, div, sig, 0.2, 675)
```

```
##         Delta      Gamma     Theta     Vega
## [1,] 0.587417 0.01675375 -4.698241 42.01669
```

```r
# Greeks of Put option
explicitFDM.EP.Greeks(s0, k, t, r, div, sig, 0.2, 675)
```

```
##          Delta      Gamma     Theta     Vega
## [1,] -0.383121 0.01675375 -2.592202 42.09779
```

The result for Greeks of call and put options are presented in 2 tables above.

Question 2

Question 2.1

Solution:

```r
library(quantmod)

# Getting Data function
get.data.FDM <- function(ticker, start.date, end.date,
                         exp.date1, exp.date2, exp.date3) {
  secp <- getSymbols(Symbols = as.character(ticker), from = start.date,
                     to = end.date, auto.assign = F, src = 'yahoo')
  optd1 <- getOptionChain(as.character(ticker), auto.assign = F,
                          exp.date1, src = 'yahoo')
  optd2 <- getOptionChain(as.character(ticker), auto.assign = F,
                          exp.date2, src = 'yahoo')
  optd3 <- getOptionChain(as.character(ticker), auto.assign = F,
                          exp.date3, src = 'yahoo')

  c1 <- data.frame(optd1$calls)
  colnames(c1) <- c(paste(c(ticker), "Call.1m.Strike", sep = "."),
```

```
                    paste(c(ticker), "Call.1m.Last", sep = "."),
                    paste(c(ticker), "Call.1m.Chg", sep = "."),
                    paste(c(ticker), "Call.1m.Bid", sep = "."),
                    paste(c(ticker), "Call.1m.Ask", sep = "."),
                    paste(c(ticker), "Call.1m.Vol", sep = "."),
                    paste(c(ticker), "Call.1m.OI", sep = "."))
  p1 <- data.frame(optd1$puts)
  colnames(p1) <- c(paste(c(ticker), "Put.1m.Strike", sep = "."),
                    paste(c(ticker), "Put.1m.Last", sep = "."),
                    paste(c(ticker), "Put.1m.Chg", sep = "."),
                    paste(c(ticker), "Put.1m.Bid", sep = "."),
                    paste(c(ticker), "Put.1m.Ask", sep = "."),
                    paste(c(ticker), "Put.1m.Vol", sep = "."),
                    paste(c(ticker), "Put.1m.OI", sep = "."))

  c2 <- data.frame(optd2$calls)
  colnames(c2) <- c(paste(c(ticker), "Call.2m.Strike", sep = "."),
                    paste(c(ticker), "Call.2m.Last", sep = "."),
                    paste(c(ticker), "Call.2m.Chg", sep = "."),
                    paste(c(ticker), "Call.2m.Bid", sep = "."),
                    paste(c(ticker), "Call.2m.Ask", sep = "."),
                    paste(c(ticker), "Call.2m.Vol", sep = "."),
                    paste(c(ticker), "Call.2m.OI", sep = "."))
  p2 <- data.frame(optd2$puts)
  colnames(p2) <- c(paste(c(ticker), "Put.2m.Strike", sep = "."),
                    paste(c(ticker), "Put.2m.Last", sep = "."),
                    paste(c(ticker), "Put.2m.Chg", sep = "."),
                    paste(c(ticker), "Put.2m.Bid", sep = "."),
                    paste(c(ticker), "Put.2m.Ask", sep = "."),
                    paste(c(ticker), "Put.2m.Vol", sep = "."),
                    paste(c(ticker), "Put.2m.OI", sep = "."))

  c3 <- data.frame(optd3$calls)
  colnames(c3) <- c(paste(c(ticker), "Call.3m.Strike", sep = "."),
                    paste(c(ticker), "Call.3m.Last", sep = "."),
                    paste(c(ticker), "Call.3m.Chg", sep = "."),
                    paste(c(ticker), "Call.3m.Bid", sep = "."),
                    paste(c(ticker), "Call.3m.Ask", sep = "."),
                    paste(c(ticker), "Call.3m.Vol", sep = "."),
                    paste(c(ticker), "Call.3m.OI", sep = "."))
  p3 <- data.frame(optd3$puts)
  colnames(p3) <- c(paste(c(ticker), "Put.3m.Strike", sep = "."),
                    paste(c(ticker), "Put.3m.Last", sep = "."),
                    paste(c(ticker), "Put.3m.Chg", sep = "."),
                    paste(c(ticker), "Put.3m.Bid", sep = "."),
                    paste(c(ticker), "Put.3m.Ask", sep = "."),
                    paste(c(ticker), "Put.3m.Vol", sep = "."),
                    paste(c(ticker), "Put.3m.OI", sep = "."))
  secp <- data.frame(secp)
  all.dat <- c(secp, c1, p1, c2, p2, c3, p3)
  return(all.dat)
}
```

Implied volatility functions for put and call options.

```r
# Implied volatility
# Using Newton Method
# Call
CallBSIMPVol <- function(spot.price, time.to.maturity, risk.free, strike.price,
                         option.price, initial.guess) {

  fx <-  function(sig){
    d1 <- (log(spot.price/strike.price) + (risk.free + sig^2 * 0.5) * time.to.maturity) /
      (sig * sqrt(time.to.maturity))
    d2 <- d1 - sig * sqrt(time.to.maturity)
    value <- spot.price * pnorm(d1) - strike.price * exp( -risk.free * time.to.maturity) *
      pnorm(d2) - option.price
    return(value)
  }

  dfx <- function(sig){
    d1 <- (log(spot.price/strike.price) + (risk.free + sig^2 * 0.5) * time.to.maturity) /
      (sig*sqrt(time.to.maturity))
    d2 <- d1 - sig * sqrt(time.to.maturity)
    value <- strike.price * exp( -risk.free * time.to.maturity) * dnorm(d2) *
      sqrt(time.to.maturity)
    return(value)
  }

  q1 <- function(x0){
    epsilon <- 0.5
    while(T){
      tmp <-  x0
      deltaX  <-  fx(x0)/dfx(x0)
      x0  <-  x0 - deltaX
      if (abs(x0-tmp) < epsilon){

        break
      }else {
        print(x0)
      }
    }
    return(x0)
  }

  IMPvol <- q1(initial.guess)
  return(IMPvol)
}

# Put
PutBSIMPVol <- function(spot.price, time.to.maturity, risk.free, strike.price,
                        option.price, initial.guess) {

  fx <-  function(sig){
    d1 <- (log(spot.price/strike.price) + (risk.free + sig^2 * 0.5) * time.to.maturity) /
      (sig * sqrt(time.to.maturity))
    d2 <- d1 - sig * sqrt(time.to.maturity)
    value <- pnorm(-d2) * strike.price * exp(-risk.free * time.to.maturity) - pnorm(-d1) *
```

```
      spot.price - option.price
    return(value)
  }

  dfx <- function(sig){
    d1 <- (log(spot.price/strike.price) + (risk.free + sig^2 * 0.5) * time.to.maturity) /
      (sig*sqrt(time.to.maturity))
    d2 <- d1 - sig * sqrt(time.to.maturity)
    value <- strike.price * exp( -risk.free * time.to.maturity) * dnorm(-d2) *
      sqrt(time.to.maturity)
    return(value)
  }

  q1 <- function(x0){
    epsilon <- 0.5
    while(T){
      tmp <-  x0
      deltaX  <-  fx(x0)/dfx(x0)
      x0  <-  x0 - deltaX
      if (abs(x0-tmp) < epsilon){

        break
      }else {

      }
    }
    return(x0)
  }

  IMPvol <- q1(initial.guess)
  return(IMPvol)
}
```

Here, I use the federal fund rate of the 22nd October, 2019. I choose 10 strike price that are closest to the spot price of the stock, by considering the difference to see which ones are closest and choose them.

```
AMZN.FDM <- get.data.FDM("AMZN", "2019-10-22", "2019-10-23", "2019-11-22",
                         "2019-12-20", "2020-01-17")
amzn.fdm.sp = AMZN.FDM$AMZN.Adjusted
r.fdm <- 0.0185 # 22 Oct

which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Call.1m.Strike))
```

```
## [1] 51
```

```
which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Call.2m.Strike))
```

```
## [1] 50
```

```
which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Call.3m.Strike))
```

```
## [1] 113
```

```
which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Put.1m.Strike))
```

```
## [1] 111
```

```r
which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Put.2m.Strike))
```

```
## [1] 72
```

```r
which.min(abs(AMZN.FDM$AMZN.Adjusted - AMZN.FDM$AMZN.Put.3m.Strike))
```

```
## [1] 113
```

```r
amzn.fdm.call.1m.strike = AMZN.FDM$AMZN.Call.1m.Strike[45:54]
amzn.fdm.call.2m.strike = AMZN.FDM$AMZN.Call.2m.Strike[46:55]
amzn.fdm.call.3m.strike = AMZN.FDM$AMZN.Call.3m.Strike[109:118]

amzn.fdm.put.1m.strike = AMZN.FDM$AMZN.Put.1m.Strike[101:110]
amzn.fdm.put.2m.strike = AMZN.FDM$AMZN.Put.2m.Strike[68:77]
amzn.fdm.put.3m.strike = AMZN.FDM$AMZN.Put.3m.Strike[109:118]

# Calculating options price to compute volatility
amzn.fdm.call.option.price.1m <- c()
amzn.fdm.call.option.price.2m <- c()
amzn.fdm.call.option.price.3m <- c()
amzn.fdm.put.option.price.1m <- c()
amzn.fdm.put.option.price.2m <- c()
amzn.fdm.put.option.price.3m <- c()
for(i in 1:10) {
  amzn.fdm.call.option.price.1m[i] = (AMZN.FDM$AMZN.Call.1m.Ask[i+44] +
                                       AMZN.FDM$AMZN.Call.1m.Bid[i+44])/2
  amzn.fdm.call.option.price.2m[i] = (AMZN.FDM$AMZN.Call.2m.Ask[i+45] +
                                       AMZN.FDM$AMZN.Call.2m.Bid[i+45])/2
  amzn.fdm.call.option.price.3m[i] = (AMZN.FDM$AMZN.Call.3m.Ask[i+108] +
                                       AMZN.FDM$AMZN.Call.3m.Bid[i+108])/2
  amzn.fdm.put.option.price.1m[i] = (AMZN.FDM$AMZN.Put.1m.Ask[i+100] +
                                      AMZN.FDM$AMZN.Put.1m.Bid[i+100])/2
  amzn.fdm.put.option.price.2m[i] = (AMZN.FDM$AMZN.Put.2m.Ask[i+67] +
                                      AMZN.FDM$AMZN.Put.2m.Bid[i+67])/2
  amzn.fdm.put.option.price.3m[i] = (AMZN.FDM$AMZN.Put.3m.Ask[i+108] +
                                      AMZN.FDM$AMZN.Put.3m.Bid[i+108])/2
}

t1 = 31/365
t2 = 59/365
t3 = 87/365

# Computing volatility
amzn.fdm.vol.c1 <- c()
amzn.fdm.vol.c2 <- c()
amzn.fdm.vol.c3 <- c()
amzn.fdm.vol.p1 <- c()
amzn.fdm.vol.p2 <- c()
amzn.fdm.vol.p3 <- c()
for(i in 1:10) {
  amzn.fdm.vol.c1[i] = CallBSIMPVol(amzn.fdm.sp, t1, r.fdm, amzn.fdm.call.1m.strike[i],
                                    amzn.fdm.call.option.price.1m[i], 0.5)
  amzn.fdm.vol.c2[i] = CallBSIMPVol(amzn.fdm.sp, t2, r.fdm, amzn.fdm.call.2m.strike[i],
                                    amzn.fdm.call.option.price.2m[i], 0.5)
  amzn.fdm.vol.c3[i] = CallBSIMPVol(amzn.fdm.sp, t3, r.fdm, amzn.fdm.call.3m.strike[i],
```

```
                                    amzn.fdm.call.option.price.3m[i], 0.5)
  amzn.fdm.vol.p1[i] = PutBSIMPVol(amzn.fdm.sp, t1, r.fdm, amzn.fdm.put.1m.strike[i],
                                   amzn.fdm.put.option.price.1m[i], 0.5)
  amzn.fdm.vol.p2[i] = PutBSIMPVol(amzn.fdm.sp, t2, r.fdm, amzn.fdm.put.2m.strike[i],
                                   amzn.fdm.put.option.price.2m[i], 0.5)
  amzn.fdm.vol.p3[i] = PutBSIMPVol(amzn.fdm.sp, t3, r.fdm, amzn.fdm.put.3m.strike[i],
                                     amzn.fdm.put.option.price.3m[i], 0.5)
}
```

Question 2.2

Solution:

Since the implied volatility for the same terms (1, 2 and 3 months) of different strike prices are very close, so I can choose 1 value of space step for each terms of all 10 strike prices. For 1 month, I choose $\Delta x = 0.02$, for 2 months, $\Delta x = 0.025$ and for 3 months, $\Delta x = 0.027$. Also, I use 675 time steps as in Problem 1 for calculation.

```
# Computing price of options using FDMs implemented above
explicit.price.call.1m.pred <- c()
explicit.price.call.2m.pred <- c()
explicit.price.call.3m.pred <- c()
explicit.price.put.1m.pred <- c()
explicit.price.put.2m.pred <- c()
explicit.price.put.3m.pred <- c()

implicit.price.call.1m.pred <- c()
implicit.price.call.2m.pred <- c()
implicit.price.call.3m.pred <- c()
implicit.price.put.1m.pred <- c()
implicit.price.put.2m.pred <- c()
implicit.price.put.3m.pred <- c()

CN.price.call.1m.pred <- c()
CN.price.call.2m.pred <- c()
CN.price.call.3m.pred <- c()
CN.price.put.1m.pred <- c()
CN.price.put.2m.pred <- c()
CN.price.put.3m.pred <- c()

for(i in 1:10) {
  explicit.price.call.1m.pred[i] = explicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.1m.strike[i],
                                                  t1, r.fdm, 0, amzn.fdm.vol.c1[i], 0.02, 675)
  explicit.price.call.2m.pred[i] = explicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.2m.strike[i],
                                                  t2, r.fdm, 0, amzn.fdm.vol.c2[i], 0.025, 675)
  explicit.price.call.3m.pred[i] = explicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.3m.strike[i],
                                                  t3, r.fdm, 0, amzn.fdm.vol.c3[i], 0.027, 675)
  explicit.price.put.1m.pred[i] = explicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.1m.strike[i],
                                                 t1, r.fdm, 0, amzn.fdm.vol.p1[i], 0.02, 675)
  explicit.price.put.2m.pred[i] = explicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.2m.strike[i],
                                                 t2, r.fdm, 0, amzn.fdm.vol.p2[i], 0.025, 675)
  explicit.price.put.3m.pred[i] = explicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.3m.strike[i],
                                                 t3, r.fdm, 0, amzn.fdm.vol.p3[i], 0.027, 675)
```

```r
  implicit.price.call.1m.pred[i] = implicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.1m.strike[i],
                                                  t1, r.fdm, 0, amzn.fdm.vol.c1[i], 0.02, 675)
  implicit.price.call.2m.pred[i] = implicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.2m.strike[i],
                                                  t2, r.fdm, 0, amzn.fdm.vol.c2[i], 0.025, 675)
  implicit.price.call.3m.pred[i] = implicitFDM.EC(amzn.fdm.sp, amzn.fdm.call.3m.strike[i],
                                                  t3, r.fdm, 0, amzn.fdm.vol.c3[i], 0.027, 675)
  implicit.price.put.1m.pred[i] = implicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.1m.strike[i],
                                                 t1, r.fdm, 0, amzn.fdm.vol.p1[i], 0.02, 675)
  implicit.price.put.2m.pred[i] = implicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.2m.strike[i],
                                                 t2, r.fdm, 0, amzn.fdm.vol.p2[i], 0.025, 675)
  implicit.price.put.3m.pred[i] = implicitFDM.EP(amzn.fdm.sp, amzn.fdm.put.3m.strike[i],
                                                 t3, r.fdm, 0, amzn.fdm.vol.p3[i], 0.027, 675)

  CN.price.call.1m.pred[i] = CN.FDM.EC(amzn.fdm.sp, amzn.fdm.call.1m.strike[i],
                                       t1, r.fdm, 0, amzn.fdm.vol.c1[i], 0.02, 675)
  CN.price.call.2m.pred[i] = CN.FDM.EC(amzn.fdm.sp, amzn.fdm.call.2m.strike[i],
                                       t2, r.fdm, 0, amzn.fdm.vol.c2[i], 0.025, 675)
  CN.price.call.3m.pred[i] = CN.FDM.EC(amzn.fdm.sp, amzn.fdm.call.3m.strike[i],
                                       t3, r.fdm, 0, amzn.fdm.vol.c3[i], 0.027, 675)
  CN.price.put.1m.pred[i] = CN.FDM.EP(amzn.fdm.sp, amzn.fdm.put.1m.strike[i],
                                      t1, r.fdm, 0, amzn.fdm.vol.p1[i], 0.02, 675)
  CN.price.put.2m.pred[i] = CN.FDM.EP(amzn.fdm.sp, amzn.fdm.put.2m.strike[i],
                                      t2, r.fdm, 0, amzn.fdm.vol.p2[i], 0.025, 675)
  CN.price.put.3m.pred[i] = CN.FDM.EP(amzn.fdm.sp, amzn.fdm.put.3m.strike[i],
                                      t3, r.fdm, 0, amzn.fdm.vol.p3[i], 0.027, 675)
}
```

```r
# Result table
FDM.call.tab.1m <- matrix(c(explicit.price.call.1m.pred, implicit.price.call.1m.pred,
                          CN.price.call.1m.pred, amzn.fdm.call.option.price.1m),
                          ncol = 10, byrow = T)
colnames(FDM.call.tab.1m) <- c(as.character(amzn.fdm.call.1m.strike))
rownames(FDM.call.tab.1m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                               "(Bid + Ask)/2")
FDM.call.tab.1m
```

```
##                        1750   1752.5     1755   1757.5     1760   1762.5
## Explicit FDM       51.23269 49.49387 47.78659 46.38679 44.54352 42.83218
## Implicit FDM       51.21780 49.47891 47.77153 46.37152 44.52819 42.81674
## Crank Nicolson FDM 51.22525 49.48639 47.77906 46.37916 44.53586 42.82446
## (Bid + Ask)/2      51.07500 49.42500 47.82500 46.55000 44.85000 43.30000
##                        1765   1767.5     1770   1772.5
## Explicit FDM       41.28010 39.86164 38.68488 37.44286
## Implicit FDM       41.26449 39.84620 38.66969 37.42795
## Crank Nicolson FDM 41.27230 39.85392 38.67729 37.43541
## (Bid + Ask)/2      41.92500 40.45000 39.12500 37.75000
```

```r
FDM.call.tab.2m <- matrix(c(explicit.price.call.2m.pred, implicit.price.call.2m.pred,
                            CN.price.call.2m.pred, amzn.fdm.call.option.price.2m),
                          ncol = 10, byrow = T)
colnames(FDM.call.tab.2m) <- c(as.character(amzn.fdm.call.2m.strike))
rownames(FDM.call.tab.2m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                               "(Bid + Ask)/2")
FDM.call.tab.2m
```

```
##                         1745     1750     1755     1760     1765     1770
## Explicit FDM        73.54250 70.18822 66.90810 63.77882 60.60005 57.96978
## Implicit FDM        73.52081 70.16640 66.88611 63.75660 60.57762 57.94773
## Crank Nicolson FDM 73.53166 70.17731 66.89711 63.76771 60.58884 57.95876
## (Bid + Ask)/2       73.42500 70.22500 67.15000 64.27500 61.40000 58.60000
##                         1775     1780     1785     1790
## Explicit FDM        55.61333 53.08834 50.69701 48.21368
## Implicit FDM        55.59169 53.06720 50.67631 48.19346
## Crank Nicolson FDM 55.60251 53.07778 50.68666 48.20357
## (Bid + Ask)/2       56.02500 53.32500 50.80000 48.22500
```

```
FDM.call.tab.3m <- matrix(c(explicit.price.call.3m.pred, implicit.price.call.3m.pred,
                            CN.price.call.3m.pred, amzn.fdm.call.option.price.3m),
                          ncol = 10, byrow = T)
colnames(FDM.call.tab.3m) <- c(as.character(amzn.fdm.call.3m.strike))
rownames(FDM.call.tab.3m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                               "(Bid + Ask)/2")
FDM.call.tab.3m
```

```
##                         1680     1700     1720     1740     1760     1780
## Explicit FDM        135.8247 120.9428 106.4521 93.85304 81.27323 70.56803
## Implicit FDM        135.8013 120.9176 106.4248 93.82527 81.24491 70.54065
## Crank Nicolson FDM 135.8130 120.9302 106.4384 93.83916 81.25907 70.55434
## (Bid + Ask)/2       134.1250 119.7500 106.3750 93.80000 81.87500 70.97500
##                         1800     1810     1815     1820
## Explicit FDM        60.79499 56.10121 53.92295 52.09710
## Implicit FDM        60.76916 56.07608 53.89825 52.07325
## Crank Nicolson FDM 60.78208 56.08865 53.91060 52.08518
## (Bid + Ask)/2       60.97500 56.37500 54.20000 52.00000
```

```
FDM.put.tab.1m <- matrix(c(explicit.price.put.1m.pred, implicit.price.put.1m.pred,
                            CN.price.put.1m.pred, amzn.fdm.put.option.price.1m),
                         ncol = 10, byrow = T)
colnames(FDM.put.tab.1m) <- c(as.character(amzn.fdm.put.1m.strike))
rownames(FDM.put.tab.1m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                              "(Bid + Ask)/2")
FDM.put.tab.1m
```

```
##                         1740   1742.5     1745   1747.5     1750   1752.5
## Explicit FDM        21.03824 21.78207 22.48214 23.23469 24.08971 24.75335
## Implicit FDM        21.02742 21.77111 22.47105 23.22345 24.07827 24.74177
## Crank Nicolson FDM 21.03283 21.77659 22.47660 23.22907 24.08399 24.74756
## (Bid + Ask)/2       20.40000 21.20000 21.97500 22.82500 23.80000 24.60000
##                         1755   1757.5     1760   1762.5
## Explicit FDM        25.59409 26.44066 27.19210 27.99892
## Implicit FDM        25.58230 26.42864 27.17989 27.98649
## Crank Nicolson FDM 25.58820 26.43465 27.18600 27.99271
## (Bid + Ask)/2       25.60000 26.62500 27.57500 28.60000
```

```
FDM.put.tab.2m <- matrix(c(explicit.price.put.2m.pred, implicit.price.put.2m.pred,
                           CN.price.put.2m.pred, amzn.fdm.put.option.price.2m),
                         ncol = 10, byrow = T)
colnames(FDM.put.tab.2m) <- c(as.character(amzn.fdm.put.2m.strike))
rownames(FDM.put.tab.2m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                              "(Bid + Ask)/2")
FDM.put.tab.2m
```

```
##                        1745     1750     1755     1760     1765     1770
## Explicit FDM       38.63852 40.20251 41.93614 43.79304 45.52152 47.94476
## Implicit FDM       38.62038 40.18416 41.91753 43.77410 45.50229 47.92595
## Crank Nicolson FDM 38.62945 40.19334 41.92684 43.78357 45.51191 47.93536
## (Bid + Ask)/2      38.42500 40.20000 42.20000 44.37500 46.47500 48.70000
##                        1775     1780     1785     1790
## Explicit FDM       50.53147 53.04980 55.80065 58.38308
## Implicit FDM       50.51320 53.03210 55.78344 58.36642
## Crank Nicolson FDM 50.52234 53.04096 55.79205 58.37475
## (Bid + Ask)/2      51.02500 53.32500 55.90000 58.35000
```

```r
FDM.put.tab.3m <- matrix(c(explicit.price.put.3m.pred, implicit.price.put.3m.pred,
                           CN.price.put.3m.pred, amzn.fdm.put.option.price.3m),
                         ncol = 10, byrow = T)
colnames(FDM.put.tab.3m) <- c(as.character(amzn.fdm.put.3m.strike))
rownames(FDM.put.tab.3m) <- c("Explicit FDM", "Implicit FDM", "Crank Nicolson FDM",
                              "(Bid + Ask)/2")
FDM.put.tab.3m
```

```
##                        1680     1700     1720     1740     1760     1780
## Explicit FDM       33.88098 38.74462 43.69177 50.80857 58.13146 67.32171
## Implicit FDM       33.86222 38.72363 43.66846 50.78464 58.10675 67.29806
## Crank Nicolson FDM 33.87160 38.73413 43.68012 50.79661 58.11911 67.30989
## (Bid + Ask)/2      31.37500 37.02500 43.40000 50.67500 58.82500 67.80000
##                        1800     1810     1815     1820
## Explicit FDM       77.89231 83.26238 86.16428 89.32749
## Implicit FDM       77.87030 83.24117 86.14352 89.30772
## Crank Nicolson FDM 77.88131 83.25178 86.15390 89.31761
## (Bid + Ask)/2      78.05000 83.47500 86.35000 89.05000
```

All result produces result roughly the same as the real options prices, especially for options with large strike prices.

Question 2.3

Solution:

```r
call.greeks.1m <- vector("list", 10)
call.greeks.2m <- vector("list", 10)
call.greeks.3m <- vector("list", 10)
put.greeks.1m <- vector("list", 10)
put.greeks.2m <- vector("list", 10)
put.greeks.3m <- vector("list", 10)

for(i in 1:10) {
  call.greeks.1m[[i]] = explicitFDM.EC.Greeks(amzn.fdm.sp, amzn.fdm.call.1m.strike[i],
                                              t1, r.fdm, 0, amzn.fdm.vol.c1[i], 0.02, 675)
  call.greeks.2m[[i]] = explicitFDM.EC.Greeks(amzn.fdm.sp, amzn.fdm.call.2m.strike[i],
                                              t2, r.fdm, 0, amzn.fdm.vol.c2[i], 0.025, 675)
  call.greeks.3m[[i]] = explicitFDM.EC.Greeks(amzn.fdm.sp, amzn.fdm.call.3m.strike[i],
                                              t3, r.fdm, 0, amzn.fdm.vol.c3[i], 0.027, 675)
  put.greeks.1m[[i]] = explicitFDM.EP.Greeks(amzn.fdm.sp, amzn.fdm.put.1m.strike[i],
                                             t1, r.fdm, 0, amzn.fdm.vol.p1[i], 0.02, 675)
  put.greeks.2m[[i]] = explicitFDM.EP.Greeks(amzn.fdm.sp, amzn.fdm.put.2m.strike[i],
                                             t2, r.fdm, 0, amzn.fdm.vol.p2[i], 0.025, 675)
```

```r
  put.greeks.3m[[i]] = explicitFDM.EP.Greeks(amzn.fdm.sp, amzn.fdm.put.3m.strike[i],
                                            t3, r.fdm, 0, amzn.fdm.vol.p3[i], 0.027, 675)
}

names(call.greeks.1m) <- amzn.fdm.call.1m.strike
names(call.greeks.2m) <- amzn.fdm.call.2m.strike
names(call.greeks.3m) <- amzn.fdm.call.3m.strike
names(put.greeks.1m) <- amzn.fdm.put.1m.strike
names(put.greeks.2m) <- amzn.fdm.put.2m.strike
names(put.greeks.3m) <- amzn.fdm.put.3m.strike
```

```r
call.greeks.1m
```

```
## $`1750`
##          Delta       Gamma     Theta    Vega
## [1,] 0.5819962 0.003745712 -236.5946 200.945
##
## $`1752.5`
##         Delta      Gamma    Theta     Vega
## [1,] 0.572951 0.00379635 -237.303 202.0583
##
## $`1755`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5637345 0.003845113 -238.2659 203.2063
##
## $`1757.5`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5541658 0.003865463 -241.2273 204.3846
##
## $`1760`
##          Delta      Gamma     Theta     Vega
## [1,] 0.5448283 0.00392868 -241.4073 205.5845
##
## $`1762.5`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5352649 0.003980351 -242.4718 206.8158
##
## $`1765`
##          Delta       Gamma    Theta     Vega
## [1,] 0.5255614 0.004016732 -244.556 208.0525
##
## $`1767.5`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5157808 0.004039715 -244.6928 207.9801
##
## $`1770`
##          Delta       Gamma    Theta    Vega
## [1,] 0.5059415 0.004037788 -245.156 207.322
##
## $`1772.5`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4960073 0.004042436 -245.2229 206.6591
```

```r
call.greeks.2m
```

```
## $`1745`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5855122 0.002584161 -188.6514 276.7038
##
## $`1750`
##          Delta       Gamma    Theta     Vega
## [1,] 0.5728224 0.002629612 -189.378 278.8024
##
## $`1755`
##          Delta       Gamma     Theta    Vega
## [1,] 0.5598534 0.002672859 -190.4097 280.973
##
## $`1760`
##         Delta       Gamma     Theta    Vega
## [1,] 0.546635 0.002710036 -191.9818 283.199
##
## $`1765`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5332647 0.002750195 -193.4352 285.4749
##
## $`1770`
##          Delta       Gamma   Theta     Vega
## [1,] 0.5197442 0.002764446 -193.58 285.1769
##
## $`1775`
##          Delta       Gamma    Theta     Vega
## [1,] 0.5061447 0.002765424 -194.066 284.3971
##
## $`1780`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4923447 0.002774421 -194.0203 283.6024
##
## $`1785`
##          Delta       Gamma    Theta     Vega
## [1,] 0.4785035 0.002776858 -194.397 282.7821
##
## $`1790`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4644489 0.002783616 -194.4839 281.9337
```

```r
call.greeks.3m
```

```
## $`1680`
##          Delta       Gamma     Theta     Vega
## [1,] 0.6975394 0.001672002 -158.7265 301.3303
##
## $`1700`
##          Delta       Gamma     Theta     Vega
## [1,] 0.6649825 0.001798556 -161.6764 313.9395
##
## $`1720`
##        Delta       Gamma     Theta     Vega
## [1,] 0.62996 0.001927391 -165.8097 327.6025
```

```
##
## $`1740`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5909663 0.002018343 -168.0964 334.9892
##
## $`1760`
##          Delta       Gamma     Theta     Vega
## [1,] 0.5501076 0.002114318 -170.7809 342.9341
##
## $`1780`
##         Delta       Gamma     Theta     Vega
## [1,] 0.507668 0.002159836 -171.7741 344.3628
##
## $`1800`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4641548 0.002178018 -172.3808 342.8597
##
## $`1810`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4422417 0.002181247 -173.0937 342.0629
##
## $`1815`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4314756 0.002177877 -173.3905 341.2255
##
## $`1820`
##          Delta       Gamma     Theta     Vega
## [1,] 0.4212056 0.002164107 -172.6412 338.4796
```

```
put.greeks.1m
```

```
## $`1740`
##           Delta       Gamma     Theta    Vega
## [1,] -0.3589194 0.004418759 -140.5761 192.535
##
## $`1742.5`
##           Delta       Gamma     Theta     Vega
## [1,] -0.3694711 0.004493442 -141.6922 194.0563
##
## $`1745`
##          Delta       Gamma     Theta     Vega
## [1,] -0.380084 0.004575932 -142.5607 195.6125
##
## $`1747.5`
##           Delta       Gamma     Theta     Vega
## [1,] -0.3910569 0.004653709 -143.8702 197.2599
##
## $`1750`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4024549 0.004718989 -145.9483 198.9997
##
## $`1752.5`
##           Delta       Gamma     Theta    Vega
## [1,] -0.4135887 0.004813404 -146.7632 200.736
##
```

```
## $`1755`
##           Delta      Gamma     Theta     Vega
## [1,] -0.4252608 0.00488391 -148.8416 202.5708
##
## $`1757.5`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4370639 0.004954774 -150.9975 204.4463
##
## $`1760`
##           Delta       Gamma     Theta    Vega
## [1,] -0.4488941 0.005042195 -152.5748 206.375
##
## $`1762.5`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4609405 0.005122565 -154.5614 208.3519
```

```
put.greeks.2m
```

```
## $`1745`
##           Delta      Gamma     Theta    Vega
## [1,] -0.4051964 0.00303353 -125.7938 275.132
##
## $`1750`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4199597 0.003099422 -126.7322 277.9102
##
## $`1755`
##           Delta       Gamma    Theta     Vega
## [1,] -0.4351742 0.003156613 -128.316 280.8099
##
## $`1760`
##           Delta      Gamma     Theta     Vega
## [1,] -0.4506716 0.00320705 -130.3576 283.7824
##
## $`1765`
##           Delta       Gamma    Theta     Vega
## [1,] -0.4663144 0.003266969 -132.027 286.8362
##
## $`1770`
##           Delta     Gamma     Theta     Vega
## [1,] -0.4822134 0.0032805 -132.3846 286.2307
##
## $`1775`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4982847 0.003282696 -132.6558 284.9475
##
## $`1780`
##           Delta      Gamma     Theta     Vega
## [1,] -0.5145793 0.00328926 -132.7106 283.6259
##
## $`1785`
##           Delta       Gamma     Theta     Vega
## [1,] -0.5308118 0.003279832 -133.5053 282.2717
##
## $`1790`
```

```
##           Delta       Gamma     Theta     Vega
## [1,] -0.5473651 0.003281494 -133.7658 280.8754
```

```
put.greeks.3m
```

```
## $`1680`
##           Delta     Gamma     Theta     Vega
## [1,] -0.2830876 0.0018471 -103.8832 292.0812
##
## $`1700`
##           Delta       Gamma     Theta     Vega
## [1,] -0.3191557 0.002009782 -107.5126 307.6085
##
## $`1720`
##           Delta       Gamma     Theta     Vega
## [1,] -0.3576239 0.002186074 -111.6089 324.4006
##
## $`1740`
##           Delta       Gamma    Theta     Vega
## [1,] -0.4018303 0.002307921 -114.109 333.4619
##
## $`1760`
##           Delta       Gamma     Theta     Vega
## [1,] -0.4484513 0.002430869 -117.4984 343.3449
##
## $`1780`
##           Delta       Gamma    Theta     Vega
## [1,] -0.4970715 0.002487288 -118.417 344.5694
##
## $`1800`
##           Delta       Gamma     Theta    Vega
## [1,] -0.5466954 0.002489762 -119.5893 341.902
##
## $`1810`
##          Delta       Gamma     Theta     Vega
## [1,] -0.571692 0.002487428 -120.3566 340.5206
##
## $`1815`
##           Delta       Gamma     Theta     Vega
## [1,] -0.5838401 0.002477985 -120.7219 339.2804
##
## $`1820`
##           Delta       Gamma     Theta    Vega
## [1,] -0.5955414 0.002458469 -119.5799 335.536
```

The results for Greeks of call and put options with different strike prices as presented in tables above. We can see baisc properties of Greeks, such as Delta of call options are always positive, while those are always negative for put options, or Vega of the same strike price for call and put options are the same. In the case above, Vega of same strike prices are roughly the same.

Question 3

Solution:

For this problem, I use the grid of 50 possible values for stock price and 40 possible values for implied volatility at the maturity. At maturity, stock prices ranges from 0 to 2 times value of stock prices and volatility range from 0 to 1.8 time values of implied volaatility. I also use 2000 time steps for calculation. After the 2000th time steps, I got a matrix of possible values for options at the beginning. This is the 2 dimensions matrix of the corresponding stock price and volatility, hence I use 2 dimensional interpolation to get the value of options with given data.

```
library(pracma)
s0 = 100
K = 100
t = 1
r = 0.06
v0 = 0.0625
kappa = 2
theta = 0.09
sig = 0.5
rho = 0.6

# Computing range of stock price, volatility, ds, dt, dv
s.vec <- seq(0, 2*s0, length.out = 50)
sig.vec <- seq(0, 1.8*sig, length.out = 40)
NS = length(s.vec)
NV = length(sig.vec)
NT = 2000

ds = (s.vec[length(s.vec)] - s.vec[1])/NS
dv = (sig.vec[length(sig.vec)] - sig.vec[1])/NV
dt = t/NT

# Initializing price data as an array of 3 dimensions
price.array <- array(data = NA, dim = c(NS, NV, NT+1))
for(s in 1:NS) {
  price.array[s, , 1] = max(0 , K - s.vec[s])
}

# Initializing array containing values for coefficients of previous
# time step to compute value of options
A <- array(data = NA, dim = c(NS, NV, NT+1))
B <- array(data = NA, dim = c(NS, NV, NT+1))
C <- array(data = NA, dim = c(NS, NV, NT+1))
D <- array(data = NA, dim = c(NS, NV, NT+1))
E <- array(data = NA, dim = c(NS, NV, NT+1))
G <- array(data = NA, dim = c(NS, NV, NT+1))
for(t in 1:(NT+1)) {
  for(s in 1:NS) {
    for(v in 1:NV) {
      A[1, v, t] = 0; B[1, v, t] = 0; C[1, v, t] = 0
      D[1, v, t] = 0; E[1, v, t] = 0; G[1, v, t] = 0

      A[NS, v, t] = 0; B[NS, v, t] = 0; C[NS, v, t] = 0
      D[NS, v, t] = 0; E[NS, v, t] = 0; G[NS, v, t] = 0
```

```r
      A[s, 1, t] = 0; B[s, 1, t] = 0; C[s, 1, t] = 0
      D[s, 1, t] = 0; E[s, 1, t] = 0; G[s, 1, t] = 0

      A[s, NV, t] = 0; B[s, NV, t] = 0; C[s, NV, t] = 0
      D[s, NV, t] = 0; E[s, NV, t] = 0; G[s, NV, t] = 0

      A[s, v, 1] = 0; B[s, v, 1] = 0; C[s, v, 1] = 0
      D[s, v, 1] = 0; E[s, v, 1] = 0; G[s, v, 1] = 0
    }
  }
}

# Computing Value of options in the grid
for(t in 2:(NT+1)) {
  for(s in 1:NS) {
    for(v in 1:(NV-1)) {
      price.array[1, v, t] = K
      price.array[NS, v, t] = 0
    }
  }
  for(l in 1:NS) {
    price.array[l, NV, t] = max(0, K - s.vec[l])
  }
  for(i in 2:(NS-1)) {
    price.array[i,1,t] = price.array[i,1,t-1]*(1 - r*dt - kappa*theta*dt/dv)
    + dt*0.5*r*s.vec[i]*(price.array[i+1,1,t-1] - price.array[i-1,1,t-1])/ds +
      kappa*theta*price.array[i,2,t-1]*dt/dv
  }
  for(j in 2:(NS-1)) {
    for(k in 2:(NV-1)) {
      A[j,k,t] = (1 - dt*(j-1)^2*(k-1)*dv - sig^2*(k-1)*dt/dv - r*dt)
      B[j,k,t] = (0.5*dt*(j-1)^2*(k-1)*dv - 0.5*dt*r*(j-1))
      C[j,k,t] = (0.5*dt*(j-1)^2*(k-1)*dv + 0.5*dt*r*(j-1))
      D[j,k,t] = (0.5*dt*sig^2*(k-1)/dv - 0.5*dt*kappa*(theta-(k-1)*dv)/dv)
      E[j,k,t] = (0.5*dt*sig^2*(k-1)/dv + 0.5*dt*kappa*(theta-(k-1)*dv)/dv)
      G[j,k,t] = 0.25*dt*sig*(j-1)*(k-1)
      price.array[j,k,t] = A[j,k,t]*price.array[j,k,t-1] + B[j,k,t]*price.array[j-1,k,t-1] +
        C[j,k,t]*price.array[j+1,k,t-1] + D[j,k,t]*price.array[j,k-1,t-1] +
        E[j,k,t]*price.array[j,k+1,t-1] +
        G[j,k,t]*( price.array[j+1,k+1,t-1]+price.array[j-1,k-1,t-1]-
              price.array[j-1,k+1,t-1]-price.array[j+1,k-1,t-1] )
    }
  }
}

# Using information obtained from the grid, use interpolation to compute value of the put option
interp2(sig.vec, s.vec, as.matrix(price.array[,,2001]), sig , s0, method = "linear")
```

```
## [1] 16.27213
```

So, the value of the put option is 16.27 USD.