MGMTMFE 405 - Project 7

Yanxiang Zhao

February 28, 2019

Set up

```
source("finiteDiff.r")
source("bsm.r")
```

Problem 1

For this problem, I used the $\Delta x = \sigma\sqrt{\Delta t}$.

```
# set parameters
s0 <- 10
k <- 10
r <- 0.04
sig <- 0.2
t <- 0.5
dt <- 0.002
dx <- sig*sqrt(dt)
```

i. Find the values:

```
# i. Values
# (a) Explicit Finite-Difference Method
Pa <- efd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
# (b) Implicit Finite-Difference Method
Pb <- ifd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
# (c) Crank-Nicolson Finite-Difference Method
Pc <- cnfd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
# output
c(EFD = Pa, IFD = Pb, CNFE = Pc)
```

```
##       EFD       IFD      CNFE
## 0.4641262 0.4641415 0.4644212
```

i. Compare the errors against BSM value:

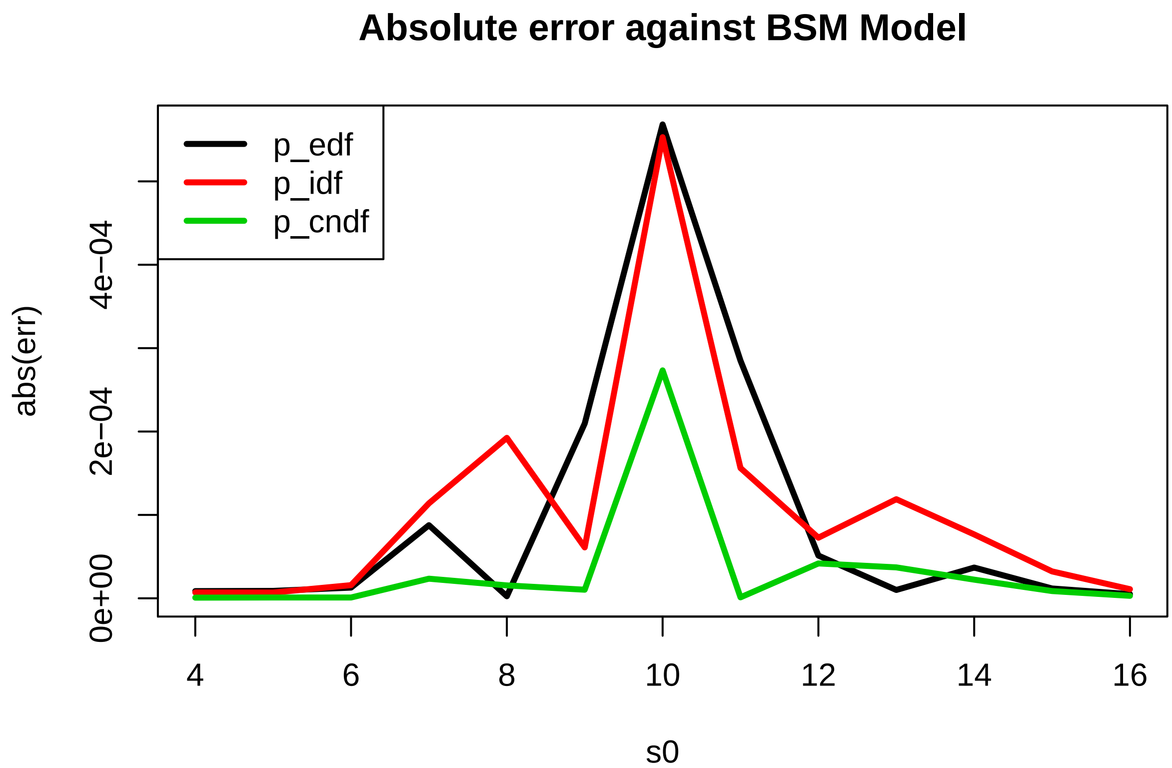
```
# ii. Comparison
s0 <- 4:16
p_edf <- p_idf <- p_cndf <- p_bs <- vector()
for (i in 1:length(s0)){
```

```r
  p_edf[i] <- efd(type="put", euro=T, s0[i], k, r, sig, t, dt, dx, log=T)
  p_idf[i] <- ifd(type="put", euro=T, s0[i], k, r, sig, t, dt, dx, log=T)
  p_cndf[i] <- cnfd(type="put", euro=T, s0[i], k, r, sig, t, dt, dx, log=T)
  p_bs[i] <- bsPut(s0[i], t, k, r, sig)
}
err <- cbind(p_edf, p_idf, p_cndf)-p_bs
rownames(err) <- s0
err
```

```
##            p_edf         p_idf        p_cndf
## 4  -8.642015e-06  7.041201e-06 -8.001019e-07
## 5  -8.876637e-06  6.928870e-06 -9.790835e-07
## 6  -1.281608e-05  1.575986e-05  9.549684e-07
## 7  -8.767709e-05  1.138702e-04  2.350953e-05
## 8   2.465697e-06  1.924229e-04  1.548799e-05
## 9  -2.096753e-04 -6.101081e-05 -1.030161e-05
## 10 -5.682852e-04 -5.530138e-04 -2.733570e-04
## 11  2.849895e-04 -1.562213e-04  1.151767e-06
## 12 -5.125694e-05  7.267623e-05  4.182273e-05
## 13 -1.002024e-05  1.188376e-04  3.707275e-05
## 14 -3.683900e-05  7.659731e-05  2.236712e-05
## 15 -1.175463e-05  3.224649e-05  8.693956e-06
## 16 -4.975808e-06  1.094352e-05  2.959839e-06
```

From the table above, we see that the errors are really small over all. The explicit finite-difference method has the highest sum of squred errors (4.603896e-07). The implicit finite-difference method has the second highest sum of squred errors (4.107216e-07). And the Crank-Nicolson finite-difference method has the lowest sum of squred errors (7.933473e-08). Overall, the Crank-Nicolson's methond is better than the other two finite-difference methods.

```r
# plot
matplot(s0, abs(err), type ="l", lwd = 3, lty=1, main = "Absolute error against BSM Model")
legend("topleft", legend = colnames(err), lwd = 3, col = 1:3)
```



To further understand the estimate errors, I plot the absolute values of the errors agains the stock price. The graph above shows that the estimate errors are the highest when the options are at the money. The errors becomes lower as the options move further in-the-money or out-of-the-money. From the graph, it also shows that the Crank-NNicolson's method has the overall lowest errors.

Problem 2

For this problem, I used $\Delta S = 0.25$.

```
# set parameters
s0 <- 10
k <- 10
r <- 0.04
sig <- 0.2
t <- 0.5
dt <- 0.002
ds <- 0.25
# i. Values
# (a) Explicit Finite-Difference Method
Pa <- efd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
# (b) Implicit Finite-Difference Method
Pb <- ifd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
# (c) Crank-Nicolson Finite-Difference Method
Pc <- cnfd(type="put", euro=T, s0, k, r, sig, t, dt, dx, log=T)
```

```r
# output
c(EFD = Pa, IFD = Pb, CNFE = Pc)
```

```
##       EFD       IFD      CNFE
## 0.4641262 0.4641415 0.4644212
```

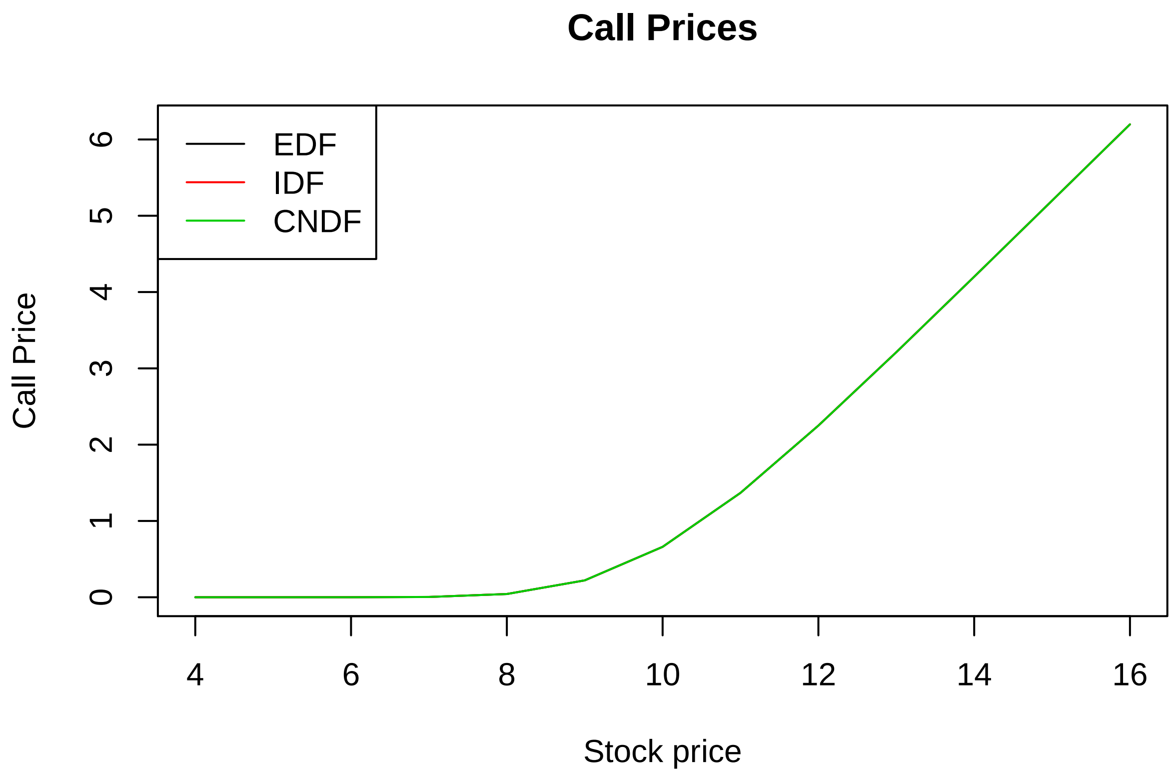
i. Find the values:

```r
# (a) Explicit Finite-Difference Method
Ca <- efd(type="call", euro=F, s0, k, r, sig, t, dt, ds, log=F)
Pa <- efd(type="put", euro=F, s0, k, r, sig, t, dt, ds, log=F)
# (b) Implicit Finite-Difference Method
Cb <- ifd(type="call", euro=F, s0, k, r, sig, t, dt, ds, log=F)
Pb <- ifd(type="put", euro=F, s0, k, r, sig, t, dt, ds, log=F)
# (c) Crank-Nicolson Finite-Difference Method
Cc <- cnfd(type="call", euro=F, s0, k, r, sig, t, dt, ds, log=F)
Pc <- cnfd(type="put", euro=F, s0, k, r, sig, t, dt, ds, log=F)
# output
cbind(EFD = c(call = Ca, put = Pa), IFD = c(Cb, Pb), CNFE = c(Cc, Pc))
```

```
##            EFD       IFD      CNFE
## call 0.6607978 0.6602173 0.6605078
## put  0.4806638 0.4797295 0.4801916
```

ii. Plot the values

```r
s0 <- 4:16
c_edf2 <- c_idf2 <- c_cndf2 <- vector()
p_edf2 <- p_idf2 <- p_cndf2 <- vector()
for (i in 1:length(s0)){
  # calls
  c_edf2[i] <- efd(type="call", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
  c_idf2[i] <- ifd(type="call", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
  c_cndf2[i] <- cnfd(type="call", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
  # puts
  p_edf2[i] <- efd(type="put", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
  p_idf2[i] <- ifd(type="put", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
  p_cndf2[i] <- cnfd(type="put", euro=F, s0[i], k, r, sig, t, dt, ds, log=F)
}
matplot(s0, cbind(c_edf2, c_idf2, c_cndf2), type ="l", lwd = 1, lty=1,
        main = "Call Prices", xlab = "Stock price", ylab = "Call Price")
legend("topleft", legend = c("EDF", "IDF", "CNDF"), lwd = 1, col = 1:3)
```




```r
matplot(s0, cbind(p_edf2, p_idf2, p_cndf2), type ="l", lwd = 1, lty=1,
        main = "Put Prices", xlab = "Stock price", ylab = "Put Price")
legend("topright", legend = c("EDF", "IDF", "CNDF"), lwd = 1, col = 1:3)
```


Put Prices
EDF
IDF
CNDF
Put Price
0
1
2
3
4
5
6
4
6
8
10
12
14
16
Stock price

Additional tables and plots

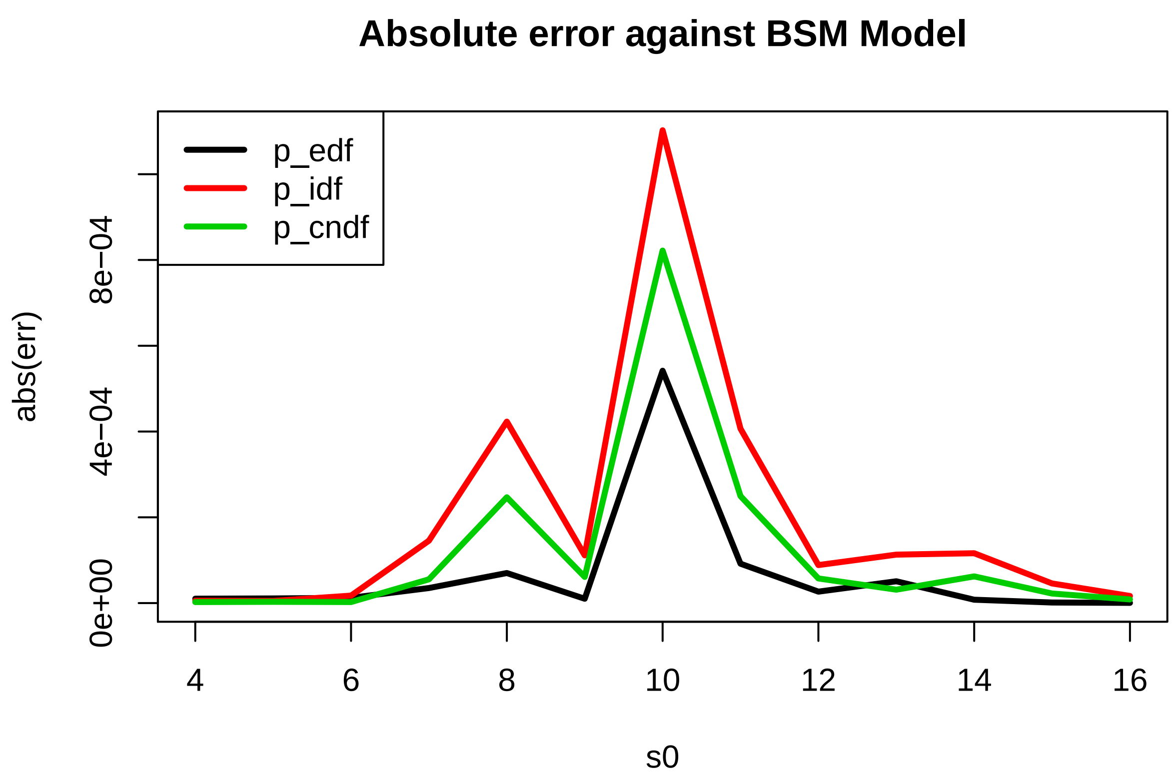
Problem 1

Now I used different Δx to test for the results. The errors get larger as the Δxs become larger. The **EFD method** becomes a better estimator with the larger Δx.

$\Delta x = \sigma\sqrt{3 * \Delta t}$:

```
##       EFD       IFD      CNFE
## 0.4641527 0.4635921 0.4638726
```

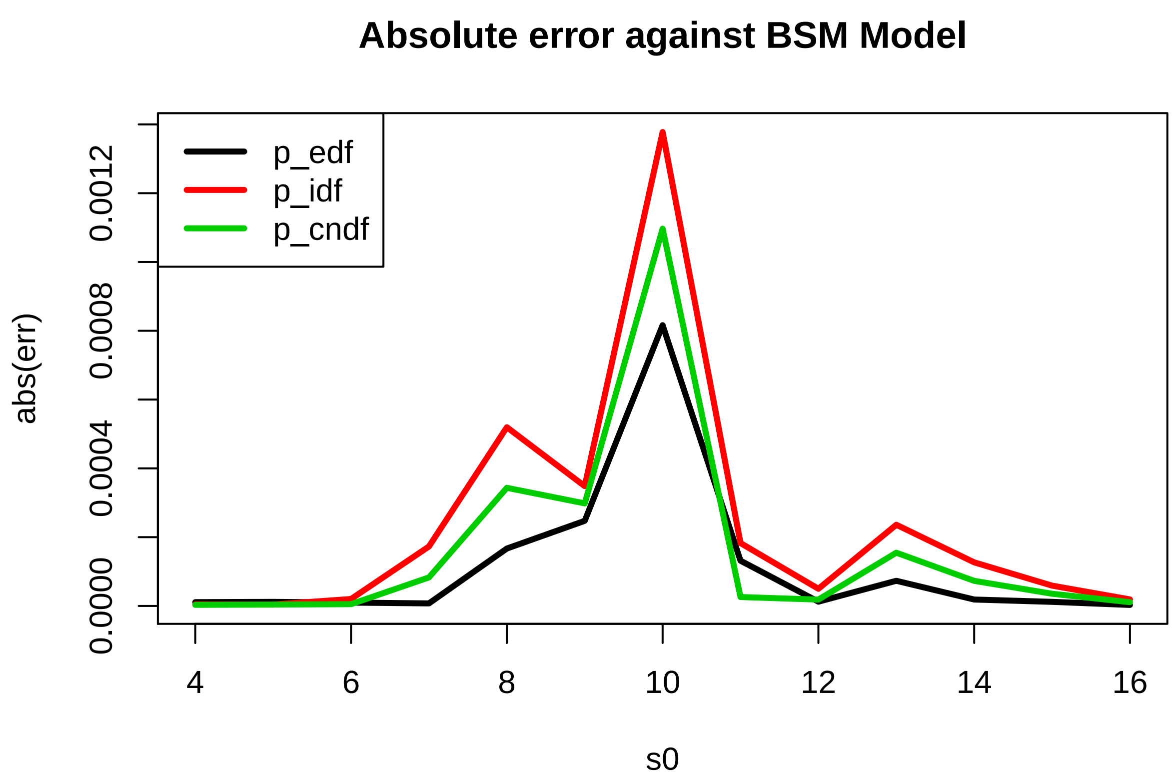
```
##            p_edf          p_idf         p_cndf
## 4  -1.024203e-05  5.441201e-06 -2.400112e-06
## 5  -1.084103e-05  4.979254e-06 -2.935936e-06
## 6  -1.228209e-05  1.732344e-05  2.431280e-06
## 7  -3.506266e-05  1.454598e-04  5.538562e-05
## 8   7.003246e-05  4.231160e-04  2.467109e-04
## 9  -1.006312e-05 -1.112234e-04 -6.095264e-05
## 10 -5.417927e-04 -1.102417e-03 -8.218885e-04
## 11 -9.181814e-05 -4.065652e-04 -2.494244e-04
## 12  2.668567e-05  8.859961e-05  5.742729e-05
## 13 -5.082038e-05  1.129493e-04  3.125184e-05
## 14  7.873586e-06  1.162051e-04  6.223119e-05
## 15 -1.315811e-06  4.573511e-05  2.224417e-05
## 16  5.333398e-07  1.654347e-05  8.502223e-06
```



$\Delta x = \sigma\sqrt{4 * \Delta t}$

```
##       EFD       IFD      CNFE
## 0.4638783 0.4633168 0.4635977
```

```
##             p_edf          p_idf         p_cndf
## 4  -1.104205e-05  4.641196e-06 -3.200123e-06
## 5  -1.182123e-05  4.006352e-06 -3.912575e-06
## 6  -9.647948e-06  2.022338e-05  5.199365e-06
## 7  -7.431167e-06  1.728377e-04  8.288851e-05
## 8   1.669199e-04  5.195894e-04  3.433930e-04
## 9  -2.474604e-04 -3.481886e-04 -2.981364e-04
## 10 -8.162538e-04 -1.377757e-03 -1.096787e-03
## 11  1.315507e-04 -1.827712e-04 -2.584288e-05
## 12 -1.245453e-05  4.988326e-05  1.849975e-05
## 13  7.312980e-05  2.360753e-04  1.547887e-04
## 14  1.868029e-05  1.267442e-04  7.290303e-05
## 15  1.190934e-05  5.900805e-05  3.549301e-05
## 16  2.940782e-06  1.899031e-05  1.092964e-05
```

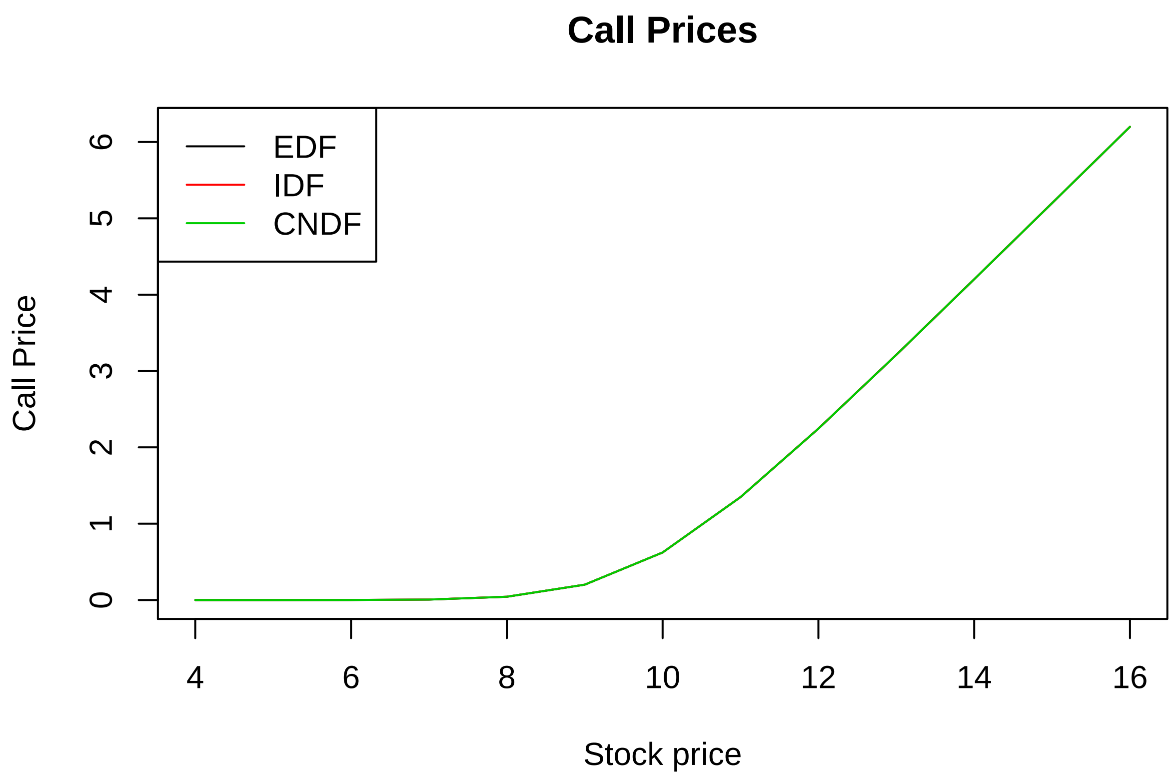


Problem 2

Now I used different ΔS to test for the results. From the plots, we can see that the errors get larger when the option moves to **out-of-the-money**.

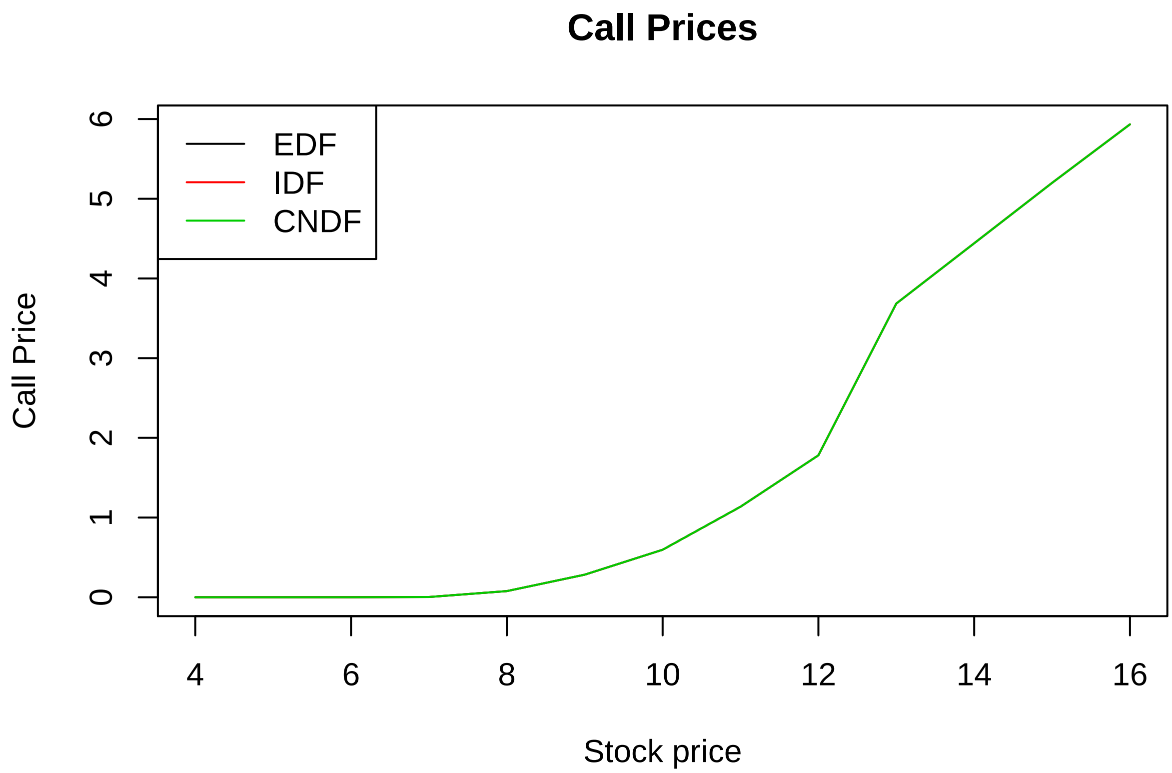
$\Delta S = 1$:

```
##             EFD       IFD      CNFE
## call 0.6230464 0.6222901 0.6226689
## put  0.4399789 0.4390812 0.4395300
```



$\Delta S = 1.25$

```
##             EFD       IFD      CNFE
## call 0.5963038 0.5955286 0.5959173
## put  0.4153277 0.4144274 0.4148772
```



Call Prices
EDF
IDF
CNDF
Call Price
0
1
2
3
4
5
6
4
6
8
10
12
14
16
Stock price

Code ## bsm.r

```
# Black-schole pricing model for European options
bsCall <- function(s0,t,x,r,sigma){
  # compute d1, d2
  d1 <- (log(s0/x)+(r+sigma^2/2)*t)/(sigma*sqrt(t))
  d2 <- d1-sigma*sqrt(t)
  # output: Black-Sholes price
  c_bs <- s0*pnorm(d1)-x*exp(-r*t)*pnorm(d2)
  return(c_bs)
}
bsPut <- function(s0,t,x,r,sigma){
  # compute d1, d2
  d1 <- (log(s0/x)+(r+sigma^2/2)*t)/(sigma*sqrt(t))
  d2 <- d1-sigma*sqrt(t)
  # output: Black-Sholes price
  p_bs <- x*exp(-r*t)*pnorm(-d2)-s0*pnorm(-d1)
  return(p_bs)
}
```

finiteDiff.r

```
# Explicit Finite-Difference Method
efd <- function(type, euro, s0, k, r, sig, t, dt, dsx, log){
  # if change in log price
```

```r
if (log) {
  n <- floor(log(s0)/dsx)
  # defind M and N
  M <- t/dt+1
  N <- 2*n+1
  # define dx
  dx <- dsx
  # find terminal conditions
  sT <- exp(c(log(s0)+dx*(n:-n)))
  # calculate pu, pm, pd
  pu <- dt*(sig^2/(2*dx^2) + (r-sig^2/2)/(2*dx))
  pm <- 1 - dt*sig^2/dx^2 - r*dt
  pd <- dt*(sig^2/(2*dx^2) - (r-sig^2/2)/(2*dx))
  # construct matrix A
  A <- diag(pm, N-2, N-2)
  diag(A[-1,]) <- pu
  diag(A[,-1]) <- pd
  # set offset constans
  b <- c(pu, pd)
} else { # if change in price
  # defind M
  M <- t/dt+1
  # define ds
  ds <- dsx
  # find terminal conditions
  sT <- seq(2*s0, 0, -ds)
  N <- length(sT)
  # calculate pu, pm, pd
  j <- (N-2):1
  pu <- 0.5*dt*(sig^2*j^2 + r*j)
  pm <- 1-dt*(sig^2*j^2+r)
  pd <- 0.5*dt*(sig^2*j^2 - r*j)
  # set A
  A <- diag(pm)
  diag(A[-1,]) <- pu[2:length(pu)]
  diag(A[,-1]) <- pd[1:length(pd)-1]
  # set offset constans
  b <- c(pu[1], pd[length(pd)])
}
# payoff matrix
payoff <- diag(0, N, M)
if (type == "call"){
  payoff[,M] <- ifelse(sT-k>0, sT-k, 0)
  payoff[1,] <- (max(sT)-k)*exp(-r*seq(t,0,-dt))
  payoff[N,] <- 0
}
else if (type == "put"){
  payoff[,M] <- ifelse(k-sT>0, k-sT, 0)
  payoff[1,] <- 0
  payoff[N,] <- (k-min(sT))*exp(-r*seq(t,0,-dt))
}
else{
  stop("Incorrect option type!")
```

```
  }
  # for loop through M
  for (i in (M-1):1){
    # if Europian option
    if (euro){
      payoff[2:(N-1),i] <- A%*%payoff[2:(N-1),i+1]
      payoff[2,i] <- payoff[2,i]+b[1]*payoff[1,i+1]
      payoff[N-1,i] <- payoff[N-1,i]+b[2]*payoff[N,i+1]
    } else { # if American option
      cv <- payoff[,i]
      cv[2:(N-1)] <- A%*%payoff[2:(N-1),i+1]
      cv[2] <- payoff[2,i]+b[1]*payoff[1,i+1]
      cv[N-1] <- payoff[N-1,i]+b[2]*payoff[N,i+1]
      payoff[,i] <- apply(cbind(payoff[,M], cv),1,max)
    }
  }
  return(payoff[floor(N/2)+1,1])
}
# Implicit Finite-Difference Method
ifd <- function(type, euro, s0, k, r, sig, t, dt, dsx, log){
  # if change in log price
  if (log) {
    n <- floor(log(s0)/dsx)
    # defind M and N
    M <- t/dt+1
    N <- 2*n+1
    # define dx
    dx <- dsx
    # find terminal conditions
    sT <- exp(c(log(s0)+dx*(n:-n)))
    # calculate pu, pm, pd
    pu = -0.5*dt*(sig^2/dx^2 + (r-sig^2/2)/dx)
    pm = 1 + dt*sig^2/dx^2 + r*dt
    pd = -0.5*dt*(sig^2/dx^2 - (r-sig^2/2)/dx)
    # construct matrix A
    A <- diag(pm, N-2, N-2)
    diag(A[-1,]) <- pu
    diag(A[,-1]) <- pd
    A <- rbind(c(1, -1, rep(0,N-2)),
               cbind(c(pu, rep(0,N-3)),A,c(rep(0,N-3), pd)),
               c(rep(0,N-2), 1, -1))
  } else { # if change in price
    # defind M
    M <- t/dt+1
    # define ds
    ds <- dsx
    # find terminal conditions
    sT <- seq(2*s0, 0, -ds)
    N <- length(sT)
    # calculate pu, pm, pd
    j <- (N-2):1
    pu <- -0.5*dt*(sig^2*j^2 + r*j)
    pm <- 1+dt*(sig^2*j^2+r)
```

```r
  pd <- 0.5*dt*(-sig^2*j^2 + r*j)
  # set A
  A <- diag(pm)
  diag(A[-1,]) <- pu[2:length(pu)]
  diag(A[,-1]) <- pd[1:length(pd)-1]
  # set offset constans
  b <- c(-pu[1], -pd[length(pd)])
}
# payoff matrix
payoff <- diag(0, N, M)
if (type == "call"){
  payoff[,M] <- ifelse(sT-k>0, sT-k, 0)
  payoff[1,] <- (max(sT)-k)*exp(-r*seq(t,0,-dt))
  payoff[N,] <- 0
}
else if (type == "put"){
  payoff[,M] <- ifelse(k-sT>0, k-sT, 0)
  payoff[1,] <- 0
  payoff[N,] <- (k-min(sT))*exp(-r*seq(t,0,-dt))
}
else{
  stop("Incorrect option type!")
}
# find A inverse
Ainv <- solve(A)
# find option price
if (log) {
  for (i in (M-1):1){
    # construct matrix B
    B <- payoff[,i+1]
    # if Europian option
    if (euro){
      payoff[,i] <- Ainv%*%B
    } else { # if American option
      cv <- Ainv%*%B
      payoff[,i] <- apply(cbind(payoff[,M], cv),1,max)
    }
  }
} else {
  # for loop through M
  for (i in (M-1):1){
    # construct matrix B
    B <- payoff[2:(N-1),i+1]
    B[1] <- B[1] + payoff[1,i+1]*b[1]
    B[length(N)] <- B[length(N)] + payoff[N,i+1]*b[2]
    # if Europian option
    if (euro){
      payoff[2:(N-1),i] <- Ainv%*%B
    } else { # if American option
      cv <- payoff[,i]
      cv[2:(N-1)] <- Ainv%*%B
      payoff[,i] <- apply(cbind(payoff[,M], cv),1,max)
    }
```

```
    }
  }
  return(payoff[floor(N/2)+1,1])
}
# Crank-Nicolson Finite-Difference Method
cnfd <- function(type, euro, s0, k, r, sig, t, dt, dsx, log){
  # if change in log price
  if (log) {
    n <- floor(log(s0)/dsx)
    # defind M and N
    M <- t/dt+1
    N <- 2*n+1
    # define dx
    dx <- dsx
    # find terminal conditions
    sT <- exp(c(log(s0)+dx*(n:-n)))
    # calculate pu, pm, pd
    pu = -0.25*dt*(sig^2/dx^2 + (r-sig^2/2)/dx)
    pm = 1 + dt*sig^2/(2*dx^2) + r*dt/2
    pd = -0.25*dt*(sig^2/dx^2 - (r-sig^2/2)/dx)
    # construct matrix A
    A <- diag(pm, N-2, N-2)
    diag(A[-1,]) <- pu
    diag(A[,-1]) <- pd
    A <- rbind(c(1, -1, rep(0,N-2)),
               cbind(c(pu, rep(0,N-3)),A,c(rep(0,N-3), pd)),
               c(rep(0,N-2), 1, -1))
  } else { # if change in price
    # defind M
    M <- t/dt+1
    # define ds
    ds <- dsx
    # find terminal conditions
    sT <- seq(2*s0, 0, -ds)
    N <- length(sT)
    # calculate pu, pm, pd
    j <- (N-2):1
    pu <- 0.25*dt*(sig^2*j^2 + r*j)
    pm <- -0.5*dt*(sig^2*j^2+r)
    pd <- 0.25*dt*(sig^2*j^2 - r*j)
    # set C
    A <- diag(1-pm)
    diag(A[-1,]) <- -pu[2:length(pu)]
    diag(A[,-1]) <- -pd[1:length(pd)-1]
    # set D
    D <- diag(1+pm)
    diag(D[-1,]) <- pu[2:length(pu)]
    diag(D[,-1]) <- pd[1:length(pd)-1]
    # set offset constans
    b <- c(pu[1], pd[length(pd)])
  }
  # payoff matrix
  payoff <- diag(0, N, M)
```

```r
  if (type == "call"){
    payoff[,M] <- ifelse(sT-k>0, sT-k, 0)
    payoff[1,] <- (max(sT)-k)*exp(-r*seq(t,0,-dt))
    payoff[N,] <- 0
  }
  else if (type == "put"){
    payoff[,M] <- ifelse(k-sT>0, k-sT, 0)
    payoff[1,] <- 0
    payoff[N,] <- (k-min(sT))*exp(-r*seq(t,0,-dt))
  }
  else{
    stop("Incorrect option type!")
  }
  # find A inverse
  Ainv <- solve(A)
  # find option price
  if (log) {
    for (i in (M-1):1){
      # construct matrix B
      B <- payoff[,i+1]
      B[2:(N-1)] <- -pu*payoff[1:(N-2),i+1] - (pm-2)*payoff[2:(N-1),i+1] - pd*payoff[3:N,i+1]
      # if European option
      if (euro){
        payoff[,i] <- Ainv%*%B
      } else { # if American option
        cv <- Ainv%*%B
        payoff[,i] <- apply(cbind(payoff[,M], cv),1,max)
      }
    }
  } else {
    # for loop through M
    for (i in (M-1):1){
      # construct matrix B
      B <- D%*%payoff[2:(N-1),i+1]
      B[1] <- B[1] + payoff[1,i+1]*b[1] + payoff[1,i]*b[1]
      B[length(N)] <- B[length(N)] + payoff[N,i+1]*b[2] + payoff[N,i+1]*b[2]
      # if European option
      if (euro){
        payoff[2:(N-1),i] <- Ainv%*%B
      } else { # if American option
        cv <- payoff[,i]
        cv[2:(N-1)] <- Ainv%*%B
        payoff[,i] <- apply(cbind(payoff[,M], cv),1,max)
      }
    }
  }
  return(payoff[floor(N/2)+1,1])
}
```