FE621 HW2

Shihao Zhang

2018-09-30

Problem 1: Binomial/Trinomial Tree Basics

```
#1.Using additive binomiall tree
#Calculate the values of the European Call and put option
#The difference between call and put option is the strike price,
#I use a factor cp which represent 1 or -1, in that way
#to calculate both options.
Binomial_tree <- function(isCall, isAmerican, K, Tm, S0, sig, r=0.0192,
                          N=200)
  {
  #Precompute constants
  dt <-  Tm/N
  v <-  r-0.5*sig^2
  u <-  sqrt(sig^2*dt+(v*dt)^2) #dxu
  d <- -u
  p <- 0.5+0.5*(v*dt/u) #probability
  disc <- exp(-r*dt) #discount factor
  M <- N+1
  cp <- ifelse(isCall, 1, -1) #factor cp with 1 or -1

  #Intialize asset prices
  V = S = matrix(0, nrow = M, ncol = M, dimnames = list(
    paste("State", 1:(N+1), sep = ""), paste("T=",0:N, sep = "")))
  S[1,1] <- S0
  for (j in 2:M) {
    S[1,j] <- S[1,j-1]*exp(u)
    for (i in 2:j) {
      S[i,j] <- S[i-1,j-1]*exp(d)
    }
  }

  #Intialize option values at maturity
  for (j in 1:M) {
    V[M-j+1,M] <- max(0, cp*(S[M-j+1,M]-K))
  }

  #Step backwards through the tree
  for (j in (M-1):1) {
    for (i in 1:j) {
      V[i,j] <- disc*(p*V[i,j+1]+(1-p)*V[i+1,j+1])
      if(isAmerican){
        V[i,j] <- max(V[i,j],cp*(S[i,j]-K))
      }
    }
  }
  #Return the price
  return(V[1,1])
```

```
}
Binomial_tree(isCall = T,isAmerican = F, K=100, Tm=1, S0=100, sig=0.2,
              r=0.06,N=3)
```

```
## [1] 11.59199
```

```
#Here I use the same parameter in figure 2.11 in textbook to check the model
#And I obtain the same result as 11.5920
```

2.Download Option prices I choose AMZN's options. And the expiry date is 2018-10-19,2018-11-02,2018-10-26,separately.

```
library(TTR)
library(quantmod)
```

```
## Warning: package 'quantmod' was built under R version 3.4.4
```

```
## Loading required package: xts
```

```
## Warning: package 'xts' was built under R version 3.4.4
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 3.4.4
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Version 0.4-0 included new data defaults. See ?getSymbols.
```

```
Aequity=function(symbol){
  A_equity <- getSymbols(symbol,src="yahoo",from="2018-09-28",to="2018-10-03",auto.assign = FALSE)
}
AMZN_price <- Aequity("AMZN")
```

```
## 'getSymbols' currently uses auto.assign=TRUE by default, but will
## use auto.assign=FALSE in 0.5-0. You will still be able to use
## 'loadSymbols' to automatically load data. getOption("getSymbols.env")
## and getOption("getSymbols.auto.assign") will still be checked for
## alternate defaults.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.warning4.0"=FALSE). See ?getSymbols for details.
```

```
##
## WARNING: There have been significant changes to Yahoo Finance data.
## Please see the Warning section of '?getSymbols.yahoo' for details.
##
## This message is shown once per session and may be disabled by setting
## options("getSymbols.yahoo.warning"=FALSE).
```

```
A_price <- AMZN_price$AMZN.Adjusted[3]
AM1=getOptionChain("AMZN",Exp = '2018-10-19',src = "yahoo",auto.assign = FALSE)#Expiry 2018-10-19 #Expi
AM2=getOptionChain("AMZN",Exp = '2018-11-02',src = "yahoo",auto.assign = FALSE)#Expiry 2018-11-02 #Expi
AM3=getOptionChain("AMZN",Exp = '2018-10-26',src = "yahoo",auto.assign = FALSE)#Expiry 2018-10-26 #Expi

# Black-Scholes Option Value
```

```r
BSmodel <- function(S0, Sigma, t, K, r, optionType){
  d1 = (log(S0/K) + (r + (Sigma^2)/2) * t)/(Sigma * sqrt(t))
  d2 = d1 - Sigma * sqrt(t)
  if(optionType == 'call'){
    call = S0 * pnorm(d1) - K * exp(-r * t) * pnorm(d2)
    return(call)
  }else if(optionType == "put"){
    put = K * exp(-r * t) * pnorm(-d2) - S0 * pnorm(-d1)
    return(put)
  }
}
#Implement the Bisection method
vrange=c(0,1)
t1=0
bisection=function(f, vrange, tol){
  while((vrange[2]-vrange[1]) >= tol){
    x=0.5*(vrange[1]+vrange[2])
if(f(x)*f(vrange[2])>0)
  {
  vrange[2]=x
}else if(f(x)*f(vrange[2])<0){
  vrange[1]=x
}
  t1=1+t1
}
return(x)
}

# Using bisection and BS model
# Calculate the implied volatility for AM_option1
r <- 0.0216 #the risk free rate
AMcall1=AM1$calls$Strike[1:20]
AMimpcall1=AMimpcall2=AMimpcall3=AMimput1=AMimput2=AMimput3=c()
# Call
for(i in 1:length(AMcall1)){
  AMvolc1 = function(sigma){
    BSmodel(A_price, sigma, 17/252,AMcall1[i], r, "call") - 0.5 * (AM1$calls$Bid[i] +AM1$calls$Ask[i])
  }
  AMimpcall1[i] = bisection(AMvolc1, vrange, 1e-2)
}
#Put
AMput1=AM1$puts$Strike[1:20]
for(i in 1:length(AMput1)){
  AMvolp1 = function(sigma){
    BSmodel(A_price, sigma, 17/252,AMput1[i], r, "put") - 0.5 * (AM1$puts$Bid[i] + AM1$puts$Ask[i])
  }
  AMimput1[i] = bisection(AMvolp1, vrange, 1e-2)
}

# Second Option
AMcall2=AM2$calls$Strike[1:20]

# Call
```

```
for(i in 1:length(AMcall2)){
  AMvolc2 = function(sigma){
    BSmodel(A_price, sigma, 31/252,AMcall2[i], r, "call") - 0.5 * (AM2$calls$Bid[i] + AM2$calls$Ask[i])
  }
  AMimpcall2[i] = bisection(AMvolc2, vrange, 1e-6)
}

#Put
AMput2=AM2$puts$Strike[1:20]
for(i in 1:length(AMput2)){
  AMvolp2 = function(sigma){
    BSmodel(A_price, sigma, 31/252,AMput2[i], r, "put") - 0.5 * (AM2$puts$Bid[i] + AM2$puts$Ask[i])
  }
  AMimput2[i] = bisection(AMvolp2, vrange, 1e-6)
}

# Third Option

AMcall3=AM3$calls$Strike[1:20]

# Call
for(i in 1:length(AMcall3)){
  AMvolc3 = function(sigma){
    BSmodel(A_price, sigma, 24/252,AMcall3[i], r, "call") - 0.5 * (AM3$calls$Bid[i] + AM3$calls$Ask[i])
  }
  AMimpcall3[i] = bisection(AMvolc3, vrange, 1e-6)
}

#Put
AMput3=AM3$puts$Strike[1:20]
for(i in 1:length(AMput3)){
  AMvolp3 = function(sigma){
    BSmodel(A_price, sigma, 24/252,AMput3[i], r, "put") - 0.5 * (AM3$puts$Bid[i] + AM3$puts$Ask[i])
  }
  AMimput3[i] = bisection(AMvolp3, vrange, 1e-6)
}


# Using BS Model and Binomial tree calculating option price
TTM1=17/252
TTM2=31/252
TTM3=24/252

# By BS model
r <- 0.0216 #the risk free rate
Acall1=Acall2=Acall3=Aput1=Aput2=Aput3=c()
# First AMZN Option
for (i in 1:length(AMcall1)){
  Acall1[i] = BSmodel(A_price, AMimpcall1[i], TTM1, AMcall1[i], r, "call")
}

for (i in 1:length(AMput1)){
  Aput1[i] = BSmodel(A_price, AMimput1[i], TTM1, AMput1[i], r, "put")
```

```
}
#Second AMZN Option
for (i in 1:length(AMcall2)){
  Acall2[i] = BSmodel(A_price, AMimpcall2[i], TTM2, AMcall2[i], r, "call")
}

for (i in 1:length(AMput2)){
  Aput2[i] = BSmodel(A_price, AMimput2[i], TTM2, AMput2[i], r, "put")
}
#Third Amzn Option
for (i in 1:length(AMcall3)){
  Acall3[i] = BSmodel(A_price, AMimpcall3[i], TTM3, AMcall3[i], r, "call")
}

for (i in 1:length(AMput3)){
  Aput3[i] = BSmodel(A_price, AMimput3[i], TTM3, AMput3[i], r, "put")
}

#By Binomial Tree
r <- 0.0216 #the risk free rate
N <- 200
BiAcall1=BiAcall2=BiAcall3=BiAput1=BiAput2=BiAput3=c() #Names Options by Binomial
# first option
for (i in 1:length(AMcall1)){
  BiAcall1[i] = Binomial_tree(isCall=T, isAmerican=F, AMcall1[i], TTM1, A_price, AMimpcall1[i], r,N)
}
for (i in 1:length(AMput1)){
  BiAput1[i] = Binomial_tree(isCall=F, isAmerican=F, AMput1[i], TTM1, A_price, AMimput1[i], r,N)
}
#Second option
for (i in 1:length(AMcall2)){
  BiAcall2[i] = Binomial_tree(isCall=T, isAmerican=F, AMcall2[i], TTM2, A_price, AMimpcall2[i], r,N)
}
for (i in 1:length(AMput2)){
  BiAput2[i] = Binomial_tree(isCall=F, isAmerican=F, AMput2[i], TTM2, A_price, AMimput2[i], r,N)
}
#Third option
for (i in 1:length(AMcall3)){
  BiAcall3[i] = Binomial_tree(isCall=T, isAmerican=F, AMcall3[i], TTM3, A_price, AMimpcall3[i], r,N)
}
for (i in 1:length(AMput3)){
  BiAput3[i] = Binomial_tree(isCall=F, isAmerican=F, AMput3[i], TTM3, A_price, AMimput3[i], r,N)
}
#Combine in a table
# Create a table to compare call and put by two methods
table_call <- data.frame(Acall1, BiAcall1,Acall2,BiAcall2,Acall3,BiAcall3)
table_put <- data.frame(Aput1,BiAput1,Aput2,BiAput2,Aput3,BiAput3)
table_call
```

```
##       Acall1  BiAcall1   Acall2 BiAcall2   Acall3 BiAcall3
## 1  1232.3876 1232.3876 567.8499 567.8847 648.2000 648.1723
## 2  1192.4458 1192.4458 508.9001 508.9702 624.0843 624.0843
## 3  1172.4749 1172.4749 421.1999 421.2479 606.2001 606.0524
## 4  1152.5040 1152.5040 375.5558 375.5558 549.0000 549.0131
```

```
## 5  1132.5332 1132.5332 385.0499 384.9652 494.3515 494.3515
## 6  1112.5623 1112.5623 363.4499 363.4334 490.9000 490.8458
## 7  1092.5914 1092.5914 354.0000 353.9040 410.5500 410.5199
## 8  1072.6205 1072.6205 344.6249 344.5278 401.0001 400.9562
## 9  1052.6497 1052.6497 300.8000 300.8254 409.1499 409.1849
## 10 1032.6788 1032.6788 269.9001 269.9598 399.5501 399.6201
## 11 1012.7079 1012.7079 245.2749 245.3673 335.6000 335.6521
## 12  992.7370  992.7370 186.0600 186.0600 315.9500 315.9821
## 13  972.7661  972.7661 213.8999 213.9593 284.7831 284.7831
## 14  952.7953  952.7953 220.0002 220.0172 313.3501 313.3978
## 15  932.8244  932.8244 189.3750 189.2689 303.7999 303.8846
## 16  912.8535  912.8535 181.5001 181.5504 254.8447 254.8447
## 17  892.8826  892.8826 173.9248 174.0240 285.4001 285.4712
## 18  872.9117  872.9117 127.4501 127.4768 260.0249 260.0715
## 19  852.9409  852.9409 172.1249 172.2449 251.2499 251.2224
## 20  832.9700  832.9700 152.7002 152.7891 204.9475 204.9475
```

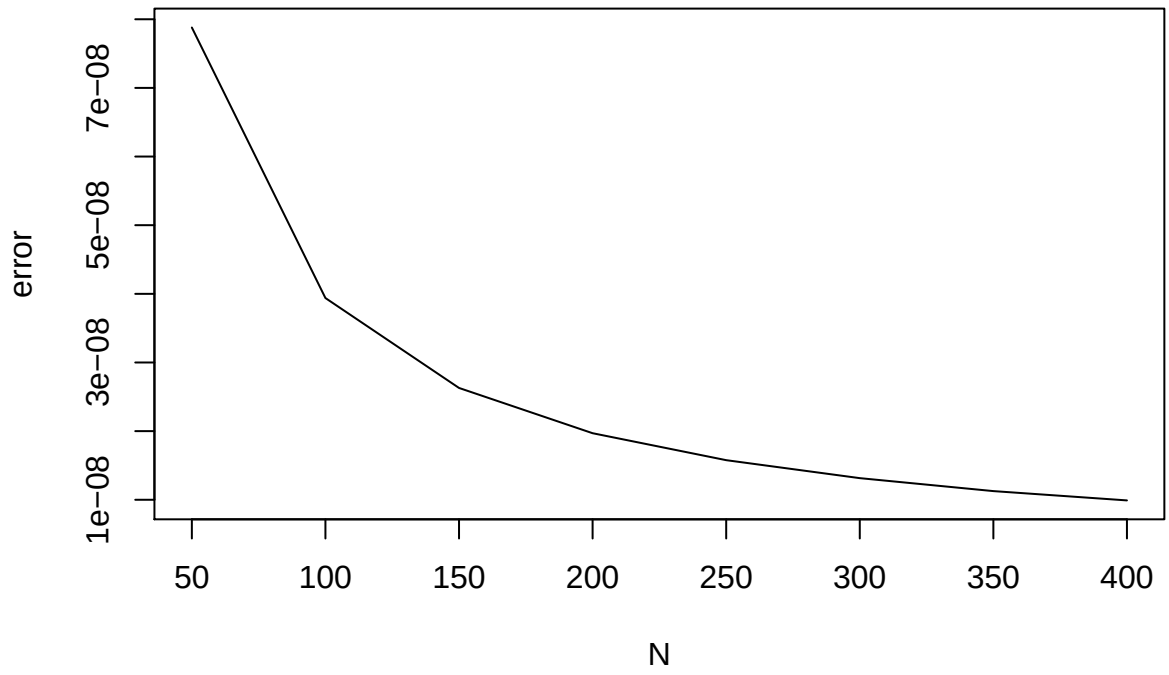
```
table_put#A is calculated by BSmodel, BiA is calculated by Binomiall Tree
```

```
##         Aput1    BiAput1    Aput2  BiAput2     Aput3   BiAput3
## 1  0.00000000 0.00000000 1.499995 1.487500 0.3050031 0.3003127
## 2  0.00000000 0.00000000 2.829993 2.805315 0.2550015 0.2471654
## 3  0.00000000 0.00000000 2.900011 2.888309 0.2800054 0.2750145
## 4  0.00000000 0.00000000 1.029987 1.010750 2.4250143 2.4002099
## 5  0.00000000 0.00000000 1.140009 1.131991 2.4499764 2.4187041
## 6  0.00000000 0.00000000 1.270012 1.262105 0.4699949 0.4646968
## 7  0.00000000 0.00000000 1.594989 1.581516 0.5150023 0.5079541
## 8  0.12257012 0.11744563 4.550008 4.546509 0.6600027 0.6515133
## 9  0.04414127 0.04115494 1.785021 1.778988 2.8049761 2.7992851
## 10 0.19106034 0.18466501 1.924987 1.910301 2.5000281 2.4806500
## 11 0.00000000 0.00000000 5.474994 5.449326 1.2949861 1.2805442
## 12 0.77912061 0.76937278 2.229995 2.212448 1.1200103 1.1136416
## 13 0.23118632 0.22603393 2.469979 2.466211 1.2399856 1.2242483
## 14 0.31244373 0.30074671 2.565015 2.559847 1.2750136 1.2524525
## 15 0.13781900 0.13439026 1.875011 1.846527 1.7349890 1.7267767
## 16 0.15000188 0.14591802 3.000017 2.962838 4.8999808 4.8489864
## 17 0.16288531 0.15683573 8.199977 8.211748 1.6599969 1.6520819
## 18 0.17648854 0.16917516 8.624999 8.633339 2.6150334 2.6040710
## 19 2.40257318 2.39428869 3.675030 3.668078 1.4549848 1.4503075
## 20 0.77893531 0.76170765 3.950008 3.917053 2.2950022 2.2914664
```

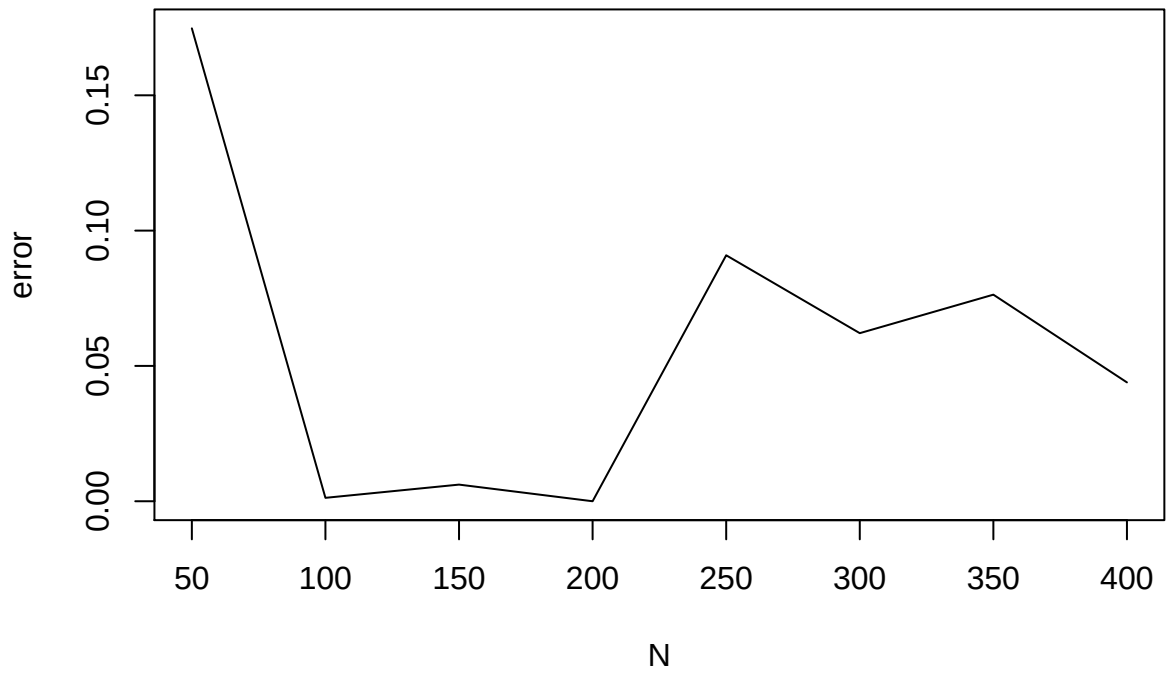
3.plot the absolute error

```
N <-  seq(50,400,by=50) #steps
error1 = c() #Calculate error
for (i in 1:length(N)) {
  error1[i] = abs(BSmodel(A_price, AMimpcall1[i], TTM1, AMcall1[i], r, "call")
                  -Binomial_tree(isCall=T, isAmerican=F, AMcall1[i], TTM1, A_price, AMimpcall1[i], r,N[i]
}
error2 = c() #Calculate error
for (i in 1:length(N)) {
  error2[i] = abs(BSmodel(A_price, AMimpcall2[i], TTM2, AMcall2[i], r, "call")
                  -Binomial_tree(isCall=T, isAmerican=F, AMcall2[i], TTM2, A_price, AMimpcall2[i], r,N[i]
}
error3 = c() #Calculate error
for (i in 1:length(N)) {
```

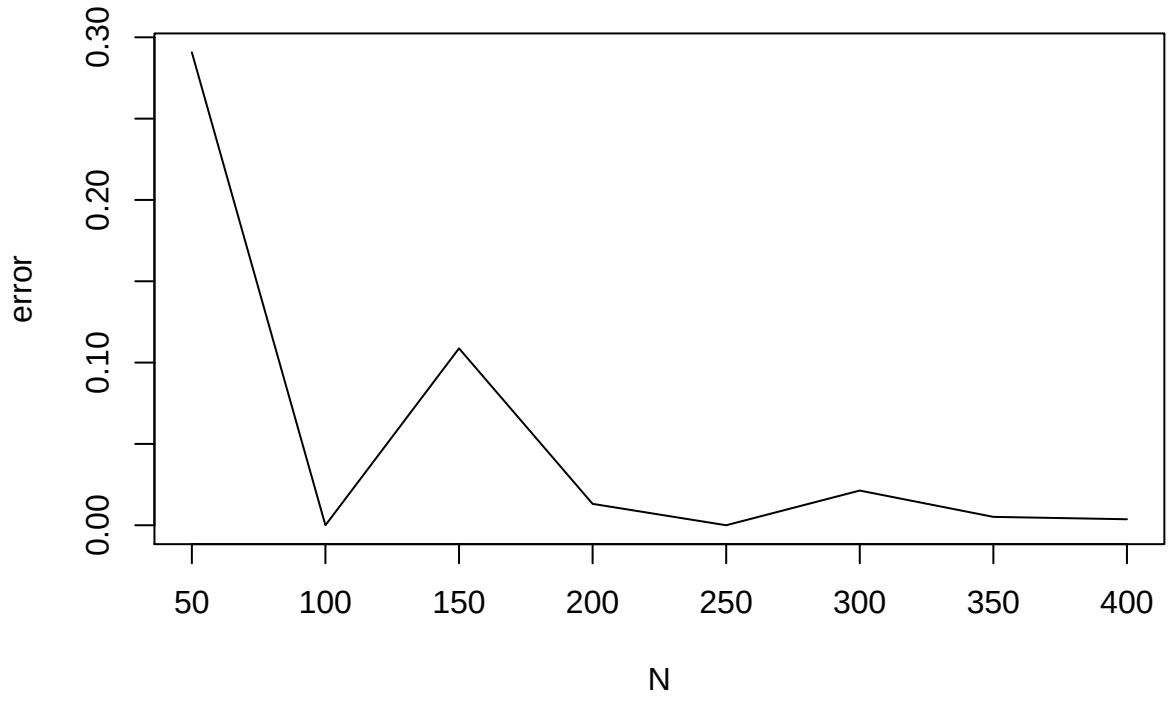
```
  error3[i] = abs(BSmodel(A_price, AMimpcall3[i], TTM3, AMcall3[i], r, "call")
                  -Binomial_tree(isCall=T, isAmerican=F, AMcall3[i], TTM3, A_price, AMimpcall3[i], r,N[i]
}
plot(N, error1, type="l", xlab = "N",ylab = "error")
```




```
plot(N, error2, type="l", xlab = "N",ylab = "error")
```




```
plot(N, error3, type="l", xlab = "N",ylab = "error")
```




```
#We can see that as the steps going up, the error will decrease.
#However, we still have some fluctuate in the middle steps.
```

4.Calculate the American option

```
#Implement an additive binomial tree
N <- 200
r <- 0.0216 #the risk free rate
BiAcall1_Am=BiAcall2_Am=BiAcall3_Am=BiAput1_Am=BiAput2_Am=BiAput3_Am=c()
# first option
for (i in 1:length(AMcall1)){
  BiAcall1_Am[i] = Binomial_tree(isCall=T, isAmerican=T, AMcall1[i], TTM1, A_price, AMimpcall1[i], r,N)
}
for (i in 1:length(AMput1)){
  BiAput1_Am[i] = Binomial_tree(isCall=F, isAmerican=T, AMput1[i], TTM1, A_price, AMimput1[i], r,N)
}
#Second option
for (i in 1:length(AMcall2)){
  BiAcall2_Am[i] = Binomial_tree(isCall=T, isAmerican=T, AMcall2[i], TTM2, A_price, AMimpcall2[i], r,N)
}
for (i in 1:length(AMput2)){
  BiAput2_Am[i] = Binomial_tree(isCall=F, isAmerican=T, AMput2[i], TTM2, A_price, AMimput2[i], r,N)
}
#Third option
for (i in 1:length(AMcall3)){
  BiAcall3_Am[i] = Binomial_tree(isCall=T, isAmerican=T, AMcall3[i], TTM3, A_price, AMimpcall3[i], r,N)
}
```

```
for (i in 1:length(AMput3)){
  BiAput3_Am[i] = Binomial_tree(isCall=F, isAmerican=T, AMput3[i], TTM3, A_price, AMimput3[i], r,N)
}
table_call_Am <- data.frame(BiAcall1_Am,BiAcall2_Am,BiAcall3_Am)
table_put_Am <- data.frame(BiAput1_Am,BiAput2_Am,BiAput3_Am)
head(table_call_Am)
```

```
##   BiAcall1_Am BiAcall2_Am BiAcall3_Am
## 1    1232.388    567.8847    648.1723
## 2    1192.446    508.9702    624.0843
## 3    1172.475    421.2479    606.0524
## 4    1152.504    375.5558    549.0131
## 5    1132.533    384.9652    494.3515
## 6    1112.562    363.4334    490.8458
```

```
head(table_put_Am)
```

```
##   BiAput1_Am BiAput2_Am BiAput3_Am
## 1          0   1.489095  0.3006023
## 2          0   2.808745  0.2474472
## 3          0   2.891819  0.2753028
## 4          0   1.012231  2.4021710
## 5          0   1.133519  2.4207187
## 6          0   1.263708  0.4651735
```

5.Create a table which contains Bid and Ask values, Black Scholes price, European and American prices

```
# Create Call table
table_call1 <- data.frame(AM1$calls$Bid[1:20], AM1$calls$Ask[1:20], Acall1, BiAcall1, BiAcall1_Am)
table_call2 <- data.frame(AM2$calls$Bid[1:20], AM2$calls$Ask[1:20], Acall2, BiAcall2, BiAcall2_Am)
table_call3 <- data.frame(AM3$calls$Bid[1:20], AM3$calls$Ask[1:20], Acall3, BiAcall3, BiAcall3_Am)
colnames(table_call1)=colnames(table_call2)=colnames(table_call3)=
  c("Call Bid", "Call Ask", "Call BS price", "European Call", "American Call")
table_call_all <- rbind(table_call1,table_call2,table_call3)
# Create Put table
table_put1 = data.frame(AM1$puts$Bid[1:20], AM1$puts$Ask[1:20], Aput1, BiAput1, BiAput1_Am)
table_put2 = data.frame(AM2$puts$Bid[1:20], AM2$puts$Ask[1:20], Aput2, BiAput2, BiAput2_Am)
table_put3 = data.frame(AM3$puts$Bid[1:20], AM3$puts$Ask[1:20], Aput3, BiAput3, BiAput3_Am)
colnames(table_put1)=colnames(table_put2)=colnames(table_put3)=
  c("Put Bid", "Put Ask", "Put BS price", "European Put", "American Put")
table_put_all <- rbind(table_put1,table_put2,table_put3)
#Merge in one table
table_all <- cbind(table_call_all,table_put_all)
#From the data, the price between BSmodel and Binomial Tree is really close.
#And the price is all between Bid and Ask.
#The price is more accurate when time to maturity is closing.
#And because all the 3 option I choose will expiry less than 31days.
#The call price between European and American option is same, while some difference in put price.
```

6.Compare implied volatility

```
#Implement Bisenction&Binomial Tree to find implied volatility for American Options
impcall1=impcall2=impcall3=imput1=imput2=imput3=c()
N <- 50
#First Option
#Call volatility
```

```r
for(i in 1:length(AMcall1)){
  volc1 <- function(sigma){
    Binomial_tree(isCall=T, isAmerican=T, AMcall1[i], TTM1, A_price, sigma, r,N)
    - 0.5 * (AM1$calls$Bid[i] + AM1$calls$Ask[i])
  }
  impcall1[i] = bisection(volc1, vrange, 1e-6)}
#Put volatility
for(i in 1:length(AMput1)){
  volp1 = function(sigma){
    Binomial_tree(isCall=F, isAmerican=T, AMput1[i], TTM1, A_price, sigma, r,N)
    - 0.5 * (AM1$puts$Bid[i] + AM1$puts$Ask[i])
  }
  imput1[i] = bisection(volp1, vrange, 1e-6)}
# Second Option
for(i in 1:length(AMcall2)){
  volc2 <- function(sigma){
    Binomial_tree(isCall=T, isAmerican=T, AMcall2[i], TTM2, A_price, sigma, r,N)
    - 0.5 * (AM2$calls$Bid[i] + AM2$calls$Ask[i])
  }
  impcall2[i] = bisection(volc2, vrange, 1e-6)}
for(i in 1:length(AMput2)){
  volp2 = function(sigma){
    Binomial_tree(isCall=F, isAmerican=T, AMput2[i], TTM2, A_price, sigma, r,N)
    - 0.5 * (AM2$puts$Bid[i] + AM2$puts$Ask[i])
  }
  imput2[i] = bisection(volp2, vrange, 1e-6)}
# Third Option
for(i in 1:length(AMcall3)){
  volc3 <- function(sigma){
    Binomial_tree(isCall=T, isAmerican=T, AMcall3[i], TTM3, A_price, sigma, r,N)
    - 0.5 * (AM3$calls$Bid[i] + AM3$calls$Ask[i])
  }
  impcall3[i] = bisection(volc3, vrange, 1e-6)}
#Put volatility
for(i in 1:length(AMput3)){
  volp3 = function(sigma){
    Binomial_tree(isCall=F, isAmerican=T, AMput3[i], TTM3, A_price, sigma, r,N)
    - 0.5 * (AM3$puts$Bid[i] + AM3$puts$Ask[i])
  }
  imput3[i] = bisection(volp3, vrange, 1e-6)}

#compare implied Vol by BS model and Binomial Tree model

#Call Implied Vol
table_call_BSBI <- data.frame(AMimpcall1,impcall1,AMimpcall2,impcall2,AMimpcall3,impcall3)
colnames(table_call_BSBI) <- c("BS Call1", "BiTree Call1", "BS Call2","BiTree Call2","BS Call3","BiTree
table_put_BSBI = data.frame(AMimput1,imput1,AMimput2,imput2,AMimput3,imput3)
colnames(table_put_BSBI) = c("BS Put1", "BiTree Put1","BS Put2", "BiTree Put2","BS Put3", "BiTree Put3"
table_call_BSBI#Call vol
```

```
##     BS Call1 BiTree Call1     BS Call2 BiTree Call2     BS Call3
## 1  0.0078125 9.536743e-07 8.185434e-01 9.536743e-07 8.668108e-01
## 2  0.0078125 9.536743e-07 7.465944e-01 9.536743e-07 9.536743e-07
## 3  0.0078125 9.536743e-07 6.450319e-01 9.536743e-07 9.832697e-01
```

```
## 4  0.0078125 9.536743e-07 9.536743e-07 9.536743e-07 7.426043e-01
## 5  0.0078125 9.536743e-07 5.070734e-01 9.536743e-07 9.536743e-07
## 6  0.0078125 9.536743e-07 5.814466e-01 9.536743e-07 6.862955e-01
## 7  0.0078125 9.536743e-07 5.717459e-01 9.536743e-07 4.678221e-01
## 8  0.0078125 9.536743e-07 5.623999e-01 9.536743e-07 4.640932e-01
## 9  0.0078125 9.536743e-07 4.428759e-01 9.536743e-07 7.090673e-01
## 10 0.0078125 9.536743e-07 4.020205e-01 9.536743e-07 6.974554e-01
## 11 0.0078125 9.536743e-07 4.711561e-01 9.536743e-07 5.230742e-01
## 12 0.0078125 9.536743e-07 9.536743e-07 9.536743e-07 4.998713e-01
## 13 0.0078125 9.536743e-07 3.904467e-01 9.536743e-07 9.536743e-07
## 14 0.0078125 9.536743e-07 4.509535e-01 9.536743e-07 5.919771e-01
## 15 0.0078125 9.536743e-07 3.758669e-01 9.536743e-07 5.801134e-01
## 16 0.0078125 9.536743e-07 3.715544e-01 9.536743e-07 9.536743e-07
## 17 0.0078125 9.536743e-07 3.680944e-01 9.536743e-07 5.603075e-01
## 18 0.0078125 9.536743e-07 1.824484e-01 9.536743e-07 4.500513e-01
## 19 0.0078125 9.536743e-07 4.139910e-01 9.536743e-07 4.446154e-01
## 20 0.0078125 9.536743e-07 3.609476e-01 9.536743e-07 9.536743e-07
##    BiTree Call3
## 1  9.536743e-07
## 2  9.536743e-07
## 3  9.536743e-07
## 4  9.536743e-07
## 5  9.536743e-07
## 6  9.536743e-07
## 7  9.536743e-07
## 8  9.536743e-07
## 9  9.536743e-07
## 10 9.536743e-07
## 11 9.536743e-07
## 12 9.536743e-07
## 13 9.536743e-07
## 14 9.536743e-07
## 15 9.536743e-07
## 16 9.536743e-07
## 17 9.536743e-07
## 18 9.536743e-07
## 19 9.536743e-07
## 20 9.536743e-07
```

```
table_put_BSBI#Put vol
```

```
##      BS Put1  BiTree Put1   BS Put2  BiTree Put2   BS Put3  BiTree Put3
## 1  0.0078125 9.536743e-07 0.5087252 9.536743e-07 0.4760637 9.536743e-07
## 2  0.0078125 9.536743e-07 0.5406561 9.536743e-07 0.4590788 9.536743e-07
## 3  0.0078125 9.536743e-07 0.5243235 9.536743e-07 0.4552984 9.536743e-07
## 4  0.0078125 9.536743e-07 0.4426298 9.536743e-07 0.5885363 9.536743e-07
## 5  0.0078125 9.536743e-07 0.4405241 9.536743e-07 0.5792379 9.536743e-07
## 6  0.0078125 9.536743e-07 0.4388781 9.536743e-07 0.4560976 9.536743e-07
## 7  0.0078125 9.536743e-07 0.4283323 9.536743e-07 0.4524317 9.536743e-07
## 8  0.8984375 9.536743e-07 0.4993296 9.536743e-07 0.4487333 9.536743e-07
## 9  0.8046875 9.536743e-07 0.4189138 9.536743e-07 0.5406351 9.536743e-07
## 10 0.8359375 9.536743e-07 0.4154329 9.536743e-07 0.5207033 9.536743e-07
## 11 0.0078125 9.536743e-07 0.4884081 9.536743e-07 0.4625750 9.536743e-07
## 12 0.9140625 9.536743e-07 0.4082804 9.536743e-07 0.4355421 9.536743e-07
## 13 0.7578125 9.536743e-07 0.4066401 9.536743e-07 0.4328299 9.536743e-07
```

```
## 14 0.7578125 9.536743e-07 0.4007254 9.536743e-07 0.4256258 9.536743e-07
## 15 0.6796875 9.536743e-07 0.3728991 9.536743e-07 0.4361734 9.536743e-07
## 16 0.6640625 9.536743e-07 0.3943644 9.536743e-07 0.5106707 9.536743e-07
## 17 0.6484375 9.536743e-07 0.4710073 9.536743e-07 0.4150820 9.536743e-07
## 18 0.6328125 9.536743e-07 0.4664984 9.536743e-07 0.4368792 9.536743e-07
## 19 0.8359375 9.536743e-07 0.3827372 9.536743e-07 0.3892031 9.536743e-07
## 20 0.6953125 9.536743e-07 0.3792009 9.536743e-07 0.4088049 9.536743e-07
```

```
#From Observation, for call volatility by the 2 models are very close.
#However, for put volatility, the Binomial Tree model provide more smaller volatility.
#I guess the reason is that the Binomial Tree is less sensitive to volatility.
```

7.Implement a trinomial tree to price an American Put option

```
#Implement a trinomial tree
Trinomial_tree <- function(isCall,isAmerican,S0,K,Tm,sig,r,div,N){
  #Precompute constants
  dt <- Tm/N
  dx <- sig*sqrt(3*dt)
  nu <- r - div - 0.5*sig^2
  edx <- exp(dx)
  pu <- 0.5*((sig^2*dt+nu^2*dt^2)/(dx^2)+nu*dt/dx)
  pd <- 0.5*((sig^2*dt+nu^2*dt^2)/(dx^2)-nu*dt/dx)
  pm <- 1 - pu - pd
  disc <- exp(-r*dt)
  #Call-Put Flag
  cp <- ifelse(isCall, 1, -1)
  #initialize asset prices
  S_mat <- matrix(ncol=2*N+1,nrow=N+1)
  S_mat[1,N+1]<-S0
  for (i in 1:N) {
    S_mat[i+1,(N+1-i):(N+1+i)]<- S0*(exp(((-i:i)*dx)))
  }
  #initialize option values
  V_mat <- matrix(ncol=2*N+1,nrow=N+1)
  V_mat[N+1,] <- pmax(cp*(S_mat[N+1,]-K),0)
  #step back through the tree
  for(i in N:1){
    V_mat[i,(N-i+2):(i+N)]<-disc*(pu*V_mat[i+1,(N-i+3):(i+N+1)]
                                  +pm*V_mat[i+1,(N-i+2):(i+N)]
                                  +pd*V_mat[i+1,(N-i+1):(i+N-1)])
    if(isAmerican){
      V_mat[i,(N-i+2):(i+N)] <- max(V_mat[i,(N-i+2):(i+N)],
                                    cp*(S_mat[i,(N-i+2):(i+N)]-K))
    }
    }
  return(Price=V_mat[1,N+1])
}
#Price an American Put option
#Given the data
N <- 3
r <- 0.0216 #the risk free rate
div <- 0
TriAput1_Am <- c()
for (i in 1:length(AMput1)){
```

```
  TriAput1_Am[i] = Trinomial_tree(isCall=F, isAmerican=F, A_price, AMput1[i], TTM1, AMimput1[i], r,div,I
}
```

```
## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithme
##   Use c() or as.vector() instead.
```

```
## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.
```

```
## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.
```

```
## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.

## Warning in `*.default`(S0, (exp(((-i:i) * dx)))): Recycling array of length 1 in array-vector arithm
##   Use c() or as.vector() instead.
```

```r
mytable <- cbind(TriAput1_Am,BiAput1_Am)
mytable#Compare with Binomial model
```

```
##       TriAput1_Am BiAput1_Am
##  [1,] 0.000000000 0.00000000
##  [2,] 0.000000000 0.00000000
##  [3,] 0.000000000 0.00000000
##  [4,] 0.000000000 0.00000000
##  [5,] 0.000000000 0.00000000
##  [6,] 0.000000000 0.00000000
##  [7,] 0.000000000 0.00000000
##  [8,] 0.007244654 0.11748588
##  [9,] 0.000000000 0.04117673
## [10,] 0.201519816 0.18473540
## [11,] 0.000000000 0.00000000
## [12,] 0.856263820 0.76967829
## [13,] 0.289006191 0.22614792
## [14,] 0.409988919 0.30087813
## [15,] 0.112611283 0.13446057
## [16,] 0.145531518 0.14599055
## [17,] 0.177007895 0.15691222
## [18,] 0.207048575 0.16929142
## [19,] 1.450026767 2.39548420
## [20,] 0.785797219 0.76219831
```

Problem 2.Adapting the Binomial tree to time varying coefficients

```r
#1
f1 <- function(x){
  0.05*(1+0.01*x)
}
f2 <- function(x){
  0.3*(1+0.005*x)
}
#Set coefficients
```

```r
i <- seq(0,11,1)
r <- sapply(i,f1)
sig <- sapply(i,f2)
v <- r-sig^2/2
dx <- 0.05 #choose dx

#Calculate delat_t,delta_p,and probabilities
dt <- (-sig^2+sqrt(sig^4+4*v^2*dx^2))/(2*v^2)
p <- 1/2+v*dt/(2*dx)
mytable <- cbind(i,r,sig,v,dt,p)
mytable
```

```
##        i      r    sig           v         dt         p
##  [1,]  0 0.0500 0.3000 0.005000000 0.02777756 0.5013889
##  [2,]  1 0.0505 0.3015 0.005048875 0.02750186 0.5013885
##  [3,]  2 0.0510 0.3030 0.005095500 0.02723024 0.5013875
##  [4,]  3 0.0515 0.3045 0.005139875 0.02696262 0.5013858
##  [5,]  4 0.0520 0.3060 0.005182000 0.02669893 0.5013835
##  [6,]  5 0.0525 0.3075 0.005221875 0.02643909 0.5013806
##  [7,]  6 0.0530 0.3090 0.005259500 0.02618302 0.5013771
##  [8,]  7 0.0535 0.3105 0.005294875 0.02593066 0.5013730
##  [9,]  8 0.0540 0.3120 0.005328000 0.02568192 0.5013683
## [10,]  9 0.0545 0.3135 0.005358875 0.02543675 0.5013631
## [11,] 10 0.0550 0.3150 0.005387500 0.02519508 0.5013574
## [12,] 11 0.0555 0.3165 0.005413875 0.02495683 0.5013511
```

```r
#2.
f3 <- function(x){
    sum((-sig[1:2]^2+sqrt(sig[1:2]^4+4*v[1:2]^2*x^2))/(2*v[1:2]^2))-0.5
}
dx <- bisection(f3,vrange, 1e-6)#Apply the bisection method
dx#determine the optimal value of dx
```

```
## [1] 0.1503782
```

```r
#3.
Binomial_tree(isCall=T, isAmerican=F, 10, 0.5, 10, mean(sig), mean(r),
                          N=11)
```

```
## [1] 1.011885
```

```r
Binomial_tree(isCall=F, isAmerican=F, 10, 0.5, 10, mean(sig), mean(r),
                          N=11)
```

```
## [1] 0.7517917
```

```r
#So the call option price is 1.0112 and put option price is 0.7518
```

Problem 3.Dealing with discrete cash dividends For this problem I used the model provided on book "Implementing Derivative Models".

```r
#Price call and put option
#Given the data provided
r <- 0.001
sig <- 0.4
S0 <- 100
K <- 90
Tm <- 4/24
```

```r
div <- 0.6
Tau <- 1/24

#Since the delta_t is constants, and T=4/24, Tau=1/24,
#Obviously we should construct a tree with steps N=4.
#And the dividends is paid at node(1,0);(1,1)
Tau_Step <- 1 #The step that the dividends is paid
Binomial_tree1 <- function(isCall, isAmerican, K, Tm, S0, sig, r,
                           N=4)
  {
  #Precompute constants
  dt <-  Tm/N
  v <-  r-0.5*sig^2
  u <-  sqrt(sig^2*dt+(v*dt)^2) #dxu
  d <- -u
  p <- 0.5+0.5*(v*dt/u) #probability
  disc <- exp(-r*dt) #discount factor
  dpu <- disc*p
  dpd <- disc*(1-p)
  edxud <- exp(2*u)
  edxd <- exp(-1*u)
  M <- N+1
  cp <- ifelse(isCall, 1, -1) #factor cp with 1 or -1

  #Intialize asset prices
  V = S = matrix(0, nrow = M, ncol = M, dimnames = list(
    paste("State", 1:(N+1), sep = ""), paste("T=",0:N, sep = "")))
  S[M,M] <- (S0-div*exp(-r*Tau))*exp(N*d)
  for (i in M:2) {
    S[i-1,M] <- S[i,M]*edxud
  }
  for (j in M:2) {
    S[1,j-1] <- S[1,j]/exp(u)
    for (i in M:2) {
      S[i-1,j-1] <- S[i,j]/exp(d)
    }
  }
  S[Tau_Step,Tau_Step] <- S[Tau_Step,Tau_Step+1]/exp(u)+div*exp(-r*(Tau-Tau)) #Add dividend on stock


  #Intialize option values at maturity
  for (j in 1:M) {
    V[M-j+1,M] <- max(0, cp*(S[M-j+1,M]-K))
  }

  #Step backwards through the tree
  for (j in (M-1):1) {
    for (i in 1:j) {
      V[i,j] <- disc*(p*V[i,j+1]+(1-p)*V[i+1,j+1])
      if(isAmerican){
        V[i,j] <- max(V[i,j],cp*(S[i,j]-K))
      }
    }
```

```
  }
  #Return the price
  return(V[1,1])
}
Call_Option_Price <- Binomial_tree1(isCall=T, isAmerican=T, K, Tm, S0, sig, r,N=4)
Put_Option_Price <- Binomial_tree1(isCall=F, isAmerican=T, K, Tm, S0, sig, r,N=4)
Results <- cbind(Call_Option_Price,Put_Option_Price)
Results #The Price of derivatives
```

```
##      Call_Option_Price Put_Option_Price
## [1,]          12.26564         2.849745
```