Homework1

FE621. Assignment #1.

Problem 1

Question (a) calculate the price of a call and a put

```
# Function to calculate opntion price by BS formular
BS <- function(type, S0, K, tau, r, sigma){

  d1 <- (log(S0/K)+(r+sigma^2/2)*tau) / (sigma*sqrt(tau))
  d2 <- d1 - sigma*sqrt(tau)
  if(type=="C"){
    Price <- S0*pnorm(d1) - K*exp(-r*tau)*pnorm(d2)
  }
  if(type=="P"){
    Price <- K*exp(-r*tau)*pnorm(-d2) - S0*pnorm(-d1)
  }
  return(Price)
}
#input the value that problem give
s0 <- 100
K <- 100
tau <- 30/252
r <- 0.05
sigma <- 0.20

C <- BS("C", s0, K, tau, r, sigma)
P <- BS("P", s0, K, tau, r, sigma)
C  #call option price
```

```
## [1] 3.051184
```

```
P  #put option price
```

```
## [1] 2.457714
```

Question(b) Check put-call parity

```
s0 - K * exp(-r*tau)
```

```
## [1] 0.5934701
```

```
C - P
```

```
## [1] 0.5934701
```

conclusion: we can see that S0 - K * exp(-r*tau)=0.5934701, which equals to call option price - put option price, so we can say that Put-Call parity relation holds.

Question(c) use bisection method to calculate Implied volatility

```
getwd() # get currency work directory
```

```
## [1] "E:/621 computational method/H1"
```

```
#read data that download from Yahoo finance
JPM1 <- read.csv("JPM CallsForFebruary 17, 2017.csv")
JPM2 <- read.csv("JPM CallsForMarch 17, 2017.csv")
JPM6 <- read.csv("JPM CallsForJuly 21, 2017.csv")
#calculate market price
MPrice1 <- matrix((JPM1$Bid + JPM1$Ask)/2) #one month
MPrice2 <- matrix((JPM2$Bid + JPM2$Ask)/2) #two month
MPrice6 <- matrix((JPM6$Bid + JPM6$Ask)/2) #six month

S0 <- 83.55 # Close price of JPM at 2017-01-17
#strike price of JPM option
K1 <- matrix(JPM1$Strike) #one month
K2 <- matrix(JPM2$Strike) #two month
K6 <- matrix(JPM6$Strike) #six month
#calculate time t of each option
tau1 <- as.numeric(difftime("2017-02-17","2017-01-17",units = "days"))/252
tau2 <- as.numeric(difftime("2017-03-17","2017-01-17",units = "days"))/252
tau6 <- as.numeric(difftime("2017-07-21","2017-01-17",units = "days"))/252

err <- function(S0,K,r,tau,sig,MPrice){
   BS("C",S0,K,r,tau,sig) - MPrice
}

#Function to find BS Implied Vol using Bisection Method
Ivol <- function(S0, K, tau, r, MPrice){
  sig <- c()
#loop for every strike price and market price
  for(i in 1:20){
    a <- 0
    b <- 2
    n = 0
    err1 <- err(S0,K[i],r,tau,a,MPrice[i])
    err2 <- err(S0,K[i],r,tau,b,MPrice[i])
#Loop until that the value of function to sigma is less than tolerance level 1e-4
    while(abs(err2-err1) > 1e-4){
        if( err1*err2 < 0){
          b <- (a+b)/2
        }
        else {
          a <- (a+b)/2
        }
    n = n+1
    err1 <- err(S0,K[i],r,tau,a,MPrice[i])
    err2 <- err(S0,K[i],r,tau,b,MPrice[i])
    }
   sig <- c(sig,(a+b)/2)
  }
  print(n) #output the step that need to reach convergence
```

```
  return(sig)
}
Imv1 <- Ivol(S0,K1,0.005,tau1,MPrice1) #one month
```

```
## [1] 18
```

```
Imv2 <- Ivol(S0,K2,0.005,tau2,MPrice2) #two month
```

```
## [1] 19
```

```
Imv3 <- Ivol(S0,K6,0.005,tau6,MPrice6) #six month
```

```
## [1] 14
```

```
#present the result in a table
DF1 <- data.frame(Imv1,Imv2,Imv3)
DF1
```

```
##         Imv1      Imv2      Imv3
## 1  0.9999847 1.9999924 0.4999847
## 2  0.9999924 0.9999962 0.4999981
## 3  0.9999924 0.4999847 0.2499962
## 4  0.4999847 0.9999962 0.2499962
## 5  0.4999924 0.9999962 0.2499962
## 6  0.9999962 0.4999924 0.2499981
## 7  0.4999924 0.4999962 0.2499981
## 8  0.4999962 0.4999962 0.2499981
## 9  0.4999962 0.4999962 0.2499981
## 10 0.4999962 0.4999962 0.2499990
## 11 0.4999962 0.4999962 0.2499990
## 12 0.2499962 0.4999981 0.1249990
## 13 0.4999962 0.2499962 0.1249990
## 14 0.4999962 0.2499981 0.1249981
## 15 0.2499962 0.2499981 0.1249981
## 16 0.2499962 0.2499981 0.1249981
## 17 0.2499962 0.1249962 0.1249962
## 18 0.2499962 0.1249981 0.1249924
## 19 0.2499962 0.2499981 0.1249695
## 20 0.2499962 0.2499981 0.1249390
```

Question(d) use secant method to calculate implied volatility

```
# Function to find BS Implied Vol using Secant Method
secant <- function(S0,K,r,tau,MPrice){
  sig <- c()
#loop for every strike price and market price
  for(i in 1:20){
    n = 0
    x0 <- -5
    x1 <- 0
    err0 <- err(S0,K[i],r,tau,x0,MPrice[i])
    err1 <- err(S0,K[i],r,tau,x1,MPrice[i])
#Loop until that the value of function to sigma is less than tolerance level 1e-4
      while(abs(x1-x0) > 1e-4){
      x <- x1 - err1*(x1 - x0) / (err1 - err0) # Calculate the new x value
```

```
      x0 <- x1
      x1 <- x
      n = n+1
    }
  sig <- c(x,sig)
  }
  print(n) #output the step that need to reach convergence
  return(sig)
}
Iv1 <- secant(S0,K1,0.05,tau1,MPrice1) #one month
```

```
## [1] 6
```

```
Iv2 <- secant(S0,K2,0.05,tau2,MPrice2) #two month
```

```
## [1] 5
```

```
Iv3 <- secant(S0,K6,0.05,tau6,MPrice6) #six month
```

```
## [1] 2
```

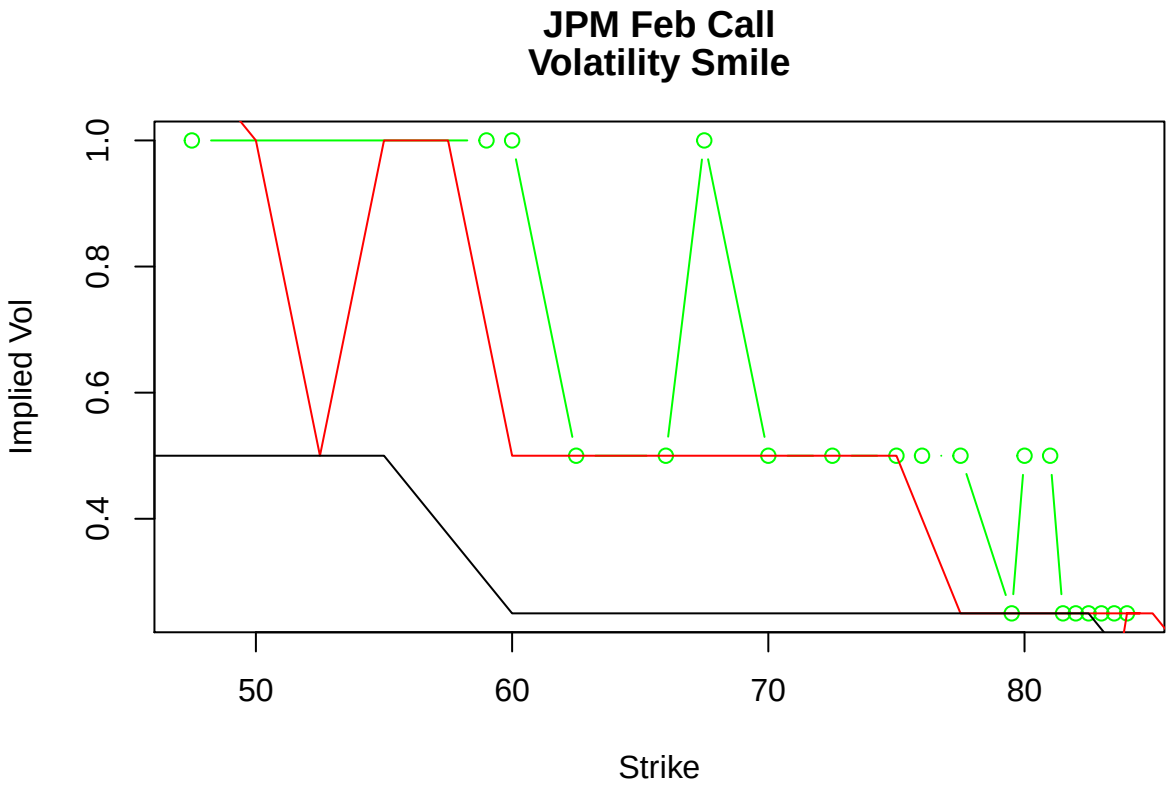
```
#present the result in a table
DF2 <- data.frame(Iv1,Iv2,Iv3)
DF2
```

```
##          Iv1       Iv2         Iv3
## 1  0.6515736 0.5686609 0.004913531
## 2  0.4668465 0.5671312 0.011512920
## 3  0.4569440 0.3892106 0.038433243
## 4  0.6442998 0.3182485 0.093142191
## 5  0.4788075 0.6212237 0.229522865
## 6  0.4730331 0.6321422 0.348258058
## 7  0.6767353 0.5928430 0.519544919
## 8  0.6279260 0.5675823 0.760450915
## 9  0.4262510 0.5511818 0.746669673
## 10 0.6160614 0.5429768 0.600962937
## 11 0.4632383 0.5308885 0.499580663
## 12 0.6042690 0.5189410 0.300032249
## 13 0.4214470 0.5473828 0.204760606
## 14 0.6123561 0.3709873 0.192347992
## 15 0.5961253 0.4942810 0.167447830
## 16 0.3915490 0.4859731 0.184632433
## 17 0.2230100 0.3337954 0.145400318
## 18 0.4044595 0.0692763 0.055764479
## 19 0.3880544 0.2398709 0.152287326
## 20 0.2672030 0.2787131 0.143129888
```

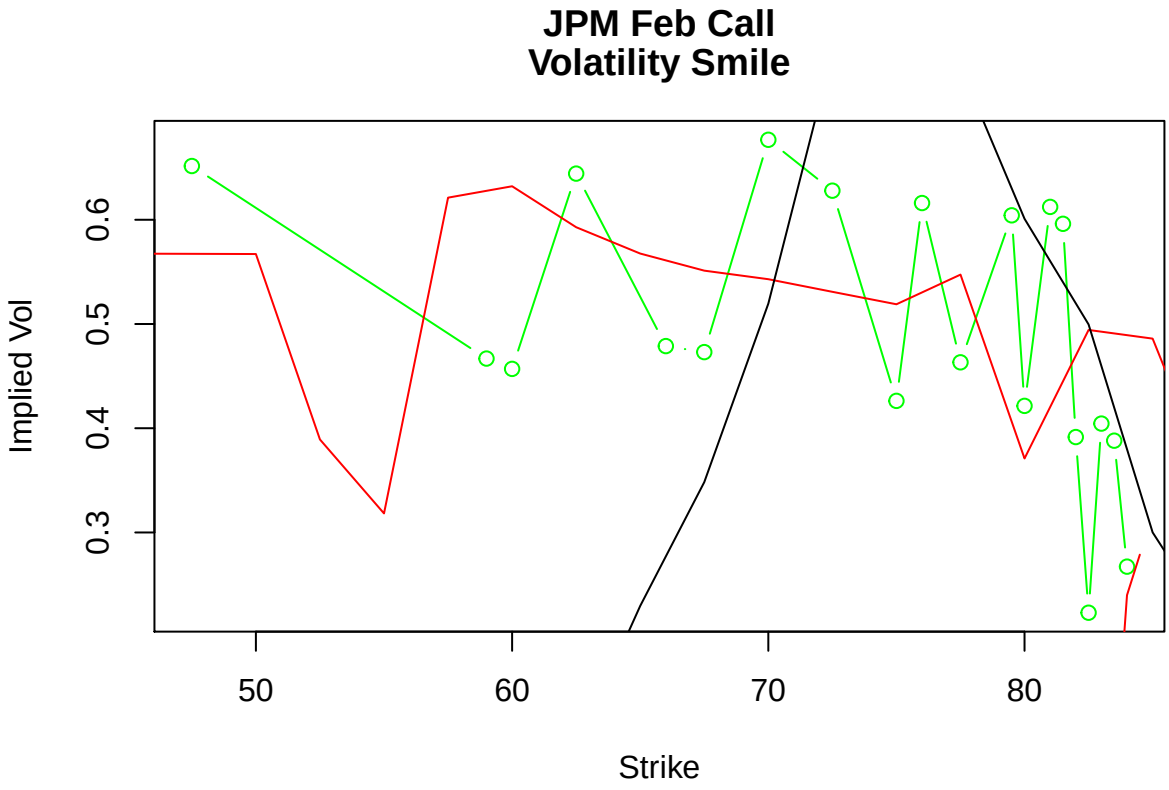
conclusion: I use two different interval for this two method, because I tried to set interval [0,2] in the secant method, but only one month option data can get outcomes. I think the reason of that is secant method has highly requirement of the interval you select. Also, I test the number of iterations neccessary to reach convergence, bisection method use 18,19,14 for JPM option of three different maturity, and secant method use 6,5,2 steps for that. Therefore, my conclusion is that secant method is faster than bisecation, but the selection of secant method has more limitation than the bisection.

Question(e) plot implied volatility

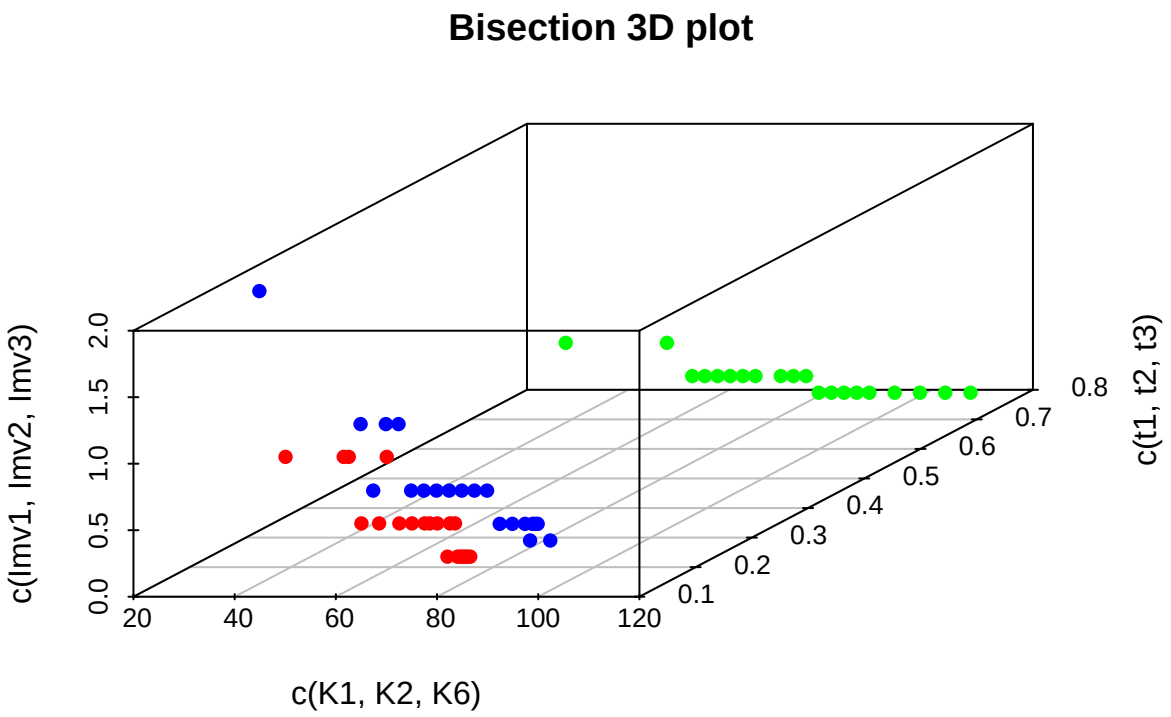
```
# two dimensional plot of implied volatilities versus strike
# 2D plot for implied vol that calculate by bisection method
plot(K1, Imv1, typ="b", col="green", main=c("JPM Feb Call", "Volatility Smile"),
     xlab="Strike", ylab="Implied Vol")
lines(K2, Imv2, col="red")
lines(K6, Imv3)
```



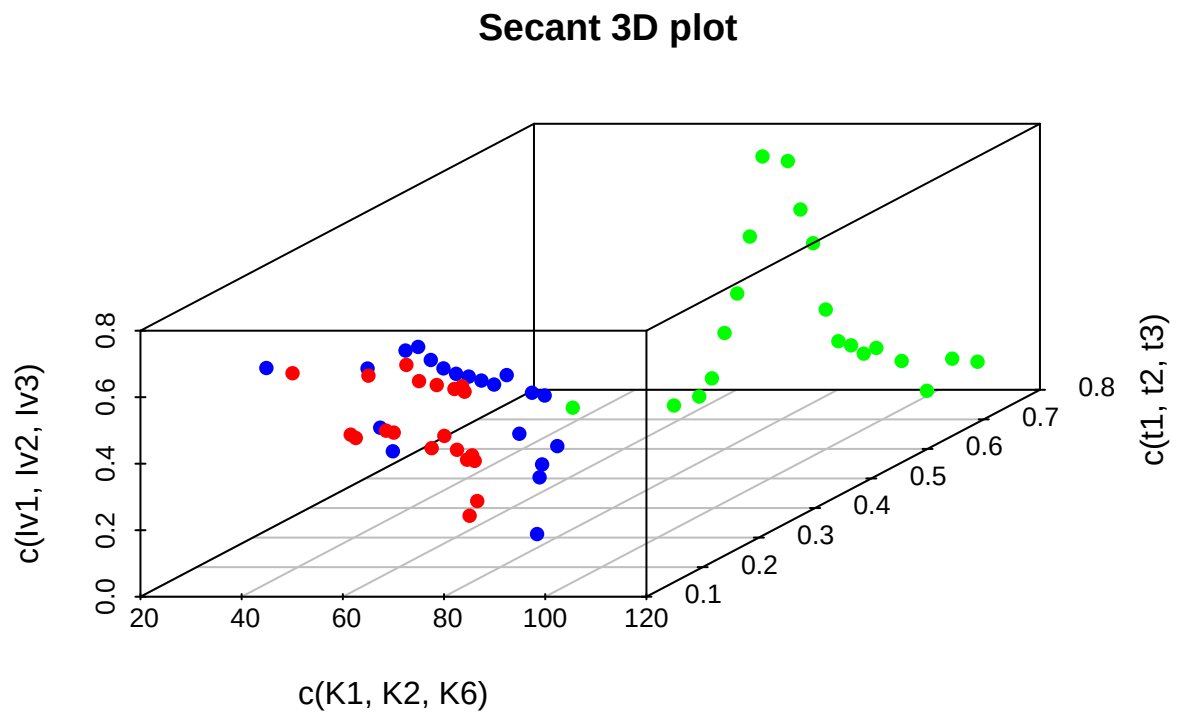

```
## 2D plot for implied vol that calculate by secant method
plot(K1, Iv1, typ="b", col="green", main=c("JPM Feb Call", "Volatility Smile"),
     xlab="Strike", ylab="Implied Vol")
lines(K2, Iv2, col="red")
lines(K6, Iv3)
```




```
# Creat 3D plot of volatilities versus K & T
COLOR=c(rep("red",20),rep("blue",20),rep("green",20))
library(scatterplot3d)
t1 <- rep(tau1,20)
t2 <- rep(tau2,20)
t3 <- rep(tau6,20)
scatterplot3d(c(K1,K2,K6),c(t1,t2,t3),c(Imv1, Imv2, Imv3),main="Bisection 3D plot",
              color = COLOR,pch=16,type = "p")
```




```
scatterplot3d(c(K1,K2,K6),c(t1,t2,t3),c(Iv1, Iv2, Iv3),main="Secant 3D plot",
              color = COLOR,pch=16,type = "p")
```



Question(f) Greeks

```
# method1 use formula to calculate
d1 <- (log(s0/K)+(r+sigma^2/2)*tau) / (sigma*sqrt(tau))
d2 <- d1 - sigma*sqrt(tau)
norm_deriv <- 1.0/sqrt(2*pi) * exp(-d1^2/2)

Delta <- pnorm(d1)
Delta
```

```
## [1] 0.54806
```

```
Vega <- norm_deriv * s0 * sqrt(tau)
Vega
```

```
## [1] 13.66481
```

```
Gamma <- norm_deriv * (1.0/(s0 * sqrt(tau)*sigma))
Gamma
```

```
## [1] 0.05739221
```

```
# method2 use approximation
#delta
delta <- function(s0,K,r,T,sigma){
  ybi <- 0.0001
  y <- (BS("C", s0+ybi, K, tau, r, sigma)-BS("C", s0, K, tau, r, sigma))/ybi
```

```
  return(y)
}
delta(s0,K,r,tau,sigma)
```

```
## [1] 0.5480629
```

```
#vega
vega <- function(s0,K,r,T,sigma){
  ybi <- 0.0001
  y <- (BS("C", s0, K, tau, r, sigma+ybi)-BS("C", s0, K, tau, r, sigma))/ybi
  return(y)
}
vega(s0,K,r,tau,sigma)
```

```
## [1] 13.66483
```

```
#gamma
gamma <- function(s0,K,r,T,sigma){
  ybi <- 0.00001
  y <- (BS("C", s0+2*ybi, K, tau, r, sigma)-2*BS("C", s0+ybi, K, tau, r, sigma)
        + BS("C", s0, K, tau, r, sigma))/ybi^2
  return(y)
}
gamma(s0,K,r,tau,sigma)
```

```
## [1] 0.05726974
```

conclusion: By using formula directly, we can get Delta=0.54806, Vega=13.66481,Gamma=0.05739221. By using approximation to partial derivatives, we can get Delta=0.5480629,Vega=13.66483,Gamma=0.05726974. I think approximation method can get more precision outcomes than using formula directly.

Question(g) calulate greeks for option that I download

```
#delta
delta <- function(s0,K,r,T,sigma){
  ybi <- 0.0001
  y <- (BS("C", s0+ybi, K, tau, r, sigma)-BS("C", s0, K, tau, r, sigma))/ybi
  return(y)
}
delta(S0,K1,0.0248,tau1,Imv1)
```

```
##             [,1]
##  [1,] 0.9654525
##  [2,] 0.8828616
##  [3,] 0.8730030
##  [4,] 0.9629536
##  [5,] 0.9292417
##  [6,] 0.7879504
##  [7,] 0.8705700
##  [8,] 0.8226918
##  [9,] 0.7670477
## [10,] 0.7429224
## [11,] 0.7050874
## [12,] 0.7432549
## [13,] 0.6387258
```

```
## [14,] 0.6114294
## [15,] 0.6425809
## [16,] 0.6157964
## [17,] 0.5886147
## [18,] 0.5611686
## [19,] 0.5335923
## [20,] 0.5060196
```

```
#vega
vega <- function(s0,K,r,T,sigma){
  ybi <- 0.0001
  y <- (BS("C", s0, K, tau, r, sigma+ybi)-BS("C", s0, K, tau, r, sigma))/ybi
  return(y)
}
vega(S0,K2,0.0248,tau2,Imv2)
```

```
##             [,1]
##  [1,]  2.1413018
##  [2,]  2.8562943
##  [3,]  0.2304587
##  [4,]  4.3597074
##  [5,]  5.1734580
##  [6,]  1.4876902
##  [7,]  2.3344637
##  [8,]  3.4140788
##  [9,]  4.6874945
## [10,]  6.0805147
## [11,]  7.4934502
## [12,]  8.8162747
## [13,]  7.3334897
## [14,]  9.7165673
## [15,] 11.2157008
## [16,] 11.4151312
## [17,]  7.1103274
## [18,] 11.4386253
## [19,] 11.4991935
## [20,] 11.4839393
```

```
#gamma
gamma <- function(s0,K,r,T,sigma){
  ybi <- 0.00001
  y <- (BS("C", s0+2*ybi, K, tau, r, sigma)-2*BS("C", s0+ybi, K, tau, r, sigma) +
        BS("C", s0, K, tau, r, sigma))/ybi^2
  return(y)
}
gamma(S0,K6,0.0248,tau6,Imv3)
```

```
##               [,1]
##  [1,]  1.421085e-04
##  [2,]  1.278977e-03
##  [3,] -1.421085e-04
##  [4,]  3.552714e-04
##  [5,]  7.105427e-04
##  [6,]  2.273737e-03
##  [7,]  5.542233e-03
```

```
##  [8,]  1.278977e-02
##  [9,]  3.510081e-02
## [10,]  4.682477e-02
## [11,]  5.400125e-02
## [12,]  1.054090e-01
## [13,]  6.842527e-02
## [14,]  2.915890e-02
## [15,]  8.434364e-03
## [16,]  1.711686e-03
## [17,]  2.719830e-05
## [18,]  1.422338e-07
## [19,]  2.864030e-10
## [20,]  2.541480e-13
```

Problem 2

Question(a) compute integral by trapzoidal and the Simpson rules

```
#input the real-value function that problem give
f <- function(x){
  if (x == 0){
    y <- 1
  }
  else{
    y <- sin(x)/x
  }
  return(y)
}

# Trapezoidal rule
trap1 <- function(a,N){
  In <- 0
  h <- (2*a)/N

  for(i in 1:N){
    x <- (-a)+i*h
    x1 <- (-a)+(i-1)*h
    area <- h*(f(x)+f(x1))/2
    In <- area + In
  }
  return(In)
}
trap1(10^4,10^4)
```

```
## [1] 3.141715
```

```
# Simpson rule
simp1 <-  function(a,N){
  In <- 0
  h <- (2*a)/N
  for(i in 1:N){
    x <- (-a)+i*h
```

```
    x1 <- (-a)+(i-1)*h
    area <- (x - x1)/6 *(f(x1)+4*f((x + x1)/2)+f(x))
    In <- area + In
  }
  return(In)
}
simp1(10^4,10^4)
```

```
## [1] 3.141784
```

Question(b) compute the truncation error

```
trunerr <- function(type,a,N){
  if(type=="trap")
    trap1(a,N)-pi
  else if(type=="simp")
    simp1(a,N)-pi
}
trunerr("trap",1e+4,1e+4) #truncation error in trapezoidal rule
```

```
## [1] 0.000122283
```

```
trunerr("simp",1e+4,1e+4) #truncation error in Simpson rule
```

```
## [1] 0.0001916357
```

conclusion: truncation error that calculate by Simpson rule is kind of larger than that calculate by trapezoidal rule.

Question(c) use tolerance value to check every iteration and evaluate the number of step until the algorithms reach convergence

```
# check iteration in trapezoidal rule
trap_check <- function(a){
    n <- 4
    h <- (2*a)/n
  tol <- 1e-4  #set tolerance value
    s <- trap1(a, n)
    s.diff <- tol + 1 # ensures to loop at once.
    while (s.diff > tol ) {
        s.old <- s #
        n <- n+1 #calculate the number of step
        h <- (2*a)/n
        s <- trap1(a,n)
        s.diff <- abs(s-s.old)
    }
    return(n)
}
trap_check(10^4)
```

```
## [1] 3262
```

```
# check iteration in Simpson rule
simp_check <- function(a){
    n <- 4
    h <- (2*a)/n
  tol <- 1e-4 #set tolerance value
    s <- simp1(a, n)
    s.diff <- tol + 1 # ensures to loop at once.
    while (s.diff > tol ) {
        s.old <- s
        n <- n+1 #calculate the number of step
        h <- h/2
        s <- simp1(a,n)
        s.diff <- abs(s-s.old)
    }
    return(n)
}
simp_check(10^4)
```

```
## [1] 3229
```

conclusion: the number of step until trapezoidal rule get convergence is 3262, and the steps of Simpson rule get convergence is 3229, so that Simpson rule convergence faster than trapezoidal rule.

Question(d) use trapezoidal and simpson to approximation function that problem give

```
#input the function
g <- function(x){
  y <- 1 + exp(-x) * sin(8*x^(2/3))
  return(y)
}
#quadrature method to approximate
Quadrature <- function(type,N){
  In <- 0
  a <- 0
  b <- 2
  h <- (b-a)/N

  for(i in 1:N){
    x <- (a)+i*h
    x1 <- (a)+(i-1)*h
    if(type == "trap")
      area <- h*(g(x)+g(x1))/2
    else if(type == "simp")
      area <- (x - x1)/6 *(g(x1)+4*g((x + x1)/2)+g(x))
    In <- area + In
  }
  return(In)
}
Quadrature("trap",10^4)
```

```
## [1] 2.016279
```

```r
Quadrature("simp",10^4)
```

```
## [1] 2.01628
```

```r
# Integral
Integral <- function(type){
  n <- 1
  a <- 0
  b <- 2
  tol <- 1e-4

  s <- Quadrature(type, n)
  s.diff <- tol + 1 # ensures to loop at once.
  while (s.diff > tol ) {
    s.old <- s
    n <- n+1
    h <- (b-a)/n
    s <- Quadrature(type,n)
    s.diff <- abs(s-s.old)
  }
  print(n)
  return(s)
}
Integral("trap")
```

```
## [1] 65
```

```
## [1] 2.012584
```

```r
Integral("simp")
```

```
## [1] 27
```

```
## [1] 2.014849
```

conclusion: convergence steps number in trapezoidal rule is 65, convergence steps number in simpson rule is 27, the out come of integral is around 2.01. Obviously, Simpson rule is more faster than trapezoidal rule.

Problem 3

```r
#input all parameters
r <- 0
q <- 0
rho <- -0.3
theta <- 0.1
sigma <- 0.2
lambda <- 0

i <- sqrt(-1+0i)
kappa <- c(4,2,1)
u1 <- 0.5
u2 <- -0.5
a <- kappa*theta
b1 <- kappa+lambda-rho*sigma
b2 <- kappa+lambda
```

```
S <- 1
V0 <- 0.1
T <- 5
k <- c(0.5,0.75,1.0,1.25,1.5)

psi <- function(Phi,b,u){
  d <- sqrt((rho*sigma*Phi*i-b)^2 - sigma^2*(2*u*Phi*i-Phi^2))
  g <- (b-rho*sigma*Phi*i+d)/(b-rho*sigma*Phi*i-d)
  C <- (r-q)*Phi*i*T+kappa*theta*((b-rho*sigma*Phi*i+d)*T - 2*log((1-g*exp(d*T))/(1-g)))/sigma^2
  D <- (b-rho*sigma*Phi*i+d)/sigma^2 * ((1-exp(d*T))/(1-g*exp(d*T)))
  y <- exp( C+D*V0+i*Phi*log(S))
  return(y)
}

expre <- function(Phi,b,u,k){
  y <- Re(exp(-i*Phi*log(k))*psi(Phi,b,u)/(i*Phi))
  return(y)
}

#use simpon method to calculate integral
Simpon <- function(up,N,b,u,k){
 In <- 0
 h <- up/N
 for(j in 1:N+1){
   Phi <- j*h
   Phi1 <- (j-1)*h
   area <- (Phi - Phi1)/6 * (expre(Phi1,b,u,k) + 4*expre((Phi + Phi1)/2,b,u,k)+expre(Phi,b,u,k))
   In <- area + In
 }
 return(In)
}

In1k1 <- mean(Simpon(700,1e+4,b1,u1,k[1]))
In1k2 <- mean(Simpon(700,1e+4,b1,u1,k[2]))
In1k3 <- mean(Simpon(700,1e+4,b1,u1,k[3]))
In1k4 <- mean(Simpon(700,1e+4,b1,u1,k[4]))
In1k5 <- mean(Simpon(700,1e+4,b1,u1,k[5]))
simpon1 <- c(In1k1,In1k2,In1k3,In1k4,In1k5)
In2k1 <- mean(Simpon(700,1e+4,b2,u2,k[1]))
In2k2 <- mean(Simpon(700,1e+4,b2,u2,k[2]))
In2k3 <- mean(Simpon(700,1e+4,b2,u2,k[3]))
In2k4 <- mean(Simpon(700,1e+4,b2,u2,k[4]))
In2k5 <- mean(Simpon(700,1e+4,b2,u2,k[5]))
simpon2 <- c(In2k1,In2k2,In2k3,In2k4,In2k5)

P1 <- 1/2 + 1/pi * simpon1
P2 <- 1/2 + 1/pi * simpon2

#calculate option price by Hedson model
Heston_Price <- S*P1 - k*exp(-(r-q)*T)*P2

DF <- data.frame(k,Heston_Price)
DF
```

```
##      k Heston_Price
## 1 0.50    0.5270180
## 2 0.75    0.3739106
## 3 1.00    0.2624349
## 4 1.25    0.1820426
## 5 1.50    0.1232388
```

conclusion: the outcomes I get is close to, but a little lower than the value of Table 1 in paper.