Question 1

Black–Scholes–Merton Pricing Formula. Implied Volatility

European Call:

```
CallOption(100,30/252,100,5/100,0.2)
```

- 3.051184

European Put

```
PutOption(100,30/252,100,5/100,0.2)
```

- 2.457714

Put-Call Parity

```
PutCallParity(100,30/252,100,0.05,0.2)
```

- RHS = 0.5934701
- LHS = 0.5934701
- LHS – RHS = 0

Implied Volatility

Bisection Method

Table 1 Implied volatility using bisection method

Implied Volatility	Strike Prices
1.4713135	60
0.0000000	65
0.0000000	70
1.0897217	75
0.0000000	80
0.0000000	85
0.0000000	90
0.0000000	95
0.0000000	100
0.0000000	105
0.0000000	110
0.0000000	115
0.0000000	120
0.1370239	125
0.1508789	130

0.1530304	135
0.1561890	140
0.1691895	145
0.1936035	150
0.2202148	155
0.0000000	65
0.0000000	80
0.0000000	90
0.0000000	95
0.0000000	100
0.3148193	105
0.2999268	110
0.2909241	115
0.2773285	120
0.2724609	125
0.2660675	130
0.2674561	135
0.2730103	140
0.2866821	145
0.3021240	150
0.3212891	155
0.3164062	160
0.3659668	165
0.3715820	170
0.3642578	175
0.0000000	65
1.3483887	75
0.0000000	80
0.9434204	85
0.8943481	90
0.8481750	100
0.8087769	105
0.7549438	110
0.7453156	115
0.7178650	120
0.6936798	125
0.6744843	130
0.6561127	135
0.6424561	140
0.6271973	145
0.6169128	150
0.6095886	155
0.6038971	160
0.5983276	165
0.5971375	170

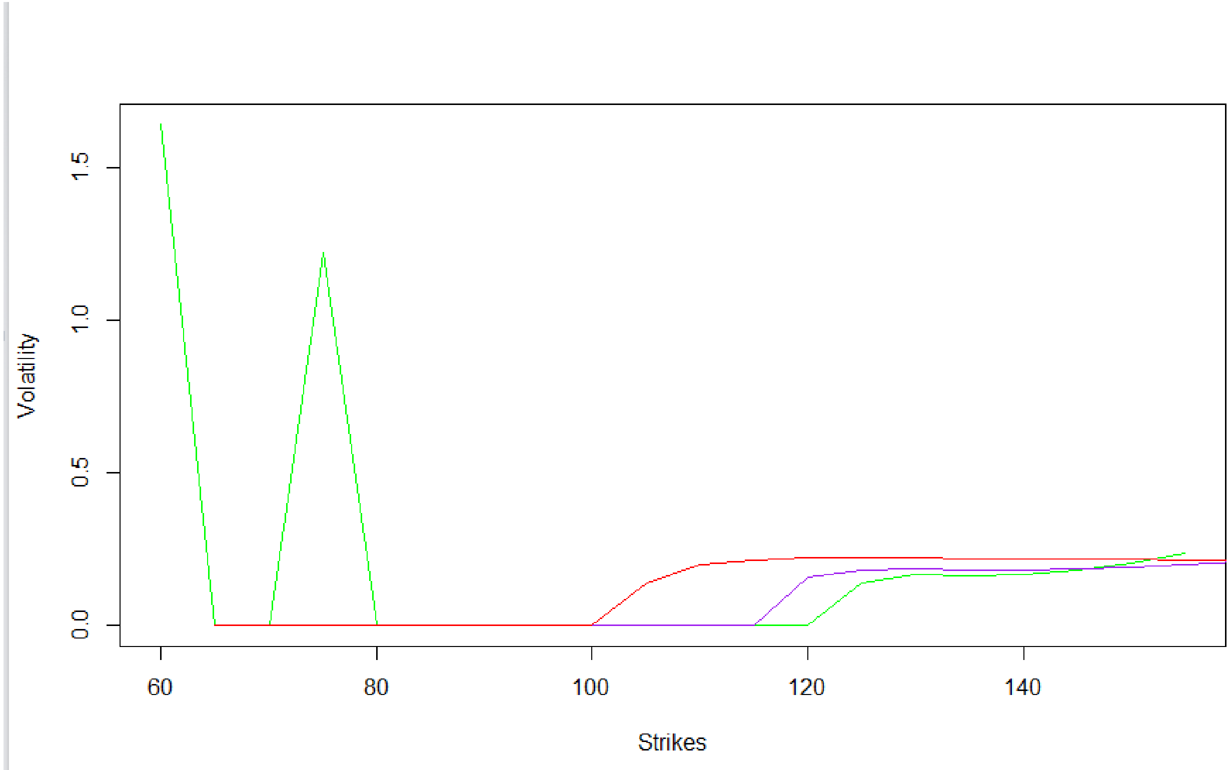

Figure 1 Implied Volatility V Strike Price

Green – 1month maturity

Purple – 2month maturity

Red – 6month maturity

The bisection method had to stopped using a counter, as the data had multiple roots. Not having a unique root resulted in the bisection method failing and going in an infinite loop. The data shown above is after the method was stopped after 10000 iterations

Greeks

```
tau <- 30/252
S <- 100
k <- 100
r <- 0.05
sigma <- 0.2
h <- 0.0001
q <- 0
```

```
Delta(S,tau,k,r,sigma) = 0.54806
Gamma(S,tau,k,r,sigma) = 0.05739221
Vega(S,tau,k,r,sigma) = 13.66481

DeltaApprox(S,tau,k,r,sigma) = 0.5480629
GammaApprox(S,tau,k,r,sigma) = 0.05739125
VegaApprox(S,tau,k,r,sigma) = 13.66483
```

Table 2: Greeks for options with different maturities

Gamma	Delta	Vega
0.0012349233	0.9559351935	4.294035e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0019994673	0.9442081424	5.183736e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0200913064	0.9342578259	5.898126e+00
0.0439939640	0.7183109066	1.555561e+01
0.0525396615	0.4677992178	1.832031e+01
0.0403687750	0.2381021262	1.426288e+01
0.0228663311	0.1111354368	8.730956e+00
0.0127982958	0.0608346766	5.551272e+00
0.0083399065	0.0430500221	4.216488e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0065611516	0.9801230263	2.221979e+00
0.0243105092	0.8770275693	9.379559e+00
0.0408007850	0.7009108202	1.599802e+01
0.0469370320	0.4738308974	1.834064e+01
0.0384716259	0.2605973744	1.496370e+01
0.0232271091	0.1185312691	9.140121e+00
0.0114581233	0.0486641073	4.653482e+00
0.0051437077	0.0194083361	2.176964e+00
0.0024201308	0.0086854545	1.086961e+00
0.0012609469	0.0044887593	6.060257e-01
0.0004197254	0.0013252249	2.011820e-01
0.0004870826	0.0018329292	2.702956e-01
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00

0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	1.0000000000	0.000000e+00
0.0000000000	0.9999997418	6.243980e-05
0.0006068035	0.9982708423	2.564547e-01
0.0043954174	0.9822188417	2.020372e+00
0.0127130306	0.9330682428	5.978691e+00
0.0245336196	0.8311694580	1.160946e+01
0.0351462859	0.6732764851	1.661857e+01
0.0391302990	0.4828691401	1.836323e+01
0.0343369777	0.3005734234	1.603390e+01
0.0242213360	0.1598410567	1.120531e+01
0.0139070977	0.0733142806	6.412731e+00
0.0065702999	0.0285360435	3.009848e+00
0.0026302738	0.0097467089	1.202278e+00
0.0009051149	0.0029285146	4.132048e-01
0.0002761014	0.0007967112	1.261813e-01

Question 2

a)

```
SimpsonRule(-1000000, 1000000, 1000000, func) = 3.141591
TrapezoidalRule(-1000000, 1000000, 1000000, func) = 3.141591
```

b)

```
SimpsonError() = 1.202971e-06
TrapError() = 1.552964e-06
```

c) 3.141591

d)

```
newSimpsonRule(0,2,1e-4,func2) = 2.01628
newTrapRule(0,2,1e-4,func2) = 2.016281
```

Question 3

```
u <- data.frame(2)
u[1]  = 0.5
u[2] = -0.5
kap <- 2
lam <- 0
phi <- 2
p <- -0.3
V <- 0.1
sig <- 0.2
the <- 0.1
a <-  kap*the
b <- data.frame(2)
b[1] <- kap + lam - p*sig
b[2]<- kap + lam
q <- 1
r <- 0.04
tau <- 5
d <- data.frame(2)
for(i in 1:2){
d[i] <- sqrt((p*sig*as.complex(phi) - b[i])^2 - sig^2 *
                 (2*u[i]*as.complex(phi) - sig^2))
}

g <- data.frame(2)
for(i in 1:2){
    g[i] = (b[i] - p*sig*as.complex(phi) + d[i]) /
        (b[i] - p*sig*as.complex(phi) - d[i])
}

C <- function(tau,phi) {
    ans1 <- data.frame(2)
    for (i in 1:2){
    ans1[i] <- (r-q)*as.complex(phi)*tau +
        (kap*the / sig^2)*(b[i] - p*sig*as.complex(phi) + d[i]
         - 2*log((1 - g[i] * exp(d[i]*tau)/(1-g[i]) )))
    }
    return(ans1)
}

D <- function(tau,phi) {
    coun <- data.frame(2)
    for(i in 1:2){
        coun[i] = ((b[i] - p*sig*as.complex(phi) + d[i])/sig^2) *
        ((1-exp(d[i] * tau))/ (1-g[i]*exp(d[i] * tau)))

    }
    return(coun)
}


sphi <- function(S,V,tau,phi) {
    ans2 <- data.frame(2)
    for(i in 1:2) {
        ans2[i] = exp(C(tau,phi)[i] + D(tau,phi)[i]*V + as.complex(phi)*S)
    }
```

```
    return(ans2)
}
sphi(1,0.1,5,1)

Real <- function(S=1,V=0.1,tau = 5,U) {
    ans <- data.frame(2)
    for(i in 1:2){
    ans[i] <- Re(exp(as.complex(-u[i])*log(k)) * sphi(S,V,tau,u[i]) /
                  (as.complex(u[i])))
    }
    return(ans)
}

P <- data.frame(2)

for (i in 1:2){
    P[i] <- 0.5 * (1/pi) * SimpsonRule(0,100000,1000,Real)
}

HestonCall <- function(S, V, k, tau) {
    S*P[1] * - k*exp-((r-q)*(tau))*P[2]
}
```

Appendix

Question 1

```
CallOption <- function(Stock,tau, Strike, rate, sigma) {
    d1 <- (log(Stock/Strike) + (rate + sigma^2/2 ) * tau) / (sigma * sqrt(tau))
    d2 <- d1 - sigma*sqrt(tau)
    price <- Stock*pnorm(d1) - Strike * exp(-rate*tau)*pnorm(d2)
    return(price)

}

PutOption <- function(Stock, tau, Strike, rate, sigma){

    d1 <- (log(Stock/Strike) + (rate + sigma^2/2 ) * tau) / (sigma * sqrt(tau))
    d2 <- d1 - sigma*sqrt(tau)
    price <- Strike * exp(-rate*tau) * pnorm(-d2) - Stock*pnorm(-d1)
    return(price)
}


PutCallParity <- function(Stock,tau, Strike, rate, sigma) {
    LHS <- CallOption(Stock,tau, Strike, rate, sigma ) - PutOption(Stock,tau, Strike
, rate, sigma )
    RHS <- Stock - Strike *exp(-rate*tau)
    print(RHS)
    print(LHS)
    return(LHS-RHS)

}

# Option data ----
maturity1 <- getOptionChain("FB","2017-03-17")
maturity2 <- getOptionChain("FB","2017-04-21")
maturity3 <- getOptionChain("FB","2017-09-15")
maturity1 <- maturity1["calls"]
maturity2 <- maturity2["calls"]
maturity3 <- maturity3["calls"]
month1 <- data.frame(maturity1)
month2 <- data.frame(maturity2)
month3 <- data.frame(maturity3)
month1 <- month1[1:20,]
month2 <- month2[1:20,]
month3 <- month3 [1:20,]
avg1 <- (month1$calls.Bid + month1$calls.Ask )/2
avg2 <- (month2$calls.Bid + month2$calls.Ask) / 2
avg3 <- (month3$calls.Bid + month3$calls.Ask) / 2
Stock1 <- getQuote("FB")


# Bisection Method ----
BisectionMethod <-  function(S, tau, Strike, r, market){
    up <- 2
    down <- 0
    mid <- (up + down) / 2
    i<- 0
    tol <- CallOption(S, tau, Strike, r, mid) - market
     while(abs(tol) > 1e-04 && i<10000){
```

```
        if(tol < 0){
            down <- mid
        }else{
            up <- mid
            }
        mid<- (up + down)/2
        tol <- CallOption(S, tau,Strike, r, mid) - market
        i <- i + 1
    }
    return(mid)
}

vol1 <- matrix(nrow = 1, ncol = 20)
vol2 <- matrix(nrow = 1, ncol = 20)
vol3 <- matrix(nrow = 1, ncol = 20)

for(i in 1:20) {
    vol1[i] = BisectionMethod(Stock1$Last,26/360,month1$calls.Strike[i],0.04,
t(avg1[i]))
   vol2[i] = BisectionMethod(Stock1$Last,58/360,month2$calls.Strike[i],0.04,t
(avg2[i]))
    vol3[i] = BisectionMethod(Stock1$Last,203/360,month3$calls.Strike[i],0.04
,t(avg3[i]))
}

# Plotting ----
plot( month1$calls.Strike, t(vol1) , type = 'l', col = 'green',xlab = 'Strike
s' , ylab = 'Volatility')
lines(month2$calls.Strike , t(vol2),type = 'l' , col = 'purple')
lines(month3$calls.Strike, t(vol3) , type = 'l' , col = 'red')



# Secant Method ----
SecantMethod <- function(S,tau,K,r,market){
 x1 <- 0
 x2 <- CallOption(S,tau,k,0.04,1)
 i <- 0
 while( i < 100) {
    ans[i] = market - (CallOption(S,tau,K,r,x1) - Stock1$Last)*
        (x2-x1)/(CallOption(S,tau,K,r,x2) - CallOption(S,tau,K,r,x1))
    x1 = x2
    x2 = ans[i]
    i = i +1


 }
 return(x2)

   }
impvol1 <- matrix(nrow = 1, ncol = 20)
impvol2 <- matrix(nrow = 1, ncol = 20)
impvol3 <- matrix(nrow = 1, ncol = 20)
impvol1[1] = SecantMethod(Stock1$Last,30/360,month1$calls.Strike[1],0.04,avg1[1])
impvol1
for (c in 1:20) {
```

```
impvol1[c] = SecantMethod(Stock1$Last,26/360,month1$calls.Strike[c],0.04,avg1[c])
impvol2[c] = SecantMethod(Stock1$Last,58/360,month2$calls.Strike[c],0.04,avg2[c])
impvol1[c] = SecantMethod(Stock1$Last,203/360,month3$calls.Strike[c],0.04,avg3[c])
}


#Greeks
tau <- 30/252
r <- 0.05
sigma <- 0.2
h <- 0.0001
Delta <- function(S,tau,k,r,sigma){
    d1 <- (log(S/k) + (r + sigma^2/2 ) * tau) / (sigma * sqrt(tau))
    return(pnorm(d1))
}
Vega <- function(S,tau, k,r,sigma){
    d1 <- (log(S/k) + (r + sigma^2/2 ) * tau) / (sigma * sqrt(tau))
    vega <-  S * sqrt(tau) * (1/sqrt(2*pi)) * exp(-d1^2/2)
    return(vega)
}
Gamma <- function(S,tau,k,r,sigma) {
    d1 <- (log(S/k) + (r + sigma^2/2 ) * tau) / (sigma * sqrt(tau))
    gamma <- exp(-d1^2/2) / (S * sigma * sqrt(2*pi*tau))
   return (gamma)
}


Delta(S,tau,k,r,sigma)

Vega(S,tau,k,r,sigma)

Gamma(S,tau,k,r,sigma)

# Greeks Approximation

DeltaApprox <- function(S,tau,k,r,sigma) {

   Delta_approx <- (CallOption(S+h,tau,k,r,sigma) -
CallOption(S,tau,k,r,sigma)) / h

    return(Delta_approx)

   }

VegaApprox <- function(S,tau,k,r,sigma){

   Vega_approx<- (CallOption(S,tau,k,r,sigma+h) -
CallOption(S,tau,k,r,sigma))/h

   return(Vega_approx)

   }

GammaApprox <- function(S,tau,k,r,sigma){

    Gamma_approx <-(CallOption(S+2*h,tau,k,r,sigma)-
2*CallOption(S+h,tau,k,r,sigma)

     + CallOption(S,tau,k,r,sigma) )/h^2

    return(Gamma_approx) }
```

```
# Implied volatility Greeks approximation
delta1 <- matrix(nrow = 1, ncol = 20)
delta2 <- matrix(nrow = 1, ncol = 20)
delta3<- matrix(nrow = 1, ncol = 20)
vega1 <- matrix(nrow = 1, ncol = 20)
vega2 <- matrix(nrow = 1, ncol = 20)
vega3 <- matrix(nrow = 1, ncol = 20)
gamma1 <-  matrix(nrow = 1, ncol = 20)
gamma2 <-  matrix(nrow = 1, ncol = 20)
gamma3 <-  matrix(nrow = 1, ncol = 20)
for(i in 1:20){
   delta1[i] <-
DeltaApprox(Stock1$Last,tau,month1$calls.Strike[i],0.04,vol1[i])
   delta2[i] <-
DeltaApprox(Stock1$Last,tau,month2$calls.Strike[i],0.04,vol2[i])
   delta3[i] <-
DeltaApprox(Stock1$Last,tau,month3$calls.Strike[i],0.04,vol3[i])
   }
for(i in 1:20){
    vega1[i] <-
VegaApprox(Stock1$Last,tau,month1$calls.Strike[i],0.04,vol1[i])
    vega2[i] <-
VegaApprox(Stock1$Last,tau,month2$calls.Strike[i],0.04,vol2[i])
    vega3[i] <-
VegaApprox(Stock1$Last,tau,month3$calls.Strike[i],0.04,vol3[i])
}
for (i in 1:20) {
    gamma1[i] <-
GammaApprox(Stock1$Last,tau,month1$calls.Strike[i],0.04,vol1[i])
    gamma2[i] <-
GammaApprox(Stock1$Last,tau,month2$calls.Strike[i],0.04,vol2[i])
    gamma3[i] <-
GammaApprox(Stock1$Last,tau,month3$calls.Strike[i],0.04,vol3[i])
}
Vega
t(gamma)
t(delta)
t(vega)
```

```r
# Question 2

SimpsonRule <- function(a,b,m, f){
    m <- m-1
    h <- (b-a)/m
    x <- seq(from = a, to = b, by = h/2)
    y <- f(x)
    ix1 <- seq(from =3, by =2, to = 2*m-1)
    ix2 <- seq(from =2, by =2, to= 2*m) - 1
    return(h/6 * (y[1] + 2*sum(y[ix1]) + 4*sum(y[(ix2)])  + y[2*m+1]))

}

TrapezoidalRule <- function(a, b, m, f){
    h <-(b-a)/(m-1)
    x <- seq(from = a, to = b, length = m)
    y <- f(x)
    h * (0.5 * y[1] + sum(y[2:(m-1)]) +y[m])
}

func <- function(x){
    if (x == 0) {
        y <- 1

    } else {
        y <- sin(x) / x
    }
 return(y)
}

SimpsonError <- function (){
    return(abs(pi – SimpsonRule(-1000000, 1000000, 1000000, func)))
}



TrapError <- function() {
    return(abs(pi - TrapezoidalRule(-1000000, 1000000, 1000000, func)))

}

#tolerance ----
newSimpsonRule <- function(a,b,tol,f){
    m =1000000
    for(i in 1:m) {
        temp <-  SimpsonRule(a,b,m,f)
```

```
        temp2 <- SimpsonRule(a,b,m+1,f)
        if (abs(temp-temp2) < tol){
            return(SimpsonRule(a,b,m,f))
        }
    }

}
newTrapRule <- function(a,b,tol,f){
    m =1000000
    for(i in 1:m) {
        temp <-  TrapezoidalRule(a,b,m,f)
        temp2 <- TrapezoidalRule(a,b,m+1,f)
        if (abs(temp-temp2) < tol){
            return(TrapezoidalRule(a,b,m,f))
        }
    }
}

func2 <- function(x) {
    return(1 + exp(-x) * sin(8 * x^(2/3)))
}
```