621_Homework 1

Liting Hu

Contents

Question 1

a)

To calculate the European Call and Put Option, define functions "Callprice" and "Putprice" based on the solution of Black-Scholes formula.

```
Callprice <- function(S0, tau, K, r, sigma) {
    d1 <- (log(S0/K)+(r-sigma^2/2)*tau)/tau/sqrt(tau)
    d2 <- d1 - sigma*sqrt(tau)
    c <- S0*pnorm(d1) - K*exp(-r*tau)*pnorm(d2)
    return(c)
}

Putprice <- function(S0, tau, K, r, sigma) {
    d1 <- (log(S0/K)+(r-sigma^2/2)*tau)/tau/sqrt(tau)
    d2 <- d1 - sigma*sqrt(tau)
    p <- K*exp(-r*tau)*pnorm(-d2) - S0*pnorm(-d1)
    return(p)
}

S0 <- 100       # Stock price
tau <- 30/252   # Time to maturity
K <- 100        # Strike price
r <- 0.05       # Interest rate
sigma <- 0.2    # Volatility
Callprice(S0, tau, K, r, sigma)
```

```
## [1] 3.049619
```

```
Putprice(S0, tau, K, r, sigma)
```

```
## [1] 2.456149
```

Under the given initial conditions, call option price is 3.0496 and the put option price is 2.4561.

b)

```r
Callprice(S0, tau, K, r, sigma) - Putprice(S0, tau, K, r, sigma) - S0 + K*exp(-r*tau)
```

```
## [1] 0
```

The Put-Call parity relation holds.

c)

Get option chains from Yahoo finance by "quantmod":

```r
AAPL.OPTS <- getOptionChain("AAPL", NULL)
C1 <- AAPL.OPTS$Mar.17.2017$calls
C1$Ave.Price <- (C1$Bid + C1$Ask)/2
C1 <- C1[, c(1, 8)]
C2 <- AAPL.OPTS$Apr.21.2017$calls
C2$Ave.Price <- (C2$Bid + C2$Ask)/2
C2 <- C2[, c(1, 8)]
C3 <- AAPL.OPTS$Jul.21.2017$calls
C3$Ave.Price <- (C3$Bid + C3$Ask)/2
C3 <- C3[, c(1, 8)]

temp <- merge(C1, C2, by = "Strike")
calls <- merge(temp, C3, by = "Strike")
colnames(calls) <- c("Strike", "Mar.17.2017", "Apr.21.2017", "Oct.20.2017")
calls <- calls[1:20, ]
```

All call option prices are showed below:

```r
knitr::kable(calls, caption = "Call option prices")
```

Table 1: Call option prices

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
50	82.175	85.775	82.250
60	72.200	72.250	72.275
70	0.000	62.275	62.300
75	60.750	57.300	57.375
80	52.200	52.300	0.000
85	47.200	47.350	47.400
90	45.750	45.875	45.775
95	40.750	40.900	40.875
100	35.775	35.875	36.025
105	30.800	30.725	31.200
110	25.750	25.900	26.425
115	20.800	21.125	21.875
120	15.850	16.225	17.575
125	10.975	11.600	13.625
130	6.375	7.400	10.150

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
135	2.780	4.150	7.200
140	0.855	1.980	4.950
145	0.240	0.845	3.200
150	0.075	0.340	1.990
155	0.025	0.145	1.190

Get the actual stock price:

```
todaystock <- getQuote("AAPL")
S_0 <- todaystock[, 2]
S_0
```

```
## [1] 135.72
```

The actual stock price is 135.72.

Treasury bill rate is 0.005. (https://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.aspx?data=billrates)

$f(\sigma) = C_{BSM} - C_M$ is defined as "fsigma":

```
r <- 0.005
fsigma <- function(sigma, K_i, maturity_i) {
    cc <- calls[K_i, maturity_i + 1]
    K <- calls[K_i, 1]
    if (maturity_i == 1) tau = 23/252 # time to maturity
    else if(maturity_i == 2) tau = 48/252
    else tau = 111/252
    ans <- Callprice(S_0, tau, K, r, sigma) - cc
    return(ans)
}
```

We want to find a interval $[a, a+d]$(d should be small enough) that makes the secant method converge

```
interval <- calls
interval[, 2:4] <- NaN
delta <- 0.1
for(i in 1:20) {
    for(j in 1:3) {
        a <- seq(1, 5, by = delta)
        for(k in a) {
            if (fsigma(k,i,j)*fsigma(k + delta, i, j) < 0) {interval[i,j + 1] <- k}
        }
    }
}
```

```
knitr::kable(interval, caption = "Left side of intervals")
```

Table 2: Left side of intervals

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
50	4.6	3.1	1.9
60	4.2	2.8	1.8
70	NaN	2.6	1.7
75	3.5	2.4	1.6

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
80	3.4	2.3	NaN
85	3.2	2.2	1.5
90	3.0	2.1	1.4
95	2.8	2.0	1.4
100	2.6	1.8	1.4
105	2.4	1.7	1.3
110	2.2	1.6	1.3
115	2.0	1.5	1.3
120	1.8	1.4	1.3
125	1.5	1.3	1.4
130	1.3	1.3	1.4
135	1.0	1.2	1.4
140	NaN	1.2	1.4
145	NaN	1.1	1.5
150	NaN	1.1	1.5
155	NaN	1.1	1.5
Here come	NaN values wh	ich means ther	e is no root for σ in such a conditon.

```r
ImpliedVolatility <- calls
ptm <- proc.time()
count <- 0
for(i in 1:20) {
    for(j in 1:3) {
        a <- interval[i, j + 1]
        if (is.na(a) == T) {
            ImpliedVolatility[i, j + 1] <- NaN
            next
            }
        b <- a + delta
        epsilon <- abs(a - b)
        while(epsilon > 1e-4) {
            count <- count +1
            mid <- (a + b)/2
            if(fsigma(a, i, j)*fsigma(mid, i, j) < 0 ) b <- mid
            else a <- mid
            epsilon <- abs(a - b)
        }
        ImpliedVolatility[i, j+1] <- a
    }
}
count
```

```
## [1] 540
```

```r
ImpliedVolatility
```

```
##    Strike Mar.17.2017 Apr.21.2017 Oct.20.2017
## 1      50    4.651074    3.165820    1.982129
## 2      60    4.212207    2.890430    1.829395
## 3      70         NaN    2.624805    1.701953
## 4      75    3.598926    2.498535    1.646289
## 5      80    3.413477    2.376074         NaN
## 6      85    3.221680    2.256348    1.554395
```

```
## 7      90    3.020117    2.117676    1.470312
## 8      95    2.827539    2.001172    1.437988
## 9     100    2.632910    1.887207    1.412402
## 10    105    2.435059    1.776855    1.394727
## 11    110    2.232617    1.666797    1.385938
## 12    115    2.023047    1.560645    1.384082
## 13    120    1.805176    1.463965    1.390332
## 14    125    1.577344    1.377441    1.404395
## 15    130    1.340723    1.308203    1.425293
## 16    135    1.094629    1.254980    1.453418
## 17    140         NaN    1.219531    1.482617
## 18    145         NaN    1.189844    1.518848
## 19    150         NaN    1.158203    1.555859
## 20    155         NaN    1.110840    1.593652
```

```r
proc.time() - ptm
```

```
##    user  system elapsed
##   0.057   0.001   0.061
```

d)

```r
# Secant
ImpliedVolatility2 <- calls
ptm <- proc.time()
count2 <- 0
for(i in 1:20) {
    for(j in 1:3) {
        x1 <- interval[i, j + 1]
        if (is.na(x1) == T) {
            ImpliedVolatility2[i, j + 1] <- NaN
            next
        }
        x2 <- x1 + delta
        dfxn <- (fsigma(x1, i, j) - fsigma(x2, i, j))/(x1 - x2)
        tang <- fsigma(x2, i, j)/dfxn
        epsilon <- abs(tang)
        while(epsilon > 1e-4) {
            count2 <- count2 + 1
            x1 <- x2
            x2 <- x2 - tang
            dfxn <- (fsigma(x1, i, j) - fsigma(x2, i, j))/(x1 - x2)
            tang <- fsigma(x2, i, j)/dfxn
            epsilon <- abs(tang)
        }
        ImpliedVolatility2[i, j+1] <- x2
    }
}
count2
```

```
## [1] 130
```

```r
ImpliedVolatility2
```

```
##    Strike Mar.17.2017 Apr.21.2017 Oct.20.2017
```

```
## 1      50    4.651052    3.165999    1.982147
## 2      60    4.212284    2.890440    1.829473
## 3      70         NaN    2.624822    1.701987
## 4      75    3.599014    2.498626    1.646355
## 5      80    3.413503    2.376112         NaN
## 6      85    3.221736    2.256419    1.554478
## 7      90    3.020166    2.117669    1.470393
## 8      95    2.827576    2.001201    1.438016
## 9     100    2.632991    1.887266    1.412354
## 10    105    2.435066    1.776905    1.394816
## 11    110    2.232604    1.666869    1.385981
## 12    115    2.023123    1.560673    1.384158
## 13    120    1.805179    1.464013    1.390356
## 14    125    1.577321    1.377475    1.404389
## 15    130    1.340737    1.308118    1.425233
## 16    135    1.094671    1.254990    1.453431
## 17    140         NaN    1.219540    1.482634
## 18    145         NaN    1.189810    1.518856
## 19    150         NaN    1.158217    1.555929
## 20    155         NaN    1.110881    1.593724
```

```r
proc.time() - ptm
```

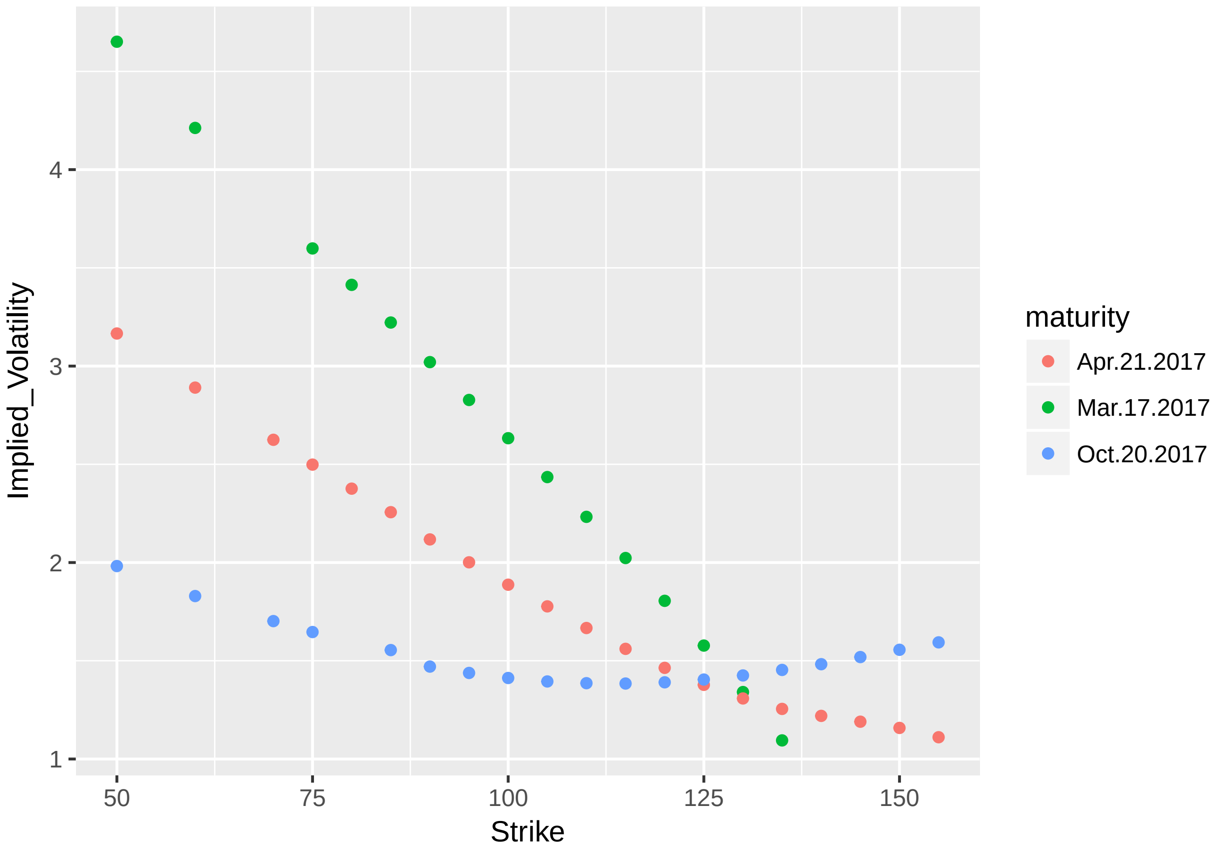
```
##    user  system elapsed
##   0.038   0.000   0.039
```

Both the steps and time spent on secant methods (130, 0.041) is less than those spent on bisection method (540, 0.078). I suppose the reason may be the interval is small enough and the secant method's order of convergence is 1.618 which is superlinear.

Also I found that if we set the initial intervals too large, the secant method seldom converge, although the bisection method always converge if there exist roots. So it is hard for me to compare these two methods when the initial intervals is large. (like [1, 10])

e)

```r
temp1 <- as.matrix(ImpliedVolatility)
temp2 <- rbind(temp1[, 1:2], temp1[, c(1, 3)], temp1[, c(1, 4)])
maturity <- c(rep("Mar.17.2017", 20), rep("Apr.21.2017", 20), rep("Oct.20.2017", 20))
newdf <- data.frame(temp2, maturity)
colnames(newdf)[2] <- "Implied_Volatility"

ggplot(data = newdf, aes(x = Strike, y = Implied_Volatility, colour = maturity)) + geom_point()
```



All call option prices are getting smaller as the Strikes increasing. Longer time to maturity means lower decreasing speed of implied volatility.

Around the actual stock price the implied volatility tend to be equal under different maturities.

This 3d plot cannot be presented in Rmarkdown pdf, but it works in R file.

```
plot3d(ImpliedVolatility$Strike, 1, ImpliedVolatility$Mar.17.2017, col = "blue", ylim = c(.5, 3.5))
points3d(ImpliedVolatility$Strike, 2, ImpliedVolatility$Apr.21.2017, col = "green")
points3d(ImpliedVolatility$Strike, 3, ImpliedVolatility$Oct.20.2017, col = "red")
```

f)

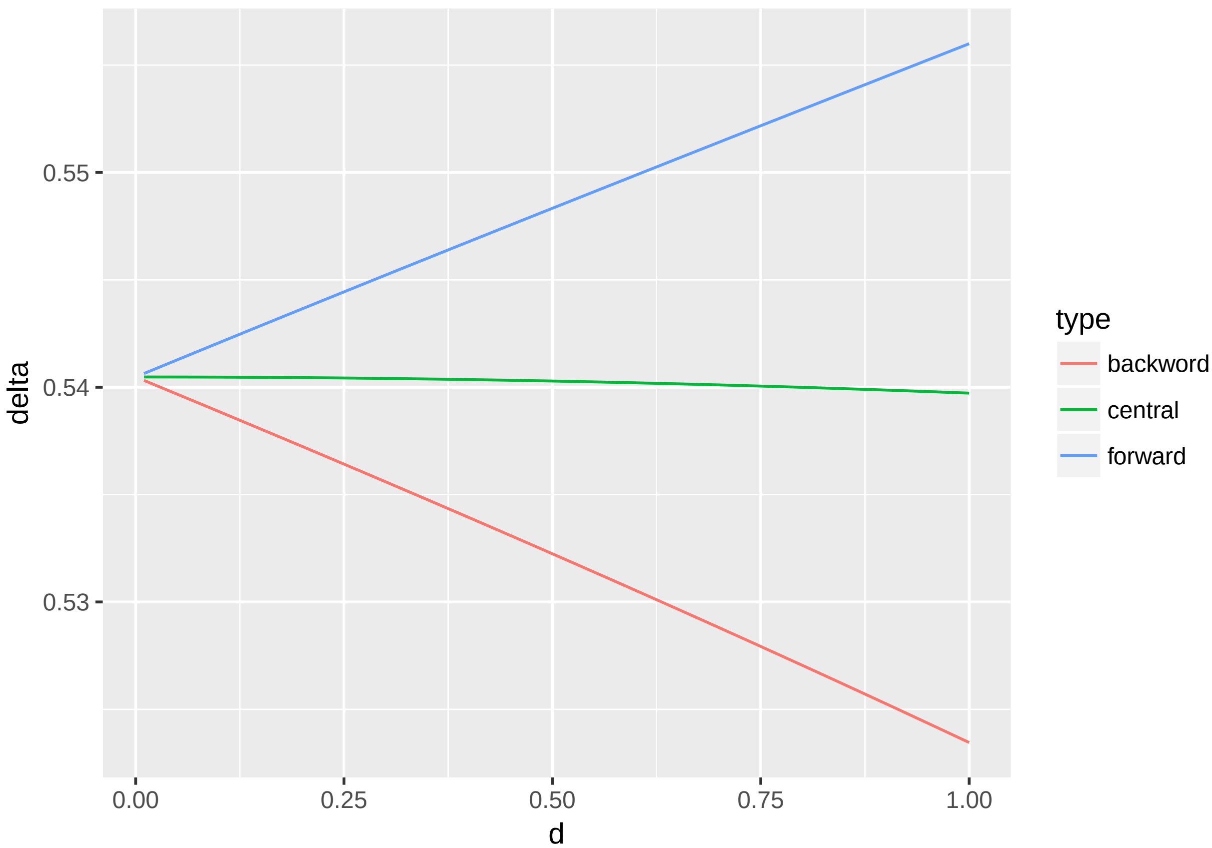
To use 3 methods (forward, backword and central) to evaluate delta, vega and gamma, define functions:

```
Delta1 <- function(d, S0, tau, K, r, sigma, ty) {
    if (ty == "b") D <- (Callprice(S0, tau, K, r, sigma) - Callprice(S0 - d, tau, K, r, sigma))/d
    else if (ty == "f") D <- (Callprice(S0 + d, tau, K, r, sigma) - Callprice(S0, tau, K, r, sigma))/d
    else if (ty == "c") D <- (Callprice(S0 + d, tau, K, r, sigma) - Callprice(S0 - d, tau, K, r, sigma)
    else D <- NaN
    return(D)
}

Vega1 <- function(d, S0, tau, K, r, sigma, ty) {
    if (ty == "b") V <- (Callprice(S0, tau, K, r, sigma) - Callprice(S0, tau, K, r, sigma - d))/d
    else if (ty == "f") V <- (Callprice(S0, tau, K, r, sigma + d) - Callprice(S0, tau, K, r, sigma))/d
    else if (ty == "c") V <- (Callprice(S0, tau, K, r, sigma + d) - Callprice(S0, tau, K, r, sigma - d)
    else V <- NaN
```

```
    return(V)
}

Gamma1 <- function(d, S0, tau, K, r, sigma, ty) {
    if (ty == "b") {G <- (Callprice(S0, tau, K, r, sigma) - 2*Callprice(S0 - d, tau, K, r, sigma) +
                                    Callprice(S0 - 2*d, tau, K, r, sigma))/d^2}
    else if (ty == "f") {G <- (Callprice(S0 + 2*d, tau, K, r, sigma) - 2*Callprice(S0 + d, tau, K, r, si
                                         Callprice(S0, tau, K, r, sigma))/d^2}
    else if (ty == "c") {G <- (Callprice(S0 + d, tau, K, r, sigma) - 2*Callprice(S0, tau, K, r, sigma) +
                                             Callprice(S0 - d, tau, K, r, sigma))/d^2}
    else G <- NaN
    return(G)
}
```

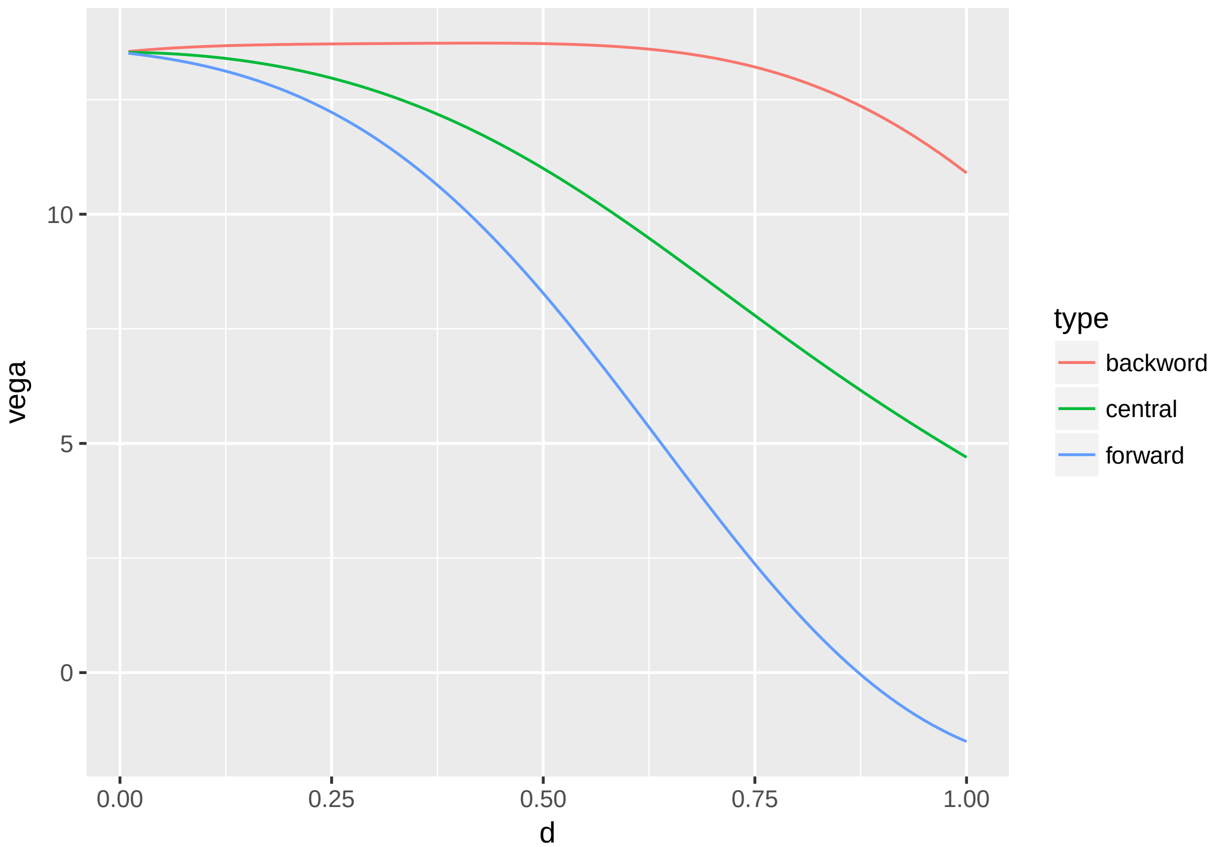
For delta

```
d1 <- seq(1, .01,by=-0.01)
fd_f <- sapply(d1, Delta1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "f")
fd_b <- sapply(d1, Delta1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "b")
fd_c <- sapply(d1, Delta1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "c")

type <- c(rep("forward", length(d1)), rep("backword", length(d1)), rep("central", length(d1)))

delta_dd <- data.frame("d" = rep(d1, 3), "delta" = c(fd_f, fd_b, fd_c), "type" = type)
ggplot(data = delta_dd, aes(x = d, y = delta, colour = type)) + geom_line()
```



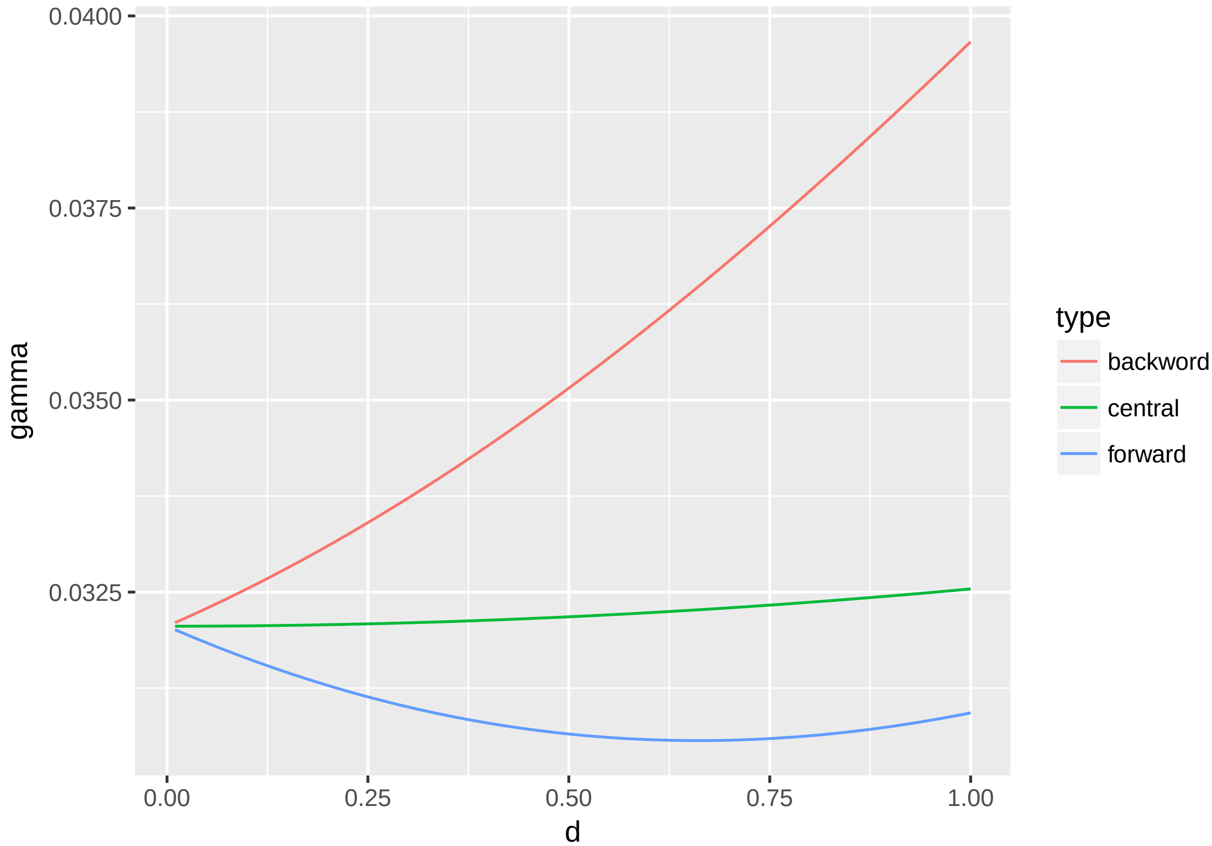
For vega

```
fv_f <- sapply(d1, Vega1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "f")
fv_b <- sapply(d1, Vega1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "b")
fv_c <- sapply(d1, Vega1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "c")

vega_dd <- data.frame("d" = rep(d1, 3), "vega" = c(fv_f, fv_b, fv_c), "type" = type)
ggplot(data = vega_dd, aes(x = d, y = vega, colour = type)) + geom_line()
```



For gamma

```
fg_f <- sapply(d1, Gamma1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "f")
fg_b <- sapply(d1, Gamma1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "b")
fg_c <- sapply(d1, Gamma1, S0=S0, tau=tau, K=K, r=r, sigma=sigma, "c")

gamma_dd <- data.frame("d" = rep(d1, 3), "gamma" = c(fg_f, fg_b, fg_c), "type" = type)
ggplot(data = gamma_dd, aes(x = d, y = gamma, colour = type)) + geom_line()
```



These plots shows that all three parameters converge as d is approaching to 0.

```r
d <- 1e-5 # let d be small enough
Delta1(d, S0, tau, K, r, sigma, "f") #forward
```

```
## [1] 0.5404777
```

```r
Delta1(d, S0, tau, K, r, sigma, "b") #backward
```

```
## [1] 0.5404774
```

```r
Delta1(d, S0, tau, K, r, sigma, "c") #central
```

```
## [1] 0.5404775
```

```r
Vega1(d, S0, tau, K, r, sigma, "f") #forward
```

```
## [1] 13.5322
```

```r
Vega1(d, S0, tau, K, r, sigma, "b") #backword
```

```
## [1] 13.53224
```

```r
Vega1(d, S0, tau, K, r, sigma, "c") #central
```

```
## [1] 13.53222
```

```r
Gamma1(d, S0, tau, K, r, sigma, "f") #forward
```

```
## [1] 0.03211653
```

```r
Gamma1(d, S0, tau, K, r, sigma, "b") #backword
```

```
## [1] 0.03204548
```

```r
Gamma1(d, S0, tau, K, r, sigma, "c") #central
```

```
## [1] 0.03204548
```

So in initial conditon, (delta = 0.5405), (vega = 13.5322), (gamma = 0.0320).

g)

If we apply the implied volatilities, access to the same process as f):

```r
delta_df <- ImpliedVolatility
vega_df <- ImpliedVolatility
gamma_df <- ImpliedVolatility

for (i in 1:20) {
    for (j in 1:3) {
        sigma <- ImpliedVolatility[i, j+1]
        if (is.na(sigma) == T) {
            delta_df[i, j+1] <- NaN
            vega_df[i, j+1] <- NaN
            gamma_df[i, j+1] <- NaN
            break}
        delta_df[i, j+1] <- Delta1(d, S_0, tau, K, r, sigma, "c")
        vega_df[i, j+1] <- Vega1(d, S_0, tau, K, r, sigma, "c")
        gamma_df[i, j+1] <- Gamma1(d, S_0, tau, K, r, sigma, "c")
    }
}
```

The results are showed below:

```r
knitr::kable(delta_df, caption = "Delta")
```

Table 3: Delta

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
50	0.0000000	0.0000000	-0.9861440
60	0.0000000	0.0001906	0.2967363
70	NaN	2.6248047	1.7019531
75	0.0000000	2.4232181	0.9284623
80	0.0000000	5.5183122	NaN
85	0.0000000	4.6416805	0.9827871
90	0.0000006	0.0261756	0.9958261
95	0.0018791	-1.0741389	0.9976333
100	0.3238269	-0.1740800	0.9985007
105	4.1816928	0.5998016	0.9989100
110	3.8854298	0.9037805	0.9990707
115	-1.1062863	0.9809504	0.9991015
120	0.4518848	0.9962629	0.9989934
125	0.9750984	0.9992039	0.9987019
130	0.9995945	0.9997784	0.9981116
135	0.9999961	0.9999184	0.9968929
140	NaN	1.2195312	1.4826172
145	NaN	1.1898438	1.5188477
150	NaN	1.1582031	1.5558594

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
155	NaN	1.1108398	1.5936523

```r
knitr::kable(vega_df, caption = "Vega")
```

Table 4: Vega

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
50	0.0000000	0.0000000	70.0798841
60	0.0000000	-0.0088265	22.6284768
70	NaN	2.6248047	1.7019531
75	0.0000000	-94.9244655	2.0721050
80	0.0000000	-198.8011377	NaN
85	0.0000000	-139.2146860	0.4720215
90	-0.0000283	40.9052278	0.1086545
95	-0.0850068	74.1611422	0.0603535
100	-13.5451414	39.0607239	0.0376064
105	-157.6137374	12.4967890	0.0270266
110	-107.0169966	2.8205666	0.0229097
115	76.5628729	0.5243691	0.0221223
120	17.3941584	0.0968948	0.0248851
125	0.6923946	0.0195155	0.0323901
130	0.0096990	0.0051838	0.0477652
135	0.0000780	0.0018384	0.0800215
140	NaN	1.2195312	1.4826172
145	NaN	1.1898438	1.5188477
150	NaN	1.1582031	1.5558594
155	NaN	1.1108398	1.5936523

```r
knitr::kable(gamma_df, caption = "Gamma")
```

Table 5: Gamma

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
50	0.0000000	0.0000000	0.1645617
60	0.0000000	0.0001592	0.2195577
70	NaN	2.6248047	1.7019531
75	0.0000000	0.6467538	0.0365219
80	0.0000000	0.4479617	NaN
85	0.0000000	-0.7195666	0.0105160
90	0.0000006	-0.6806289	0.0029843
95	0.0013920	0.0888178	0.0017053
100	0.1486344	0.2782485	0.0011369
105	0.7247358	0.1496403	0.0007105
110	-0.8711964	0.0474643	0.0011369
115	-0.0221689	0.0110845	0.0012790
120	0.1870148	0.0022737	0.0001421
125	0.0142109	0.0004263	0.0012790
130	0.0002842	-0.0004263	0.0015632
135	0.0007105	0.0002842	0.0019895
140	NaN	1.2195312	1.4826172

Strike	Mar.17.2017	Apr.21.2017	Oct.20.2017
145	NaN	1.1898438	1.5188477
150	NaN	1.1582031	1.5558594
155	NaN	1.1108398	1.5936523

Question 2

a)

Define the the real–valued function:

```
fx <- function(x) {
    if (x == 0) fx <- 1
    else fx <- sin(x)/x
    return(fx)
}
```

For the trapezoidal rule:

```
trapezoidal <- function(n, a) {
    h <- 2*a/(n-1)
    x <- seq(-a, a, by = h)
    wn <- rep(h, n)
    wn[1] <- h/2
    wn[n] <- h/2
    fn <- sapply(x, fx)
    ans <- sum(fn*wn)
    return(ans)
}
trapezoidal(1e6, 1e6)
```

```
## [1] 3.141591
```

For the Simpson's quadrature rule

```
simpson <- function(n, a) {
    h <- 2*a/(n-1)
    x <- seq(-a, a, by = h)
    wn <- rep(c(0, 4*h/3), floor(n/2)) + rep(c(2*h/3, 0), floor(n/2))
    wn[1] <- h/3
    wn[n] <- h/3
    fn <- sapply(x, fx)
    ans <- sum(fn*wn)
    return(ans)
}
simpson(1e6, 1e6)
```

```
## [1] 3.141592
```

Both the results are close to π.

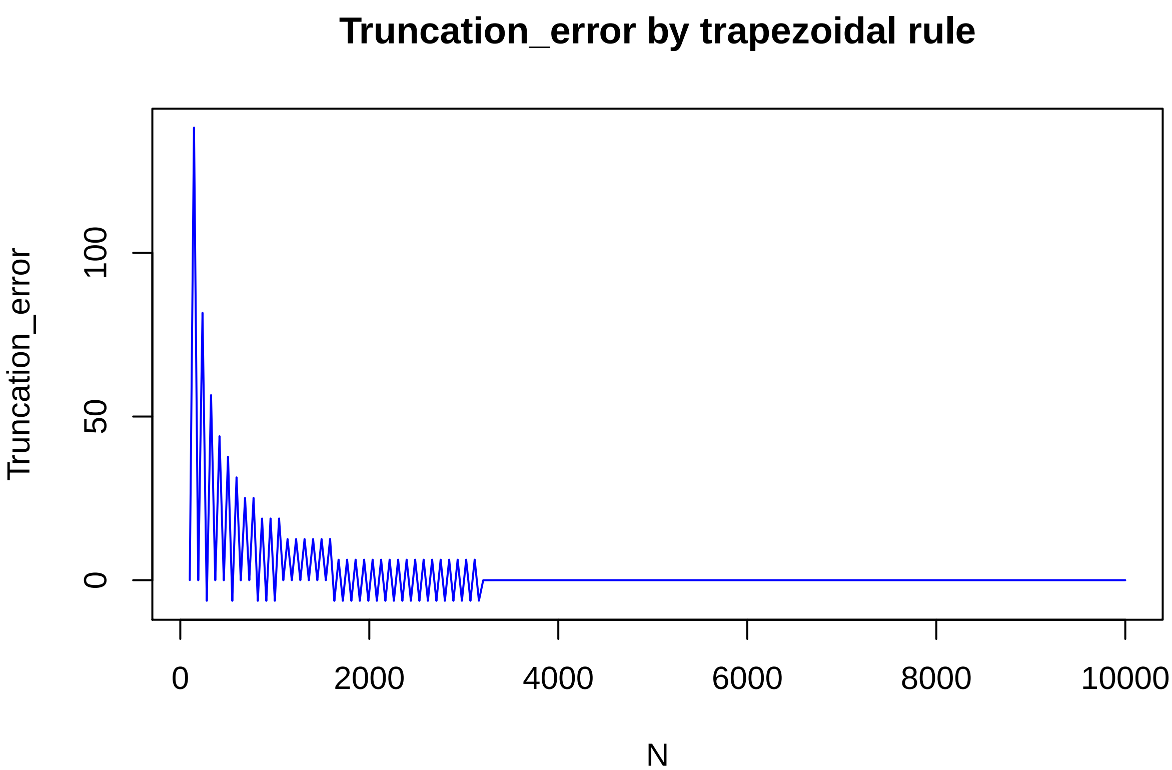
b)

To compute the truncation errors, define functions "TE_trape" and "TE_simp" respectively:

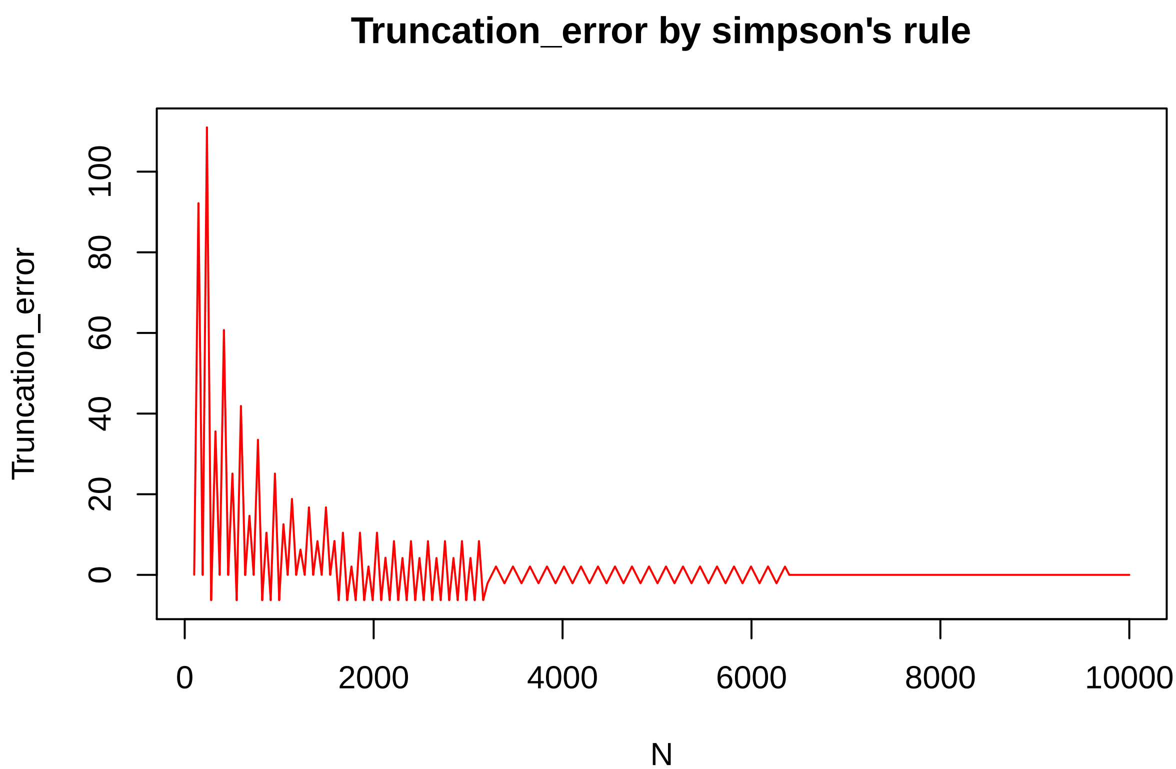
```r
TE_trape <- function(n, a) {
    temp <- trapezoidal(n, a)
    te <- temp - pi
    return(te)
}

TE_simp <- function(n, a) {
    temp <- simpson(n, a)
    te <- temp - pi
    return(te)
}
```

(1) Fix $a = 10^4$

```r
# Truncation error of trapezoidal rule
a <- 1e4
N <- seq(100, 1e4, by = 45)
Truncation_error <- sapply(N, TE_trape, a=a)
plot(N, Truncation_error, type = "l", col = "blue", main = "Truncation_error by trapezoidal rule")
```



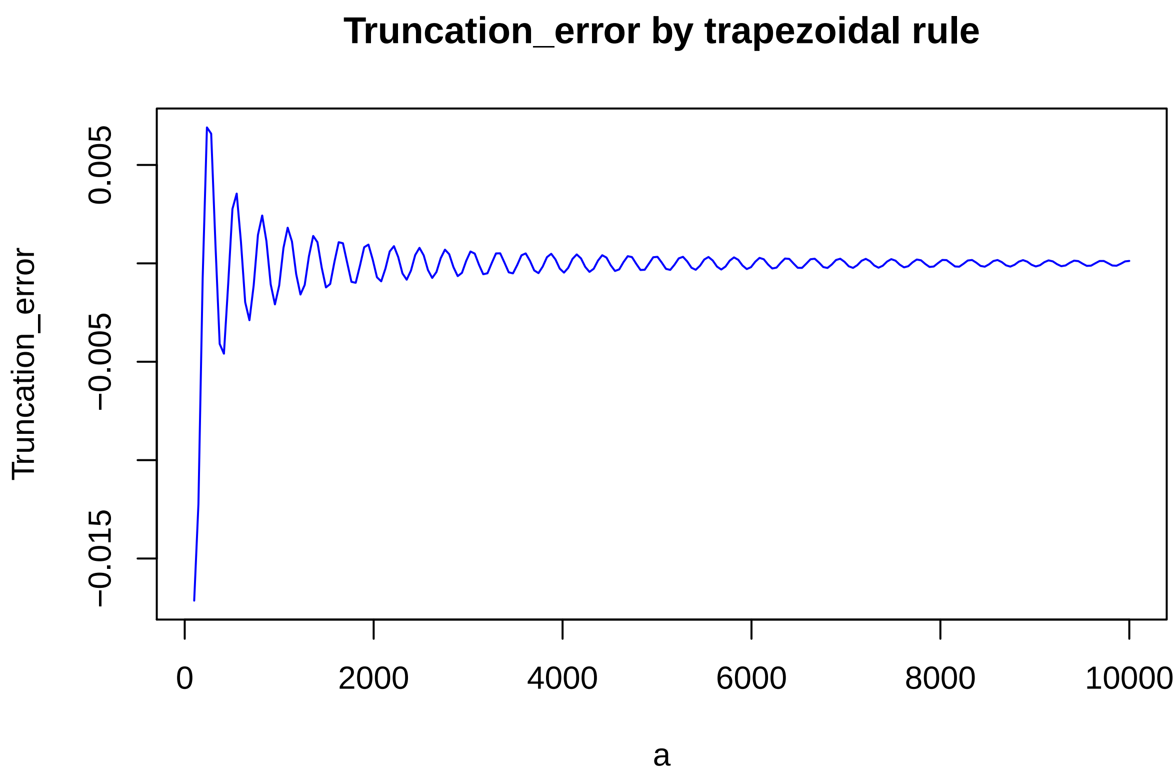

```r
# Truncation error of Simpson's rule
a <- 1e4
N <- seq(100, 1e4, by = 45)
Truncation_error <- sapply(N, TE_simp, a=a)
plot(N, Truncation_error, type = "l", col = "red", main = "Truncation_error by simpson's rule")
```



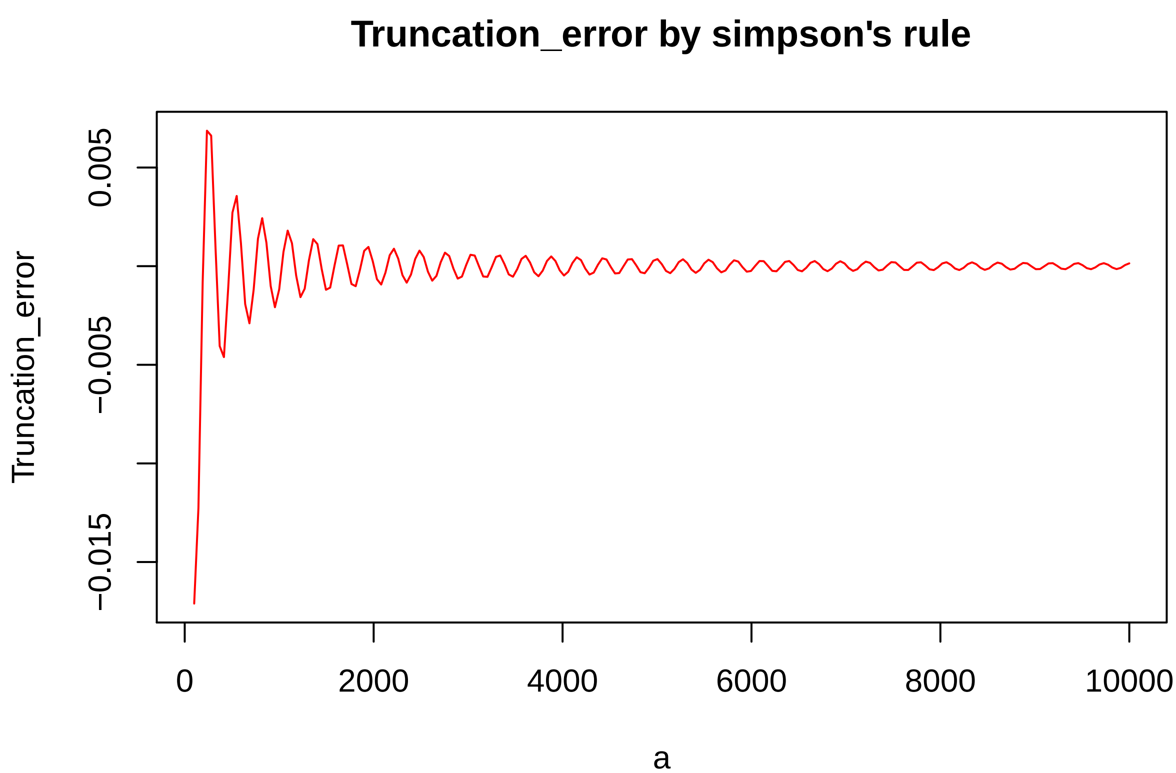
For both rules, the truncation error decrease progressively as n gets larger and larger. Besides, the simpson's rule converge later and fluctuate more.

(2) fix $N = 10^6$:

```
# Truncation error of trapezoidal rule
N <- 1e4
a <- seq(1e2, 1e4, by = 45)
Truncation_error <- sapply(a, TE_trape, n = N)
plot(a, Truncation_error, type = "l", col = "blue", main = "Truncation_error by trapezoidal rule")
```




```r
# Truncation error of Simpson's rule
N <- 1e4
a <- seq(1e2, 1e4, by = 45)
Truncation_error <- sapply(a, TE_simp, n = N)
plot(a, Truncation_error, type = "l", col = "red", main = "Truncation_error by simpson's rule")
```



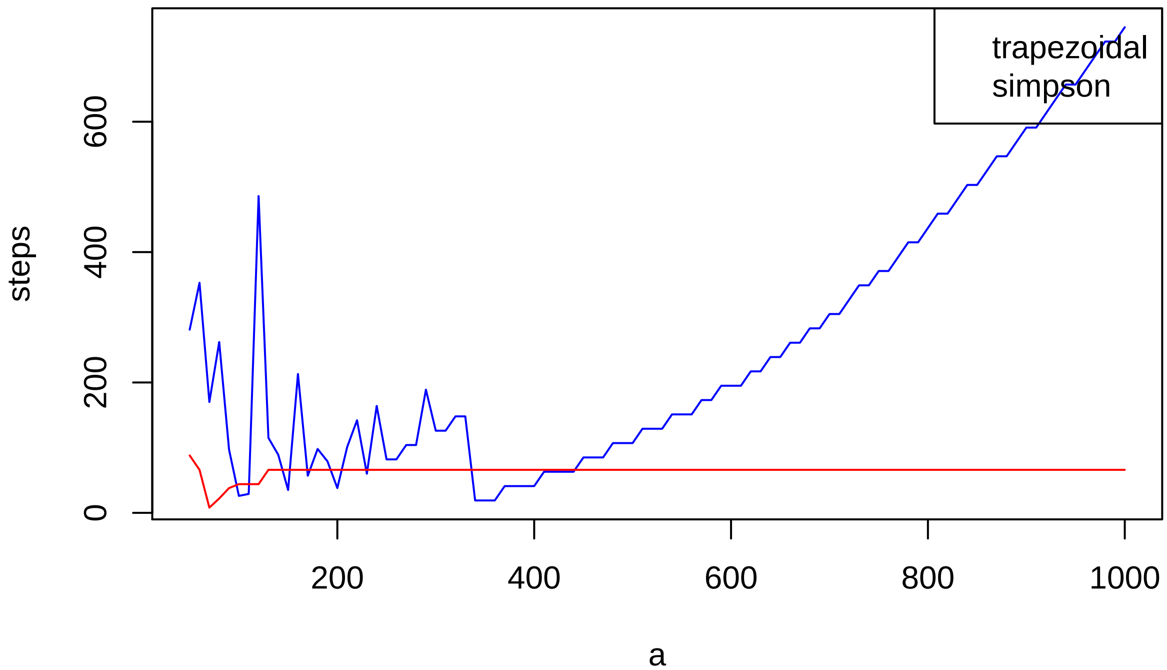
For

both rules, the truncation error decrease as a gets larger and larger and fluctuate around 0.

c)

```
steps_trape <- function(a) {
    diff <- abs(trapezoidal(a, 102)-trapezoidal(a, 101))
    n0 = 103
    n = n0
    epsilonI <- 1e-4
    while(diff > epsilonI) {
        diff <- abs(trapezoidal(a, n)-trapezoidal(a, n-1))
        n <- n + 1
    }
    count <- n - n0 +1
    return(count)
}

steps_simp <- function(a) {
    diff <- abs(simpson(a, 102)-simpson(a, 101))
    n0 = 103
    n = n0
    epsilonI <- 1e-4
    while(diff > epsilonI) {
        diff <- abs(simpson(a, n)-simpson(a, n-1))
        n <- n + 1
    }
    count <- n - n0 +1
    return(count)
}
```

```
a <- seq(50, 1000, by=10)
steps_by_tr <- sapply(a, steps_trape)
steps_by_si <- sapply(a, steps_simp)
plot(a, steps_by_si, type = "l", col = "blue",  ylab = "steps")
lines(a, steps_by_tr, col = "red")
legend("topright", c("trapezoidal", "simpson"),
       col=c("red", "blue"))
```



When a increases, steps processes by simpson's rule also increase but can still converge when steps are large enough. While steps by trapezoidal rule will converge to a number around 80 in this case.

d)

For another integrand:

```
fx2 <- function(x) {
    f <- 1 + exp(-x)*sin(8*x^(2/3))
}
```

```
trapezoidal2 <- function(n) {
    a1 <- 0
    a2 <- 2
    x <- seq(a1, a2, length.out = n)
    h <- x[2] - x[1]
    wn <- rep(h, n)
    wn[1] <- h/2
    wn[n] <- h/2
    fn <- sapply(x, fx2)
    ans <- sum(fn*wn)
    return(ans)
}

simpson2 <- function(n) {
    a1 <- 0
    a2 <- 2
    x <- seq(a1, a2, length.out = n)
    h <- x[2] - x[1]
    wn <- rep(c(0, 4*h/3), floor(n/2)) + rep(c(2*h/3, 0), floor(n/2))
    wn[1] <- h/3
    wn[n] <- h/3
    fn <- sapply(x, fx2)
```

```
    ans <- sum(fn*wn)
    return(ans)
}

epsilon <- 1
N <- 11
I1 <- trapezoidal2(10)
while (epsilon > 1e-4) {
    I2 <- trapezoidal2(N)
    epsilon <- abs(I1 - I2)
    I1 <- I2
    N <- N + 1
}
print(I1)
```

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

```
epsilon <- 1
N <- 11
I1 <- simpson2(10)
while (epsilon > 1e-4) {
    I2 <- simpson2(N)
    epsilon <- abs(I1 - I2)
    I1 <- I2
    N <- N + 1
}
print(I1)
```

```
## [1] 2.016179
```

The integral by trapezoidal rule converges to 2.0126 while the integral by simpson's rule converges to 2.0162. These two integral are very close.

Question 3

```
S0 <- 1
TT <- 5
t <- 0
tau <- TT-t
r <- 0
V0 <- 0.1
theta <- 0.1
sigma <- 0.2
rho <- -0.3
lambda <- 0
q <- 0 # Divident yield

a <- 1000
N <- 1001 # must be an integer

# The Simpson's rule
simpsonReal <- function(n, a, i, K, kappa) {
    x <- seq(1/a, a, length.out = n)
    h <- x[2] - x[1]
```

```
    wn <- rep(c(2*h/3, 4*h/3), floor(n/2))
    wn[1] <- h/3
    wn[n] <- h/3
    fn <- sapply(x, RE, i = i, K = K, kappa = kappa)
    ans <- sum(fn*wn)
    return(ans)
}

ub <- function(i, kappa) {
    if (i == 1) {
        u <- 0.5
        b <- kappa + lambda - rho*sigma
    }
    else if (i == 2) {
        u <- -0.5
        b <- kappa + lambda
    }
    ub <- c(u, b)
    return(ub)
}

C <- function(x, i, kappa) {
    u <- ub(i, kappa)[1]
    b <- ub(i, kappa)[2]
    d <- sqrt((rho*sigma*x*1i - b)^2 - sigma^2*(2*u*x*1i - x^2))
    g <- (b - rho*sigma*x*1i + d)/(b - rho*sigma*x*1i - d)
    cc <- (r - q)*x*tau*1i +
        kappa*theta/sigma^2*((b - rho*sigma*x*1i + d)*tau - 2*log((1 - g*exp(d*tau))/(1 - g)))
    return(cc)
}

D <- function(x, i, kappa) {
    u <- ub(i, kappa)[1]
    b <- ub(i, kappa)[2]
    d <- sqrt((rho*sigma*x*1i - b)^2 - sigma^2*(2*u*x*1i - x^2))
    g <- (b - rho*sigma*x*1i + d)/(b - rho*sigma*x*1i - d)
    dd <- (b - rho*sigma*x*1i + d)/sigma^2*(1 - exp(d*tau))/(1 - g*exp(d*tau))
    return(dd)
}

phi <- function(x, i, kappa) {
    phi <- exp(C(x, i, kappa) + D(x, i, kappa)*V0 + 1i*x*log(S0))
    return(phi)
}

RE <- function(x, i, K, kappa) {
    temp <- exp(-1i*x*log(K))*phi(x, i, kappa)/(1i*x)
    RE <- Re(temp)
    RE[is.nan(RE)] = 0
    return(RE)
}

Calloption <- function(N, a, K, kappa) {
```

```r
    P1 <- 1/2 + 1/pi*simpsonReal(N, a, 1, K, kappa)
    P2 <- 1/2 + 1/pi*simpsonReal(N, a, 2, K, kappa)
    C <- S0*P1 - K*exp(-(r-q)*tau)*P2
    return(C)
}

K <- c(0.5, 0.75, 1, 1.25, 1.5)
kappa <- c(1, 2, 4)

temp <- as.data.frame(matrix(rep(0, 15), nrow = 5))
comparetable <- data.frame(temp)
rownames(comparetable) <- as.character(K)

for (i in 1:5) {
    for (j in 1:3) {
        k <- K[i]
        kap <- kappa[j]
        Call <- Calloption(N, a, k, kap)
        comparetable[i, j] <- Call
    }
}
Real_value <- c(0.543017, 0.385109, 0.273303, 0.195434, 0.14121)
comparetable <- data.frame(comparetable, Real_value)
colnames(comparetable)[1:3] <- c("kappa=1","kappa=2","kappa=4")
```

The result is showed below:

```r
knitr::kable(comparetable, caption = "Table")
```

Table 6: Table

	kappa=1	kappa=2	kappa=4	Real_value
0.5	0.5432728	0.5424644	0.5418507	0.543017
0.75	0.3842401	0.3851181	0.3853539	0.385109
1	0.2708477	0.2737037	0.2750267	0.273303
1.25	0.1916967	0.1959883	0.1981810	0.195434
1.5	0.1365473	0.1416323	0.1443799	0.141210