European and Asian Options Pricing Using CRR Model

Computational Finance Student Organization

Jan Kościałkowski

1 November 2017

```
library(ggplot2)
library(dplyr)
library(tidyr)
```

The purpose of this document is to present CRR model option pricing methods, in R. European and Asian call options were considered and for each case an adequate function was implemented.

European option pricing

In this case the pricing problem reduces to determining the expected value of option payoff and then discounting to time 0. This does not pose any difficulties as price up- and downshift magnitudes and probabilities are determined by the input arguments and the result depends solely on the underlying price at expiration time. European option pricing using the CRR model for parameters input by the user is implemented by the function below.

```
priceEuropean <- function(S_0, # underlying price at time 0
                          K, # strike price
                          r, # interest intensity
                          sigma, # volatility
                          t, # time until option expiration
                          n) { # number of subintervals of [0,t]

    delta_n <- t/n

    # Checking whether the CRR model will work
    if(delta_n >= sigma^2/r^2) stop("Incorrect input data!")

    u_n <- exp(sigma * sqrt(delta_n))
    d_n <- exp(- sigma * sqrt(delta_n))

    p_n <- (exp(r * delta_n) - d_n)/(u_n - d_n)

    exp(-r * t) * sum(choose(n, 0:n) * p_n^(0:n) * (1 - p_n)^(n:0)
                      * sapply(S_0 * u_n^(0:n) * d_n^(n:0) - K, max, 0))
}
```

Asian option pricing

This case is slightly more complicated - derivation of an analytical formula for option price in this model would be possible yet arduous. Instead, a Monte Carlo approach can be applied - one can simulate a large number of price process trajectories basing on the binomial model, calculate payoff at time `t` and then average all payoffs and discount to time 0.

```
priceAsian <- function(S_0, # underlying price
                       K, # strike price
                       r, # interest intensity
                       sigma, # volatility
                       t, # time until option expiration
```

```
                    n, # number of subintervals of [0,t]
                    n_average, # average over how many points in Asian option payoff
                    n_MC, # number of Monte Carlo iterations
                    plotSimulations = TRUE) { # whether to present simulations on a graph

    delta_n <- t/n
    u_n <- exp(sigma * sqrt(delta_n))
    d_n <- exp(- sigma * sqrt(delta_n))
    p_n <- (exp(r * delta_n) - d_n)/(u_n - d_n)

    # Checking whether the CRR model will work
    if(delta_n >= sigma^2/r^2) stop("Incorrect input data!")

    plotTrajectories <- NULL
    price <- 0
    traject <- vector()

    for(i in 1:n_MC) {

        # Generating price 'ups'
        traject[1] <- rbinom(n = 1, size = 1, prob = p_n)
        for(j in 2:n) traject[j] <- traject[j - 1] + rbinom(n = 1, size = 1, prob = p_n)

        # Calculating price for each moment using generated shifts
        traject <- S_0 * u_n ^ traject * d_n ^ (1:n - traject)

        # Saving some trajectories for plotting
        if(plotSimulations & i%%500 == 0) plotTrajectories <- cbind(plotTrajectories, traject)

        # Payoff for each trajectory is included in the final price
        price <- price + 1/n_MC * max(mean(traject[floor(n/n_average*1:n_average)]) - K, 0)
    }

    # Discounting to time 0
    price <- exp(- r * t) * price

    if(plotSimulations) {
        plotTrajectories <- as.data.frame(plotTrajectories)
        colnames(plotTrajectories) <- 1:ncol(plotTrajectories)
        plotTrajectories <- plotTrajectories %>%
            gather(key = Trajektoria, value = Cena) %>%
            mutate(Czas = rep(t/n*1:n, ncol(plotTrajectories)))

        g <- ggplot(data = plotTrajectories,
                    mapping = aes(x = Czas, y = Cena, colour = Trajektoria),
                    environment = environment())
        g <- g + geom_line() + geom_hline(yintercept = K) + theme(legend.position = "none") +
            scale_color_brewer(palette = "Spectral")
        return(list(price = price, plot = g))
    }

    price
}
```

An exemplary use of the above function:

```
set.seed(1234)
priceAsian(80, 90, .05, .1, 1, 300, 100, 3000, TRUE)
```

Monte Carlo methods tend to be time-consuming, so the dependence of the computation time on `n_MC` was investigated, ceteris paribus.

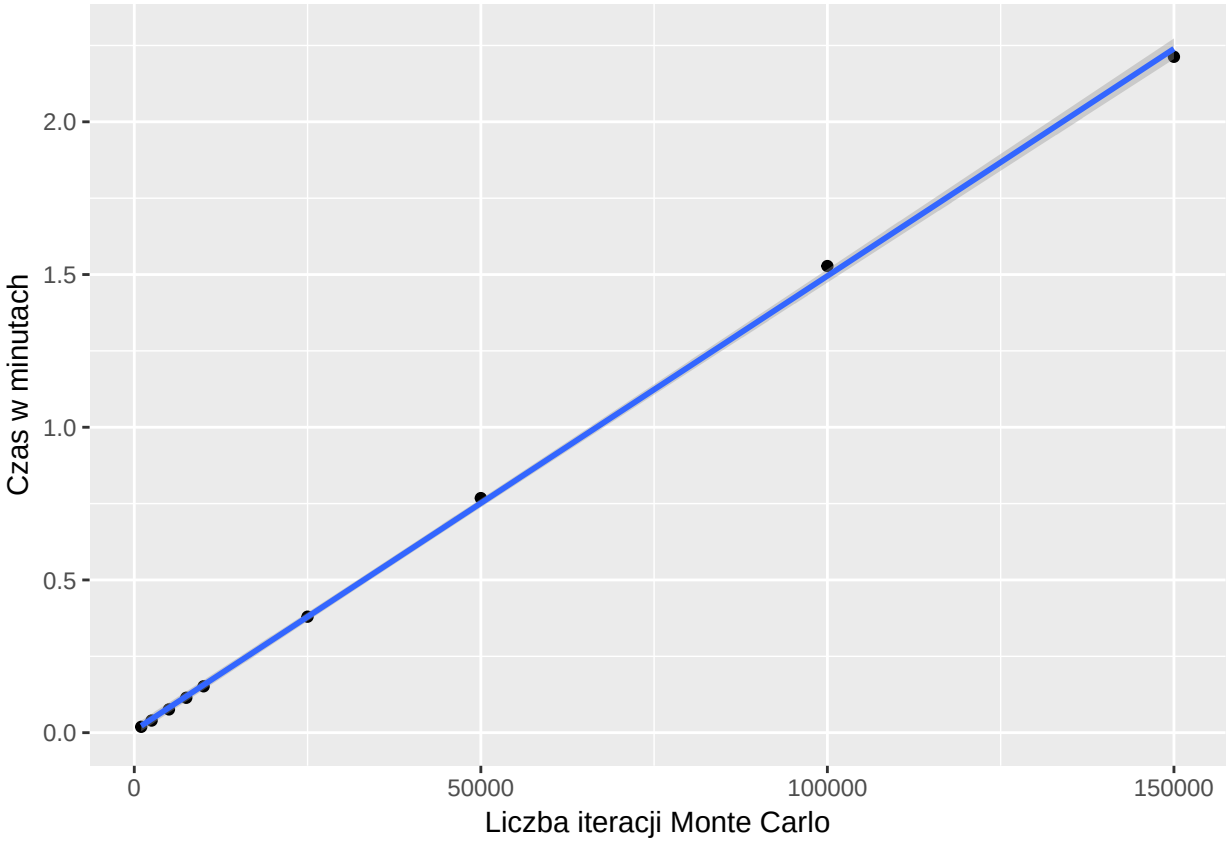
```
set.seed(123)
n_MC <- c(1000, 2500, 5000, 7500, 10000, 25000, 50000, 100000, 150000)

time <- vector()

for(i in 1:length(n_MC)) {
    start <- Sys.time()
    priceAsian(100, 80, .05, .1, 1, 500, 30, n_MC[i], FALSE)
    end <- Sys.time()
    time[i] <- difftime(end, start, units = "mins")
}
```

```
library(ggplot2)

qplot(x = n_MC, y = time, xlab = "Liczba iteracji Monte Carlo", ylab = "Czas w minutach") +
    geom_smooth(method = "lm")
```



Execution time seems to increase linearly with the number of Monte Carlo iterations, which is a very encouraging result.