Cox, Ross, Rubinstein model

Márton Zörényi, Peter Pavicic, Gábor Aurél Antal

The model

The simplest model for pricing options and derivatives is the binomial model, also called model of Cox, Ross and Rubinstein (CRR model). The model consists of two assets, the underlying (the risky asset) $(S_t)_{t=0}^{T}$ and the bank account (the riskless asset) $(B_t)_t^T = 0$.

The project

We consider a CRR-model. Let n denote the number of periods. We chose n to be large, using $n = 100$ for our purposes. Let $\sigma > 0$, for instance $\sigma = 0.5$, let the up- and down-movements be $U = e^{\sigma/\sqrt{n}}, D = e^{-\sigma/\sqrt{n}}$ and the value of the underlying at date $t = 0$ be $S_0 = 100$. We assume that $r = 0$, i.e., prices and discounted prices are the same. European options are exersized at $T = n$. Let A denote an Asian option, with payoff $C_n = (S_n - \bar{S_n})^+$. $\bar{S}_n$ is the mean of prices $S_0, \ldots, S_n$. This Asian option allows to buy S_n in time n for the average price over the period $0, 1, \ldots, n$.

CRR parametres

- $n = 100$ (Number of periods)
- $\sigma = 0.5$ (Variance)
- $U = e^{\sigma/\sqrt{n}}, D = e^{-\sigma/\sqrt{n}}$ (Up and Down nodes' functions)
- $S_0 = 100$ (Starting value of risky asset)
- $r_f = 0$ (risk-free rate)
- A (Asian option)
- $C_n = (S_n - \bar{S_n})^+$ (Payoff function)
- $\bar{S_n}$ (Average price of option)

Given that $r_f = 0$ there is no discounting and therefore $\bar{U} = U$ as well as $\bar{D} = D$.

Option payoffs

In order to find the payoff of the option in one scenario we do the following:

1. Given the data calculate π, the risk-neutral probability of the stock price moving up, given by: $\frac{1-\bar{D}}{\bar{U}-\bar{D}}$.
2. Generate random up- and down-movements drawn with probability π resp. $1 - \pi$.
3. Run through all time periods $t = 0, \ldots, n$, in each period saving the current price of the underlying S_t and the mean of all stock prices up to that point, $\bar{S}_{0,\ldots,t}$.
4. Return the payoff given by $C_n = (S_n - \bar{S_n})^+ = \max\{S_n - \bar{S_n}, 0\}$.

To do so we wrote the following R code:

```r
# Function to get the payoff once
optionPayoff <- function(n, sigma = 0.5) {
  # input
  # n: number of periods


  # given:
  rf <- 0
  S <- S0 <- 100
  u <- exp(sigma / sqrt(n))
  d <- 1 / u

  rnprob <- (1-d)/(u-d)
  cumulativeMean <- S0

  # vector with True if stock moves up, False otherwise
  movements <- sample(c(T, F), n, replace = TRUE, prob = c(rnprob, 1-rnprob))

  for (i in 1:n) {
    S <- ifelse(movements[i], S*u, S*d)
    cumulativeMean <- (cumulativeMean * (i - 1) + S) / i
  }

  return(max(S - cumulativeMean, 0))
}
```

Monte Carlo simulation

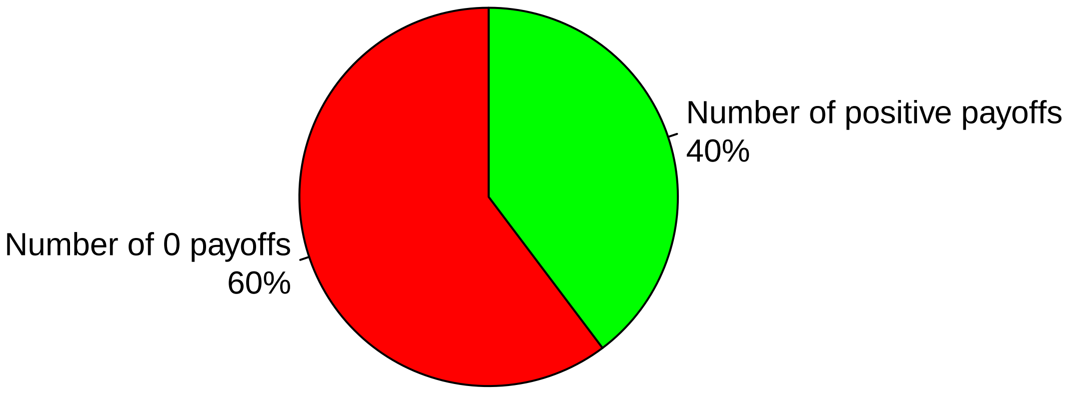
We perform a Monte Carlo simulation which yields the price of the option, A_0 calculated as the mean of all payoffs C, the standard deviation of the payoffs, as well as the number of positive payoffs.

In the following code we define a function which runs `optionPayoff` N times with parametres n and σ to calculate the desired results.

```r
monteCarlo <- function(N, n = 100, sigma = 0.5) {
  # input
  # N: sample size
  # n: number of periods
  # sigma: volatility in asset

  payoffs <- c()

  # payoffMeans <- optionPayoff(n, sigma)
  for (i in 1:N) {
    payoffs <- c(payoffs, optionPayoff(n, sigma))
  }

  price <- mean(payoffs)
  st_dev <- sd(payoffs)

  numOfPositives <- sum(payoffs > 0)


  cat("We observed the average price to be:", price,
      "\nand the standard deviation of: ", st_dev,
      "\nin total: ", numOfPositives, "of the payoffs were positive\n")

  return(c(price, st_dev, numOfPositives))
}

results <- monteCarlo(1e5)
```

```
## We observed the average price to be: 11.40378
## and the standard deviation of:  23.7699
## in total:  39725 of the payoffs were positive
```

```
price <- results[1]
st_dev <- results[2]
positives <- results[3]

positivePercent <- 100 * round(positives/1e5, 2)

pie(c(positives, 1e5 - positives),
    c( paste0("Number of positive payoffs\n", positivePercent, "%")
      , paste0("Number of 0 payoffs\n", 100 - positivePercent, "%")),
    clockwise = TRUE, col=c("green", "red"))
```



Confidence intervals

Compute the confidence intervals

The formula to compute confidence intervals is:

$$CI = \overline{x} \pm z \cdot \frac{\sigma}{\sqrt{n}}$$

where $\overline{x}$ is the arithmetic mean, which in our example is the average payoff $\bar{C}_{0,\ldots,N}$. n is the sample size, which in our case was $N = 10^5$, and σ is the standard deviation of the payoffs which was computed in the `monteCarlo` function.

z is dependent on the confidence level and, with the standard deviation and the sample size known, determines the length of the confidence interval. It is calculated by using the quantile function of the Gaussian distribution.

In order to be able to quickly compute the confidence intervals for every confidence level, we have written a function:

```r
# 95% confidence interval:
confidence_interval <- function(xbar, n, sigma, confidenceLevel = 0.95) {
  alpha <- 1 - confidenceLevel
  tails <- qnorm(1 - alpha / 2) * sigma / sqrt(n)
  return(xbar + c(-tails, tails))
}
```

We have set the default confidence level to be 95%. This means, that if we do not specify the confidence interval in our command, the output will be a 95% confidence interval. The output of the function will be a vector where the entries in the first and second columns represent the lower and upper bounds of the confidence interval.

```r
confidence_interval(price, 1e5, st_dev)
```

```
## [1] 11.25645 11.55110
```

This indicates that with a probability of 95% the price will lie somewhere between these values.

Optimal sample size

Compute the sample size N that would be necessary to estimate the price up to ± 1.

Hence, we need to find a confidence interval with length 2, symmetrically distributed around $\overline{x}$. To get a confidence interval with length 2, the following property must hold:

$$z \cdot \frac{\sigma}{\sqrt{n}} = 1$$

Therefore, the sample size must be:

$$n = (z \cdot \sigma)^2$$

In our code, we again wrote a function so that we can easily compute the optimal sample size for other confidence levels as well.

```r
# given sigma and the confidence level we can calculate the sample size
# if the CI should be 2 wide.
samplesize <- function(sigma, confidenceLevel = 0.95) {
  alpha <- 1 - confidenceLevel

  res <- (qnorm(1 - alpha / 2) * sigma)^2
  cat("We found the optimal sample size to be: ", res, "\n")
  res
}
```

Now we run the command to see what the optimal sample size is.

```r
optimalSampleSize <- samplesize(st_dev)
```

```
## We found the optimal sample size to be:  2170.455
```

We run another Monte Carlo simulation with the optimal sample size.

```r
results <- monteCarlo(optimalSampleSize)
```

```
## We observed the average price to be: 11.28538
## and the standard deviation of:  22.49454
## in total:  868 of the payoffs were positive
```

Now, when we compute the confidence interval, we get one which is close to $\overline{x} \pm 1$. However, it is not exactly of length 2 because the standard deviation of the payoffs changes when running another MC-simulation.

```r
confidence_interval(results[1], optimalSampleSize, results[2])
```

```
## [1] 10.33903 12.23172
```

Price A_0 dependency on σ

Analyse how the price A_0 depends on σ: Repeat the experiment for different choices of σ. Show the results by an appropriate plot.

For this task we chose barplots that can display multiple choices of σ values and their effects on the price A_0 at the same time.

```r
a <- setNames(monteCarlo(optimalSampleSize, sigma = 0.1)[c(1, 2)], "                    0.1")
```

```
## We observed the average price to be: 2.252263
## and the standard deviation of:  3.600192
## in total:  1010 of the payoffs were positive
```

```r
b <- setNames(monteCarlo(optimalSampleSize, sigma = 0.5)[c(1, 2)], "              0.5")
```

```
## We observed the average price to be: 10.07114
## and the standard deviation of:  21.13061
## in total:  809 of the payoffs were positive
```

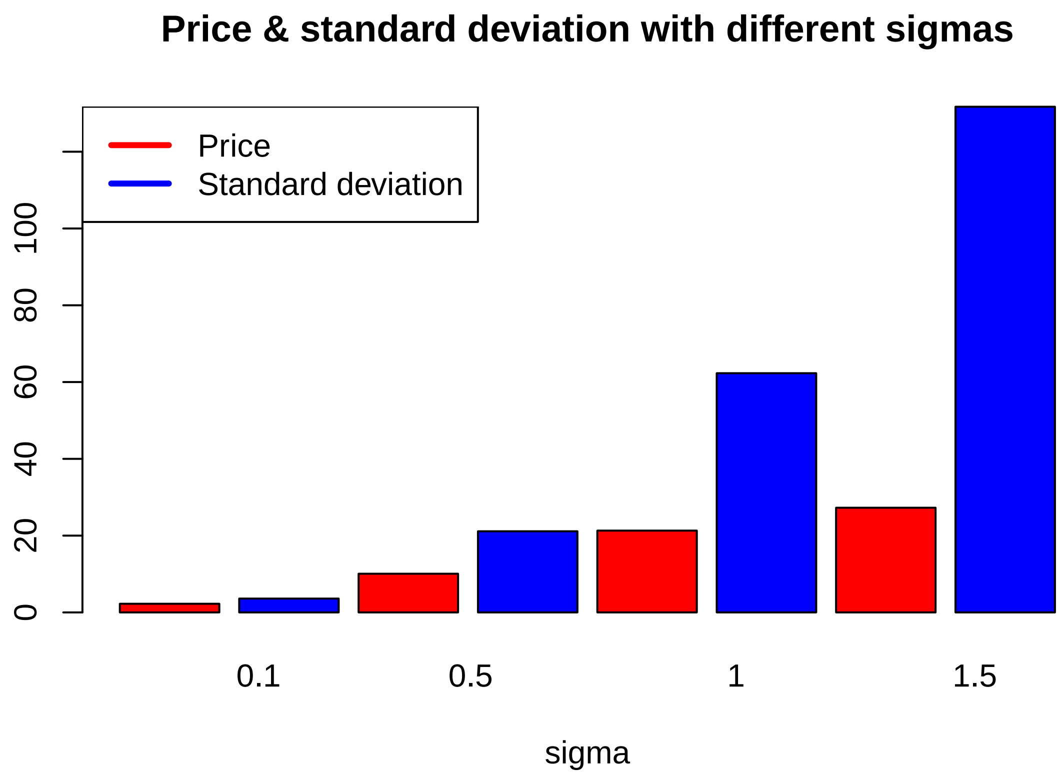
```
c <- setNames(monteCarlo(optimalSampleSize, sigma = 1)[c(1, 2)], "                    1")
```

```
## We observed the average price to be: 21.30122
## and the standard deviation of:  62.30097
## in total:  650 of the payoffs were positive
```

```
d <- setNames(monteCarlo(optimalSampleSize, sigma = 1.5)[c(1, 2)], "                    1.5")
```

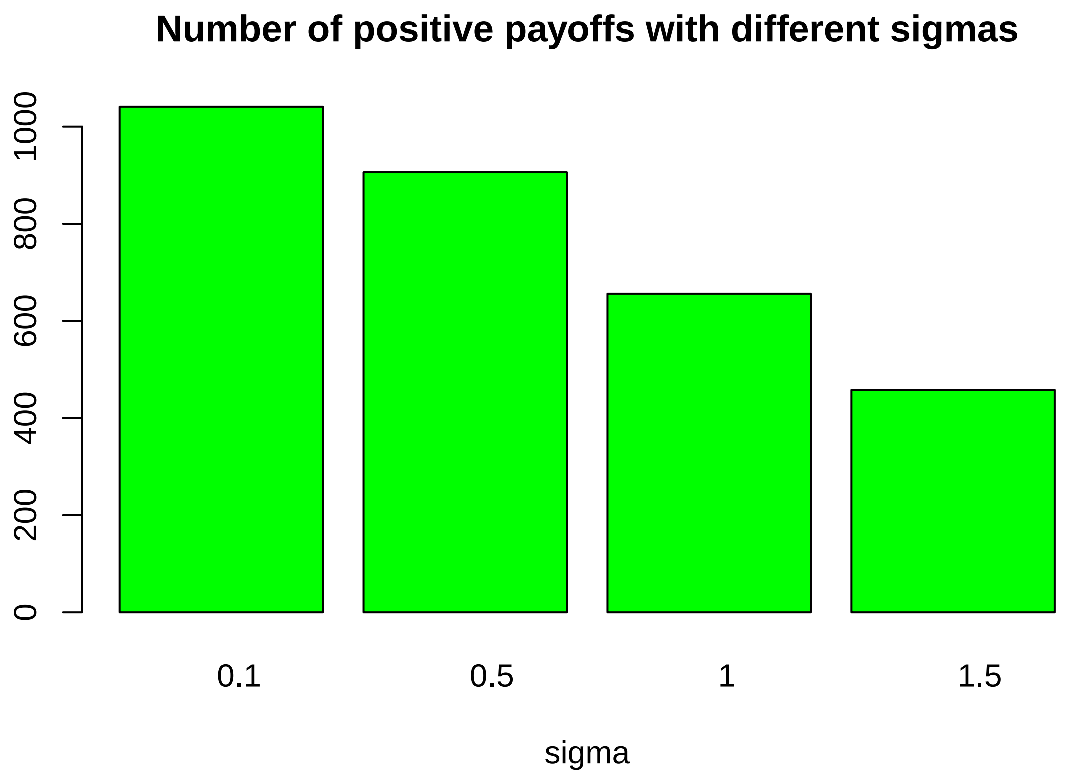
```
## We observed the average price to be: 27.26045
## and the standard deviation of:  131.7095
## in total:  463 of the payoffs were positive
```

```
sigmas <- c("0.5", "0.1", "1", "1.5")
barplot(c(a, b, c, d), col = rep(c("red", "blue"), 4), xlab="sigma",
        main = "Price & standard deviation with different sigmas")
legend("topleft", c("Price", "Standard deviation"), col=c("red", "blue"), lwd=3)
```



The results show that with the σ increasing, the price A_0 rises as well. That is because the price can never go below 0 and at the same time has a potentially unlimited upside. Therefore, with higher volatility more payoffs will be very high and will consequently raise the average. With a higher σ, the standard deviation of the payoffs also increases. The reason for that is that σ is the volatility of the underlying asset, S, and of course with a higher volatility the price will fluctuate more.

What is more is that with σ increasing, we observed the number of positive payoffs to decrease.



We explain this change as a result of the risk neutral probability, π decreasing when we increase σ ceteris paribus which becomes clear when we write down the following equation:

$$\pi = \frac{1-\bar{D}}{\bar{U}-\bar{D}} = \frac{1-e^{-\sigma/\sqrt{n}}}{e^{\sigma/\sqrt{n}}-e^{-\sigma/\sqrt{n}}} = \frac{e^{\sigma/\sqrt{n}}-1}{e^{2(\sigma/\sqrt{n})}-1} = \frac{e^{\sigma/\sqrt{n}}-1}{(e^{\sigma/\sqrt{n}}+1)(e^{\sigma/\sqrt{n}}-1)} = \frac{1}{e^{\sigma/\sqrt{n}}+1}$$