Post02: Monte Carlo Simulations and Basic Financial Tools in R

Jobe Cowen

November 21, 2017

Introduction

Hello!! And welcome to Jobe Cowen's Post02 on financial tools in R! As mentioned in my previous post, R contains lots of packages and means of accessing, analyzing, and working with financial data. This comes in handy if you are interested in markets and economics, and you'll soon begin to see that R, with the proper understanding and skills, contains a mighty arsenal of tools that can be applied to the business world.

To give you an overview, I will start off with a brief mention of what motivated me to explore this subject. Then, we'll look at a quick example of how to use a simple function in R to obtain basic stock market data. Next, we'll see how to use a Monte Carlo simulation to price a financial instrument called an option. We'll compare the results of this to the price we get from the famed Black-Scholes formula (one of the most important equations in finance). Finally, we are going to try something a little fancier. Something I experimented with in my previous post was the concept of bootstrapping, and I briefly mentioned the related technique of Monte Carlo simulations. Here, though, we'll take it to the next level and run a simulation in R that allows us to map out various future stock price paths.

Don't get overwhelmed! Even for those who are not business-saavy or experts of economics, there are very simple, but interesting, things we access in this area. We'll start off nice and easy.

Motivation

As mentioned in my Post01, I had an opportunity to study finance this past summer at the London School of Economics. Beyond just gaining an exposure to the fundamentals of asset pricing, most frequently by using the famous Black-Scholes formula, I learned a great deal about applications of statistics. In particular, I learned that Monte Carlo simulations can be very useful in financial modeling. Similar to the bootstrapping techniques that I used in my first post, Monte Carlo simulations use repeated random sampling to model the probability of different outcomes. The more simulations that are run, the more accurate the results become. This topic, and related concepts, have fascinated me ever since. Let's get on to the examples!

First Example: Basics of Retrieving Stock Market Data

A good place to begin is to simply use a tool in R which allows us to access market data. This is the natural place to begin, since no matter what you want to do regarding markets or economic analysis, you at least need to access the data! A great package for this exists called **quantmod**

First we will install and load the **quantmod** package. We'll also download the **OptionPricing** package which will be useful later in the post.

```r
# download package quantmod once
#install.packages("quantmod")
# download package OptionPricing once
#install.packages("OptionPricing")
library(quantmod)
library(OptionPricing)
```

Here let's use the **getSymbols()** function in the **quantmod** package to get basic stock data including daily volume of trades, daily price high, and closing price. We'll use the stock ticker symbol "NKE" to specify data on the footwear and apparel corporation Nike. We'll enter the appropriate function arguments to pull data from Yahoo Finance, beginning November 26th of this year.

Note: We also could have obtained the same data from different sources by changing the input to the *src* = argument of the getSymbols function (e.g. Google, using src = 'google').

```r
# retrieve data on Nike, Inc.
getSymbols("NKE", from="2017-11-26", src="yahoo")
```

```
## [1] "NKE"
```

```r
# display the data
head(NKE)
```

```
##            NKE.Open NKE.High NKE.Low NKE.Close NKE.Volume NKE.Adjusted
## 2017-11-27    59.20    59.70   59.17     59.63    7891600     59.43262
## 2017-11-28    58.76    59.60   58.54     59.58    8665300     59.38278
## 2017-11-29    59.73    60.63   59.73     60.36   12022800     60.16020
## 2017-11-30    60.26    61.21   60.20     60.42   12551200     60.22000
## 2017-12-01    60.42    60.43   59.24     59.88   10113500     59.88000
```

Swoosh!

Second Example: Option Pricing (Black-Scholes vs. Monte Carlo)

Now let's shift gears and take this idea of stocks and simple commodities in the marketplace a little further. As students of statistics, we can appreciate the fact that much of finance is concerned with the topic of uncertainty. Specifically, if an investor is uncertain about the future changes of price of an asset he or she owns or wishes to own, he or she could possibly lose lots of money, or at least lose the opportunity of financial gain - a frightening prospect in our capitalist society. This fear has led, over time, to the development of financial instruments to mitigate the risk that comes with this uncertainty. Some of the simplest ways to deal with these risks, at least when it comes to stock prices, is with instruments called options. We will get into the details of how to price these options, but we won't discuss all of the different flavors of options available to investors. To put it briefly, though, options are essentially side bets on the movement of the stock price, and we all know gambling and statistics have been closely related for ages. In fact, the simulations we are dealing with were even named for the famous gambling city of Europe, Monte Carlo!

Now we're going to create two custom functions. The first is a Monte Carlo simulation using a for loop. The second is a function to calculate the price of a put/call option according to the **Black-Scholes model**. Black-Scholes, though flawed, is one of the most famous formulas in finance. The Black-Scholes formula is as follows:

Price of a call option according to the Black-Scholes model

$c = S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2)$

Price of a call option according to the Black-Scholes model

$p = Ke^{-rT}\Phi(-d_2) - S_0\Phi(-d_1)$

$$d_1 = \frac{ln(S_0/K) + (r+\sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{ln(S_0/K) + (r-\sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

Once again, don't get overwhelmed! For our purposes here, it's just important to know that the Black-Scholes formula is an equation with several variables that is used to price financial instruments. Also, we will only deal here with European options which means that the option to buy (or sell) the underlying stock can only be exercised at the maturity date. Finally, Φ signifies the normal function. Here we go!

```r
## Examples of Monte Carlo simulations

## Pricing a call Option with MC

mc_call <- function(So, K, r, sigma, t, N)
{
  C=0
  for (i in 1:N)
  {
    #simulating the stock price at time T, no need for the whole path
    St = So * exp((r - (sigma ^ 2) / 2) * t + sigma * sqrt(t) * rnorm(1))

    #calculating the payoff for each path and summing them
    C = C + max(St - K, 0)

    # find prices by taking the expectation and discounting it to time 0
    price = exp(-r * t) * C / N
  }
  return(price)
}

BSprice <- function(pc, S, k, vol, d, r, t)
{
  #pc  put/call indicator call=1, put=-1
  #S   Stock price at 0
  #K   strike
  #vol volatility
  #d   dividend yield (we'll assume to be 0)
  #r   riskless rate
  #t   time to maturity (in number of years)

  d1 = (log(S / k) + t * (r - d + (vol ^ 2) / 2)) / (vol * sqrt(t))
  d2 = d1 - vol * sqrt(t)

  BSprice = pc * exp(-d * t) * S *
    pnorm(pc * d1) - pc * k * exp(-r * t) * pnorm(pc * d2)
  return(BSprice)
}
```

Now we'll compare the results we get from each method. Since Monte Carlo methods rely on repeated random sampling, we will set the seed each time we use the **mc_call()** function to ensure consistent results each time we call it.

```r
#compare the simulated price with the real one
BSprice(1,100,100,0.2,0,0.05,1)
```

```
## [1] 10.45058
```

```r
set.seed(123)
mc_call(100,100,0.05,0.2,1,100)
```

```
## [1] 10.70076
```

```
set.seed(123)
mc_call(100,100,0.05,0.2,1,10000)
```

```
## [1] 10.39192
```

Looks great! Notice how the Monte Carlo error decreases as we increase the number of simulations.

For good measure, we'll use a function from the R package *OptionPricing* to check our results. There is no need for us to worry now about the delta and gamma values, as they simply provide information on the option's price sensitivity to changes in the price of the underlying stock.

```
BS_EC(K=100, r = 0.05, sigma = 0.2, T = 1, S0 = 100)
```

```
##       price       delta       gamma
## 10.45058357  0.63683065  0.03184153
```

Look at that!! The option price matches our calculated value!

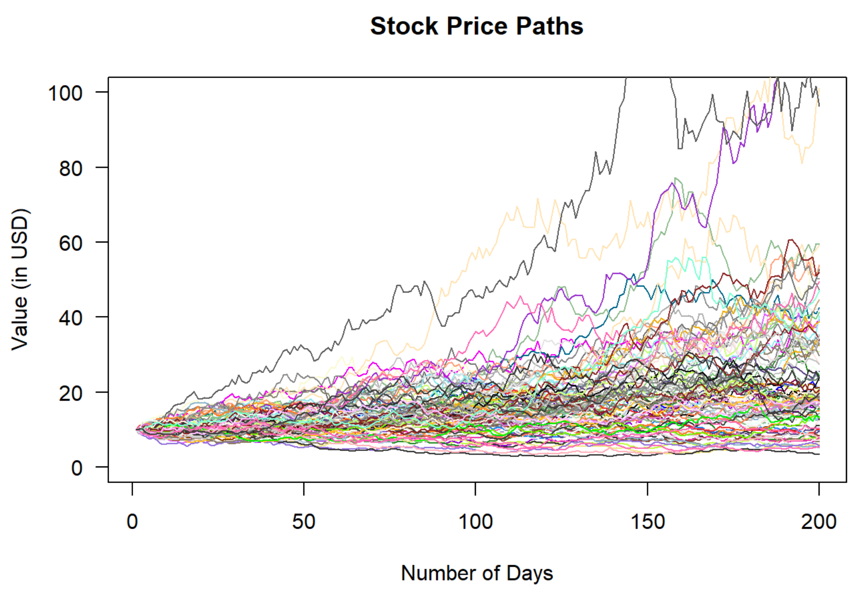
Third Example: Lognormal Price Paths with Monte Carlo

Finally, returning to stocks, a common technique is to use simulations and periodic daily returns in terms of natural logarithms. This will allow us to map out many possible changes in the price of the stock in the future. In order to do this we'll create two new functions. The first one will be used to calculate the lognormal price of a stock. Our second function will allow us to plot a given number of simulations of future stock prices of this stock using lines.

```
## Simulate stock price paths

## simulate lognormal stock price
# S     stock price
# mu    the drift
# sigma the volatility
# dt    the time interval
# N     the length of the time period (in number of days)
sim <- function(S, N, mu, sigma, dt)
{
  Log <- c()
  Log[1] <- S
  W <- rnorm(N) * sqrt(dt) #generate Brownian Motion
  for(i in 2:N)
  { #generate stock path - this is the BS equation solution
    Log[i] <- Log[i-1] * exp((mu - 0.5 * sigma ^ 2) * dt + sigma * W[i-1])
  }
  Log
}

# create function to simulate num_sim paths of the stock and plot them

# num_sim is the number of paths
sim_paths <- function(S, N, mu, sigma, dt, num_sim)
{
  #plotting the paths
  plot(sim(S, N, mu, sigma, dt), type = "l", col = "blue",
       main = "Stock Price Paths", ylim = c(0, S * 10),
       ylab = "Value (in USD)", xlab = "Number of Days", las = 1)

  for (i in 1:num_sim) {
    lines(sim(S, N, mu, sigma, dt),
    col = colors(1)[400 * round(runif(1), 2)])
    }
}
```

Great! Now that we have our functions, let's set our seed, enter some values for a stock currently priced at $10, and run 100 simulations.

```
# set the seed and plot some simulations
set.seed(123)
sim_paths(10,200,0.05,0.15,0.1,100)
```



As we might expect, there are not that many scenarios where the stock price increases or decreases dramatically. This is shown by the fact that even with 100 simulations run over a period of 200 days, few of the paths show huge change from the starting price. (For further treatment of Monte Carlo simulations and lognormal price simulations, the link in the reference section is an excellent resource).

Conclusion

As the above examples begin to demonstrate, R contains many useful and powerful tools for those interested in finance. In fact, just looking at the first link in the reference section below should give you an idea. It contains dozens of different R packages dealing with finance, risk management, and related topics. It would be quite useful to explore and play around with them. I hope you enjoyed the post, and I think that one thing to take away from it is that all of these financial tools in R are actually extensions of principles of statistics and R. In fact, Monte Carlo simulations can be created using basic statistical functions (like random sampling) and for loops. After observing the above examples, and exploring the related resources, you can begin to explore and conquer the world of finance. The tools are now in your hands! Happy exploring!!

References

- https://cran.r-project.org/web/views/Finance.html
- https://cran.r-project.org/web/packages/OptionPricing/OptionPricing.pdf
- https://cran.r-project.org/web/packages/quantmod/quantmod.pdf
- https://www.investopedia.com/terms/m/montecarlosimulation.asp

(for more concerning Monte Carlo simulations and lognormal price paths)

- https://www.investopedia.com/articles/optioninvestor/02/120602.asp
- https://www.investopedia.com/terms/b/blackscholes.asp
- https://software.intel.com/en-us/articles/black-scholes-merton-formula-on-intel-xeon-phi-coprocessor
- https://www.sharelatex.com/learn/List_of_Greek_letters_and_math_symbols