Exercise Boundary of an American Put Option

Michael Schopper, Dayten J. Sheffar
Department of Mathematics and Statistics

University of Victoria

Keywords: American Put Option, Exercise Boundary, Binomial Tree

12th April, 2021

Abstract

An *American Put Option* is a financial contract giving the owner the right (without obligation) to sell –or sell short– some amount of an underlying security at a price (the *strike price*) within a predetermined time frame. Puts are traded on bonds, stocks, currencies, commodities, futures and indexes. It is considered a bet *against* the underlying security, with profits coming by a decrease in the security price. It is often demonstrated that it is never ideal to exercise an *American Call Option* early. A natural question that then arises is whether it can be beneficial (and if so *when*) to exercise an American put option early. This *could* be found by hand in back propagating a binomial tree, but is inefficient. This paper looks at using vectorized computing in R to quickly find the exercise boundary between exercising and continuing to hold the option.

1 Introduction

A European option gives the contract holder the right (but not the obligation) to buy/sell a set number of underlying assets at a fixed maturity time t. The value of European call (betting for the underlying asset price to increase) and put (betting the price will decrease) options can easily be found through a closed-form formula of Nobel Prize fame, known as the Black and Scholes equation. Using the Black and Scholes equation, one can derive a formula to calculate the cost of a European call option

$$C(s,t,K,\sigma,r) = s\Phi(\omega) - Ke^{-rt}\Phi(\omega - \sigma\sqrt{t})$$

where s,t,K,σ,r are the initial share price, the exercise time, the strike price, the volatility and the risk-free interest respectively. Here, $\Phi(\cdot)$ represents the cumulative distribution function (cdf) of a normal random variable, $e^{\cdot}$ is the exponential function denoting a discount factor for the present value, and ω is represented by

$$\omega = \frac{\left(r + \frac{\sigma^2}{2}\right)t - log\frac{K}{s}}{\sigma\sqrt{t}}$$

By using the Put-Call parity formula ($C - P = s - Ke^{-rt}$), one can find an expression for the European put option quite easily

$$P(s,t,K,\sigma,r) = Ke^{-rt}\bar{\Phi}(\omega - \sigma\sqrt{t}) - s\bar{\Phi}(\omega)$$

where $\bar{\Phi}(\cdot) = 1 - \Phi(\cdot)$. However, this formula breaks down for computing American options which can be exercised early, but are otherwise similar to a European option. This requires us to turn to another useful tool, the binomial tree.

2 Methods

Below is a two-step binomial tree, to model a stock with initial share price S.

$$\begin{tikzcd}
 & & u^2S \\
 & uS \arrow[ur] \arrow[dr] & \\
S \arrow[ur] \arrow[dr] & & udS \\
 & dS \arrow[ur] \arrow[dr] & \\
 & & d^2S \\
n=0 & n=1 & n=2 \\
 & & \textit{expiry}
\end{tikzcd}$$

Avoiding some lengthy details, we simply provide that we can define the risk-neutral probability of the stock going up as

$$\mathbb{P}_u = \frac{e^{r\frac{t}{n}} - d}{u-d} = \frac{e^{r\frac{t}{n}} - e^{-\sigma\sqrt{\frac{t}{n}}}}{e^{\sigma\sqrt{\frac{t}{n}}} - e^{-\sigma\sqrt{\frac{t}{n}}}}$$

and the probability of going down $\mathbb{P}_d$ simply as $1-\mathbb{P}_u$. Here we've defined u and d as $e^{\pm\sigma\sqrt{\frac{t}{n}}}$, which are the increments of increase and decrease the share price may observe in a given step $0 < m \leq n$. And so at step 1, the share price is either $uS = Se^{\sigma\sqrt{\frac{t}{n}}}$ or $dS = Se^{-\sigma\sqrt{\frac{t}{n}}}$, where n is the number of steps to the maturity time t. The term $e^{r\frac{t}{n}}$ represents the discount factor, and may be addressed as β throughout this paper.

To give a brief but further explanation, a European put option has a maturity time t and cannot be exercised early. An American call however can be exercised before t, and if we want to consider continuous versus discrete time, we can examine the price of the option with some n steps between $t=0$ and the maturity time of the option, and arguably let n tend to infinity, however this isn't quite necessary for practical reasons. Evaluating the second to second value of the option would be computationally expensive and rather uninformative, but conceptually checks out.

We can find the *exercise* value of the put at any point by considering the difference between the strike price and the share price, denoted $(K-S_t)^+$ which equals zero if $S_t > K$ (the option is out of the money, it wouldn't be profitable to exercise) and simply the difference otherwise (the payout of the option that that time t). Hence, the payouts are calculable at any instant, observe the example below with an arbitrary strike price,

$$\begin{tikzcd}
 & & & \text{“}OTM\text{”} \\
 & & (K-u^2S)^+ = 0 & \\
 & K-uS \arrow[ur] \arrow[dr] & & K \\
K-S \arrow[ur] \arrow[dr] & & K-udS & \\
 & K-dS \arrow[ur] \arrow[dr] & & \text{“}ITM\text{”} \\
 & & K-d^2S & \\
n=0 & n=1 & n=2 & \\
 & & \textit{expiry} &
\end{tikzcd}$$

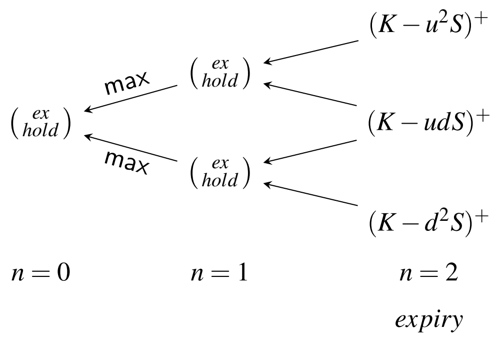
The value of the put is not simply the exercise value however, if we consider that we haven't yet priced in *holding* the put. To do so, we can first find all the share prices and all the exercise values of the put, for the whole binomial tree. Since there is no hold value of the put at expiry (it must be exercised), to find the hold value for any node at step $n-1$, we simply take the "up" node and the "down" node, and plug them into the following formula

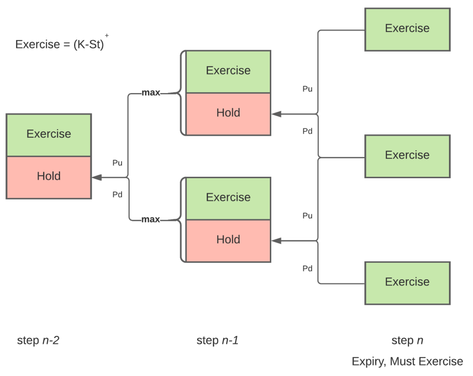
$$\text{Hold}_{new} = \beta\left[\mathbb{P}_u \text{Exercise}_{up} + \mathbb{P}_d \text{Exercise}_{down}\right]$$

however the situation is a little more complex for nodes $n-2$ to the origin. For these instances, suppose we have a node at $n-2$, we must take the maximum of the exercise and hold values at both the up and down nodes at step $n-1$, and plug those into a similar formula

$$\text{Hold}_{new} = \beta\left[\mathbb{P}_u \max(\text{Exercise}_{up}, \text{Hold}_{up}) + \mathbb{P}_d \max(\text{Exercise}_{down}, \text{Hold}_{down})\right]$$

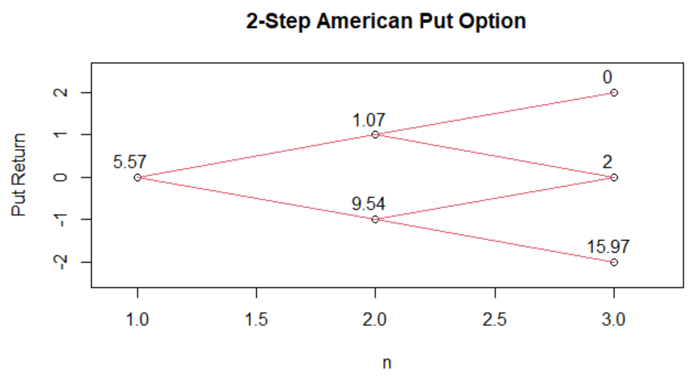
we provide some visualization below.





2-Step Backwards Propagation

Using the R library `fOptions`, we can generate a put-return binomial tree graph for an option with parameters[*] $s = 51$, $t = 0.5$, $K = 53$, $\sigma = 0.32$, $r = 0.05$ and $n = 2$



which computes the values at each node via back propagation with the two formula provided, depending if we are computing at step $m = n-1$ or $m \leq n-2$.

3 Computation

To begin with, we do not seek to store the put values in a 2-dimensional array since it would have n^2 cells, for larger n this would quickly require gigabytes of memory, which may not fit within your computer's RAM. In a 2-d array, With each step of the tree stored as a column, early columns will be sparse (many zeroes per column), and final columns will be nearly all populated, such as the table below. So one would have to store at least $\frac{n^2}{2}$ integer types (the zeroes) and approximately $\frac{n^2}{2}$ floating point numbers.

[*]The `fOptions` package for the binomial tree requires a parameter b representing the annualized cost-of-carry rate. We set this to zero for this example.

m=0	m=1	$\dots$	$m = n-1$	$m = n$
p_{11}	p_{12}	$\dots$	$p_{1,n-1}$	p_{1n}
0	p_{22}	$\dots$	$p_{2,n-1}$	p_{2n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
0	0	0	$p_{n-1,n-1}$	$p_{n-1,n}$
0	0	$\dots$	0	p_{nn}

In the table, the p_{ij} are the computed values of the put, and the top row represents the upward branch of the binomial tree, and the diagonal p_{ii} represents the downward branch. Since we are back propagating, we first calculate column n (which is the mandatory exercise value at expiry) via $(K-S_t)^+$, which is zero for any values out of the money (OTM, above K for a put). To calculate column $n-1$, we can take the expiry column twice, first as C_n^u, and then also C_n^d, which subset rows 1 to $n-1$, and 2 to n respectively. C_n^u represents the "up" nodes and C_n^d the "down".

Seeing as we can freely compute the share price S_t (in R; $s*u\wedge(0:k)*d\wedge(k:0)$ for some step $k \leq n$) and the exercise value $(K-S_t)^+$ (which we will denote by e_m for some step m), we suppose they are given. What we need to do then, is compute the hold values and store them in a new column h_{n-1}, which are found via

$$h_{n-1} = \beta\left(\mathbb{P}_u C_n^u + \mathbb{P}_d C_n^d\right).$$

Next, for any step $m \leq n-2$, we can find

$$h_m = \beta\left(\mathbb{P}_u \max(e_{m+1}^u, h_{m+1}^u) + \mathbb{P}_d \max(e_{m+1}^d, h_{m+1}^d)\right).$$

A code sample follows.

3.1 Vectorization

```r
x_over_n <- tibble(); speed_tbl <- tibble()      #initialize storage

s = 51; t = .5; K = 53; sig = .32; r = 0.05;     #set params
n = 2 #this reflects a two step binomial table

u = exp(sig*sqrt(t/n))
d = exp(-sig*sqrt(t/n))
P_up = (exp(r*t/n) - exp(-sig*sqrt(t/n)))/(exp(sig*sqrt(t/n)) - exp(-sig*sqrt(t/n)))
P_down = 1 - P_up
beta = exp((-r*t)/n)

x_i <- tibble(values = pmax(K - s*u^(0:n)*d^(n:0),0))
threshold_prices <- tibble(price = NULL, time = NULL,n = NULL)

#### INNER FOR LOOP ####
for(i in seq(n,1,-1)){
    tbl <- tibble(hold = as.numeric(unlist(beta *(P_up*x_i[2:(i+1),] + P_down*x_i[1:i,]))),
                  exercise = pmax(K - s*u^(0:(i-1))*d^((i-1):0),0),
                  share_price = s*u^(0:(i-1))*d^((i-1):0),
                  choice = ifelse(hold >= exercise, "hold", "exercise"))

    threshold_index <- match("hold",tbl$choice)
    threshold_price_i <- tibble(price = tbl$share_price[[threshold_index]], time = i,n=k)
    threshold_prices <- rbind(threshold_prices, threshold_price_i)
    x_i <- tibble(values = pmax(tbl$hold,tbl$exercise))
}
x_over_n <- rbind(x_over_n, threshold_prices)
```

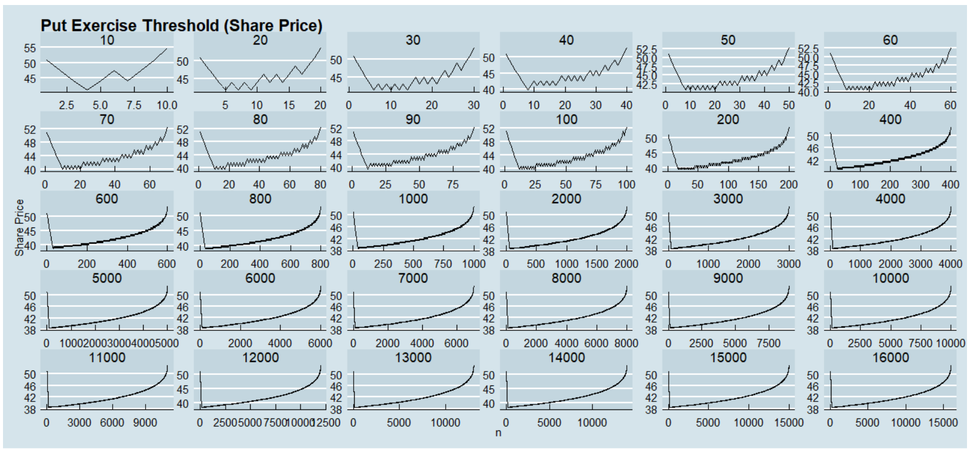
If the process wasn't vectorized, there would be another `for` loop. In order to generate an exercise boundary for multiple n, we can do so as follows

```
for(k in seq(1:100,1){
    n = k

    u = exp(sig*sqrt(t/n))
    d = exp(-sig*sqrt(t/n))
    P_up = (exp(r*t/n) - exp(-sig*sqrt(t/n)))/(exp(sig*sqrt(t/n)) - exp(-sig*sqrt(t/n)))
    P_down = 1 - P_up
    beta = exp((-r*t)/n)

    x_i <- tibble(values = pmax(K - s*u^(0:n)*d^(n:0),0))
    threshold_prices <- tibble(price = NULL, time = NULL,n = NULL)

    #### INSERT PREVIOUS INNER FOR LOOP HERE ####
    ####

    x_over_n <- rbind(x_over_n, threshold_prices)
}
```

Here we have simply included some of the header code from the first code sample *inside* a new for loop that loops over various n, and that's it, the process is vectorized and we can visualize our results.

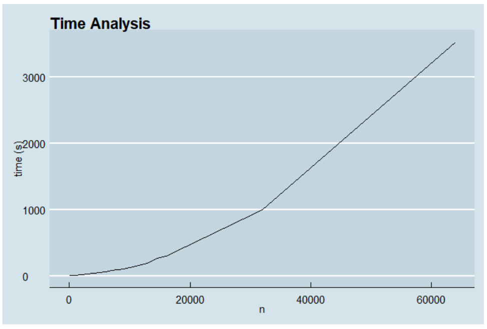
3.2 Visualization

For the following plot, the values of n that we used are indicated per facet.



4 Discussion

The plot of the exercise boundary above can be interpreted as: if you are on or below the line you can exercise the put; above the line, continue to hold. Moreover, as n increases, the line approaches a continuous curve. However, despite vectorization, to run the code is of order n^2, which we demonstrate here



where we have added $n = 32000, 64000$ for the visualization of the time complexity. And so, the main takeaway from vectorization is that we eliminate massive storage requirements, at most storing some columns that are less than or equal to n.

5 Conclusion

We have seen that it can be beneficial to exercise an American put option early, unlike an American call option. To do so, there is some consideration of how to compute the put value, depending on if you're examining the step $n-1$ (compute hold value straight from expiry exercise values) or $m \leq n-2$ (take the max of the up and down exercise and hold values at the step $m+1$). Vectorizing the code allows some possible but not obvious increased time efficiency, but mainly defeats having to store a table with n^2 elements, which can quickly grow into the gigabytes. Performing this computation allows one to find the exercise threshold for the put option, in order to invest intelligently versus blindly.