In this report, we are going to implement the CRR binomial model and verify its convergence to the Black-Scholes-Merton model. We will also be discussing about the effects of the maturity period on the initial stock price for which it first becomes optimal to early exercise the put/call option. Lastly, we will discuss the significance of dividend yield and its impact.

Options Pricing using Binomial Model

40.242-Mathematical Finance

Ong Jing Jie 1000722

Introduction

To start things off, let us establish our understanding of the binomial tree and its equations.

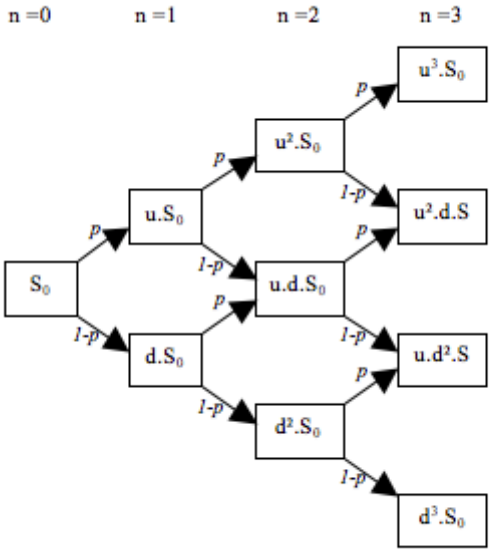


Fig. 1 A 3-step binomial tree

These are the equations we would need and $\Delta t = \frac{T}{n}$

$$p = \frac{e^{(r-q)\Delta t} - d}{u - d}$$

$$u = e^{\sigma\sqrt{\Delta t}}$$

$$d = \frac{1}{u}$$

We are given this function to implement: *Binomial (Option, K, T, S, σ, r, q, n, Exercise)*

Option: 'put' for put option and 'call' for call option

K: Strike price

T: Period to maturity

S: Initial Stock price

σ: Volatility

r: Continuous compounding risk free interest rate

q: Continuous dividend yield

n: Number of time steps

Exercise: 'A' for American options and 'E' for European options

Convergence of European Option to the Black-Scholes Model

Given the situation

Binomial ('call', 100, 1, 100, 0.2, 0.05, 0.04, *n, 'E')*

Using Excel and the Black-Scholes formula,

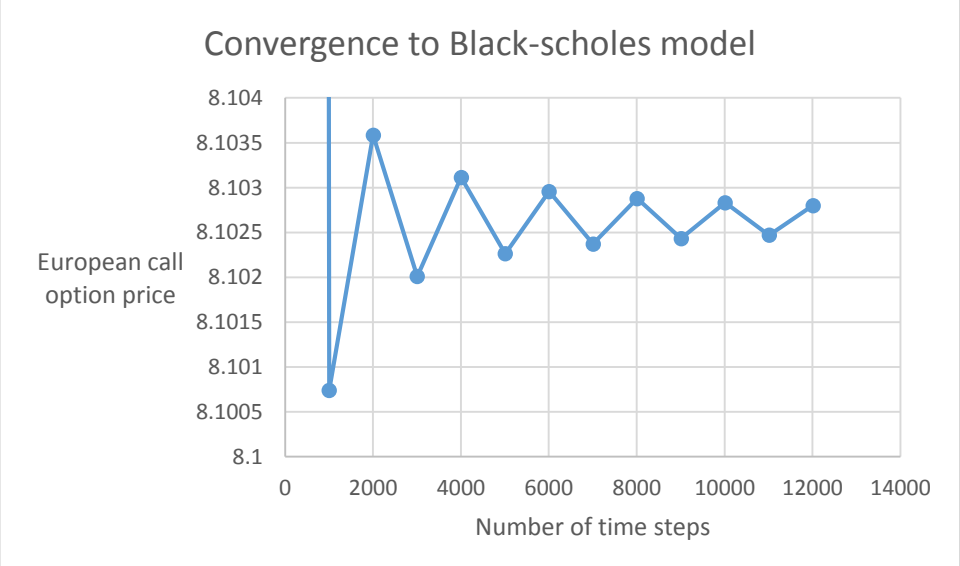
European call price:

$$c = S_0 e^{-qT} N(d_1) - K e^{-rT} N(d_2)$$

where

$$d_1 = \frac{\ln(\frac{S_0}{K}) + \left(r - q + \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}}, d_2 = d_1 - \sigma\sqrt{T}$$

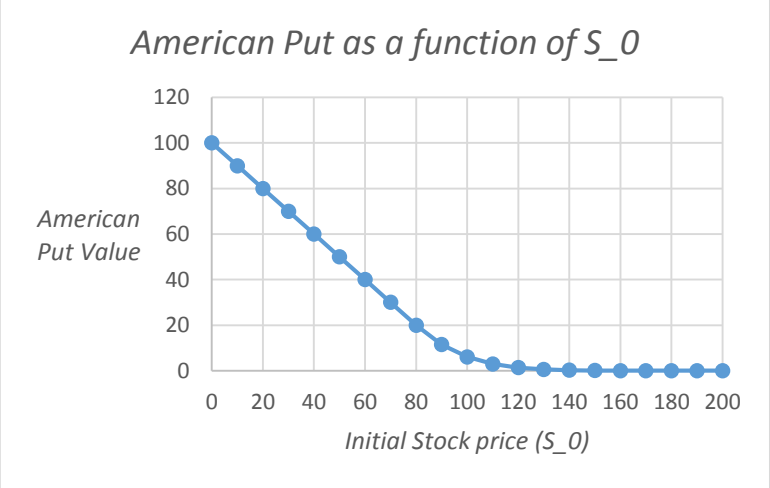
We have computed that the European call option = **8.102644**



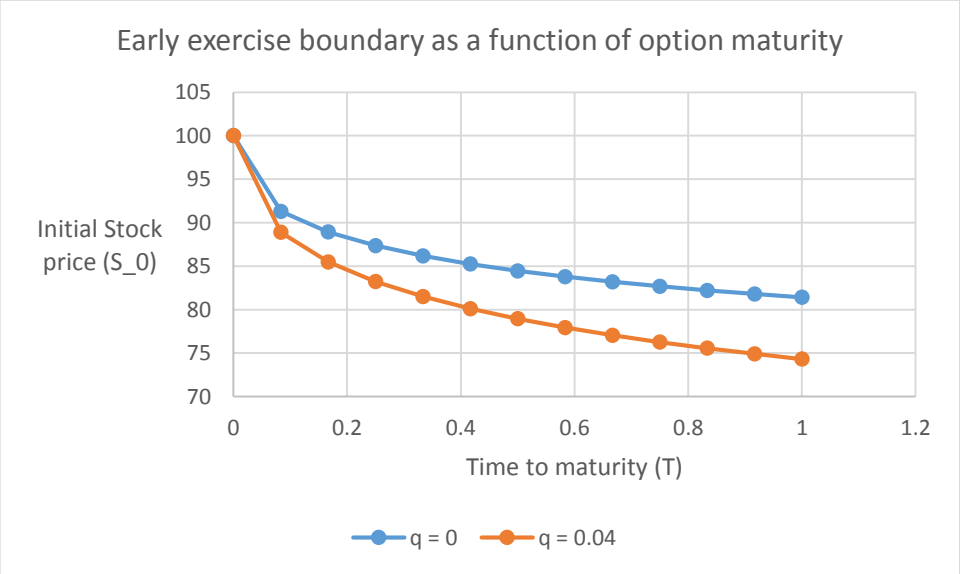
American puts and calls

Next, we consider an American put option and the underlying stock doesn't pay dividends

Binomial ('put', 100, 1, S, 0.2, 0.05, 0, n, 'A')



As the price goes to the strike price, the payoff (K - S_0) decreases and starts to go towards 0.



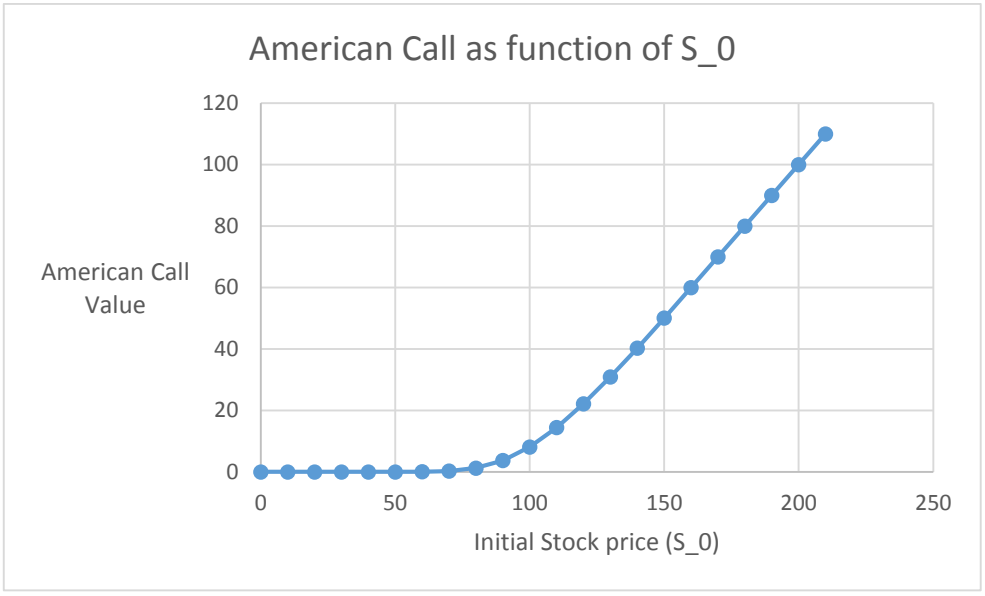
From the graph above, S*(12) = 81.409

When dividend yield is taken into account, it decreases the value of the stock and increase the likelihoodof (1 – *p*) and therefore the initial stock price is lower.

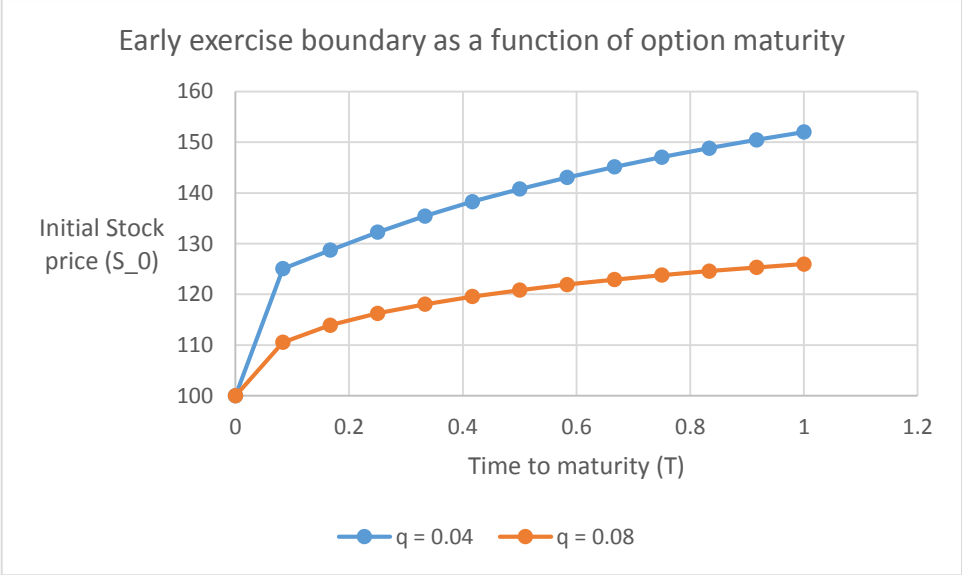
With dividend yield of 0.04, put option prices actually increases because its initial stock price is lower as comapred to no dividend yield. If we can exercise the put at a lower initial price, we will gain more and thus it is more expensive.

Next, we move on to American calls:

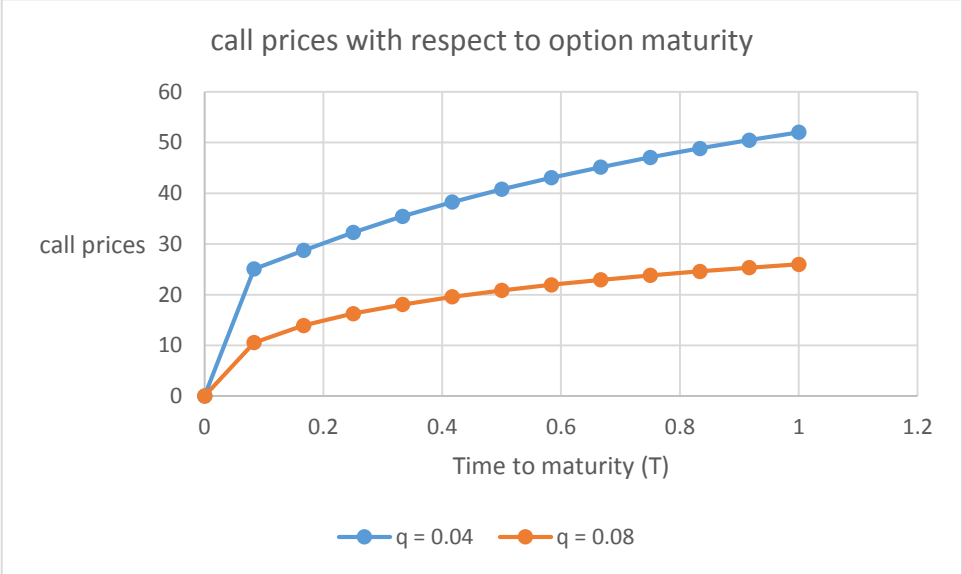
$$\underline{\textit{Binomial ('call', } 100,\ 1,\ S,\ 0.2,\ 0.05,\ 0.04,\ n,\ \textit{'A')}}$$



Before the strike price of 100, the payoff (S_0 – K) is negative and after 100, it started to increase steadily.



When q = 0.08, the value of p decreases and after the backward induction, its option value is low and therefore closer to its intrinsic value and thus has a lower initial stock price.



With a higher dividend yield, we can see from the previous graph that it also has a lower initial stock price. If I have a lower initial stock price, I would gain less from the early exercise and therefore it has a lower call price too.

Optimizing my code

1) Splitting my code into different functions rather than a single function

2) I have tried using interactive python (ipython) but the computational time only has a minor difference from python IDLE. However, using ANACONDA (spyder) is faster than using python IDLE.

To iilustrate the difference between using IDLE and spyder:

IDLE:

```
>>>
>>> ================================ RESTART ================================
>>>
S_0: 81.409 put price: 18.5959818169
Time Taken: 382.730000019
```

Spyder:

```
In [1]: runfile('C:/Users/ong/Desktop/Term 7/Mathematical
Finance/Project/financeproject.py', wdir='C:/Users/ong/Desktop/Term 7/Mathematical
Finance/Project')
S_0: 81.409 put price: 18.5959818169
Time Taken: 370.241000175

In [2]:
```

Therefore, to reduce computational time, I run all my codes in spyder.

3) American puts and calls

When plotting the put value as a function of the initial stock price S, I tried experimenting with the number of time steps to get the put value converge to a value up to 3 decimal place. Finally, I settled with 2050 because there is not much difference after the third decimal place even if I increase the number of time steps.

4) I have shorten this code which took 370.298999786 seconds to compute

```
# To find S*
start = time.time()

i = 100
while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
    i -= 10


i = i + 10
while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
    i -= 1


i = i + 1
while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
    i -= 0.1

i = i + 0.1
while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
    i -= 0.01

i = i + 0.01
while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
    i -= 0.001
print i

end = time.time()
print "Time Taken:", (end - start)
```

to this code which took 324.592999935 seconds to compute

```
i = 100
for m in [10, 1, 0.1, 0.01, 0.001]:
    while abs(100 - i - binomialtree('put', 100, 1, i, 0.2, 0.05, 0, 2050, 'A')) >= 0.005:
        i -= m
    if m != 0.001:
        i = i + m
    else:
        print i
```

The End