Using Python to Find the Replication Error if Delta Hedge a Trinomial Tree Option Over Many Short Periods

Nuoxing Shang[1], Yujia Liu[2], Zewei Lin[3*]

[1] *Jiangsu Tianyi High School, No.18 Erquan Road Xishan District Wuxing City Jiangsu Province, 214101*
[2] *Jiangsu Tianyi High School, No.18 Erquan Road Xishan District Wuxing City Jiangsu Province, 214101*
[3] *School of Economics and Finance, Queen Mary University of London, London E1 4NS, United Kingdom*
**Corresponding author. Email: eltham_nj@hotmail.com*

ABSTRACT

In this paper, the researcher creates a model for trinomial tree option pricing with multiple time periods by using Monte-Carlo estimation and Python. However, the delta hedging strategy needs to be improved to minimize the replication error.

Keywords: *Python, Delta hedgem, replication error, Monte-Carlo estimation*

1.INTRODUCTION

What is the problem that always exists with any type of research or experiment is conducted? The answer would be the existence of different errors [1]. In the case of option pricing, the replication error is being considered when Delta hedge is use over periods. Particularly, how large the replication error is. A more precise calculation can be done if we can accurately calculate the existed error. Thus, a more reliable option price can be provided to the market. This paper will explain how we can get the size of the replication error that would occur when we use delta hedge over many short periods by considering a trinomial model.

2. METHODS

1.Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables [2]. It is a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. In our research, we used code to practice Monte Carlo method to price an option in the case of three different possible stock price change, or trinomial tree model. The definition of each variable is displayed in the following annotation of code.

2.We firstly attempted to model a trinomial tree with one period. The process chronologically included defining function to calculate the payoff of an option, defining variables, solving simultaneous equations for ‘pd’ and ‘pm’, simulating the stock price under three situations, and obtaining Y1. Then we reedited the code by increasing the possible range of stock values and corresponding probability to improve the method to price trinomial tree option with multiple time periods.

3.We also defined function to calculate the payoff of the investment portfolio, which equals number of stock *stock price + number of bond* annual interest rate. Using loops and formulas, we replicated the process of obtaining a new Ns and bringing it into portfolio to gain a new value of payoff.

4.The differences between two functions (trinomial tree model & payoff of investment portfolio is named as hedging error/replication error. We brought several groups of exact variables to the functions and repeated process as representatives.

3.TRINOMIAL TREE MODEL WITH WITH A PERIOD OF ONE

```
In [1]:import sympy
import numpy as np

    # Define Functions
    def payoff_call(S, K):
    return np.maximum(S-K, 0)
```

```
def probability_of_d_and_u(R, d, u, m, pm):
    pd = (R - u - pm * (m - u)) / (d - u)
    pu = (R - d - pm * (m - d)) / (u - d)
return pd,pu

# Define Variables
S0 = 100
R = 1.07
u = 2
d = 0.5
m = 1   # m = d * u
pm = 0  # fix the value of probability if the situatuion 'm' occurs
n_simulation = 1_000_000
strike = 80
pd,pu=probability_of_d_and_u(R,d,u,m,pm)
print('pd = ',pd,' pu = ', pu,'pm = ', pm)   # Values for 'pd' and 'pu'
#  simulate three situations (u,m,d), and obtain Y1 (shock at time 1)
Y1 = np.random.choice(
[d,m,u],size=n_simulation, replace=True, p=[pd,pm,pu])
S1 = Y1 * S0
f = payoff_call(S1, strike)
print('S1.mean()/R = ',S1.mean() / R)
print('A dummy check to see whether the calcualtions are correct,the value is close to S0, so it is correct.')
print(f'The Monte-Carlo estimate of the price equals {f.mean() / R}.')
    pd =  0.62  pu =  0.38000000000000006 pm =  0
    S1.mean()/R =  99.9018691588785

A dummy check to see whether the calcualtions are correct, the value is close to S0, so it is correct.
The Monte-Carlo estimate of the price equals 42.538317757009345.
```

3.1.Trinomial tree model with a period of 2

now we are simulating situations in the period 2 where the stock values can be S0∗u2S0∗u2, S0∗uS0∗u, S0S0, S0∗dS0∗d or S0∗d2S0∗d2 with probability of pu2,pm∗pu+pu∗pm,pm∗pm+pu∗pd+pd∗pu,pm∗pd+pd∗pm,pd2pu2,pm∗pu+pu∗pm,pm∗pm+pu∗pd+pd∗pu,pm∗pd+pd∗pm,pd2

```
In [2]:# Define Functions
        def payoff_call_2(S, K):
        return np.maximum(S-K, 0)
    # Define Variables
    n_simulation = 1_000_000
    p1= pd * pd
    p2= 2 * pm * pd
    p3= pm * pm + 2 * pu * pd
    p4= 2 * pu * pm
    p5= pu * pu
    Y2 = np.random.choice(
[d * d,d,m,u,u * u],
 size=n_simulation,
 replace=True,
 p=[p1,p2,p3,p4,p5]
)
    S2 = Y2 * S1
    f = payoff_call_2(S2, strike)

    print('S2.mean() / R**2 = ', S2.mean() / R**2)
    print(f'The Monte-Carlo estimate of the price equals {f.mean() / R}.')
    S2.mean() / R**2 =  106.90819067167438

    The Monte-Carlo estimate of the price equals 66.96706542056074.
```

3.2.Ploting the error

```
In [3]:   import matplotlib.pyplot as plt
    # Define Functions
  rolling_average = (np.cumsum(f)/R) / (np.arange(n_simulation) + 1)

  plt.plot((np.arange(n_simulation) + 1), rolling_average);
  plt.xlabel('Number of samples')
  plt.ylabel('Approximated call value')
  plt.title('Monte-Carlo approximation');

  print('Assuming the option payoff is equal to the stock price at maturity, the value returned should be close to S0 ')
  print('S2.mean() / R**2 = ', S2.mean() / R**2)
  Assuming the option payoff is equal to the stock price at maturity, the value returned should be close to S0
  S2.mean() / R**2 =  106.90819067167438
```

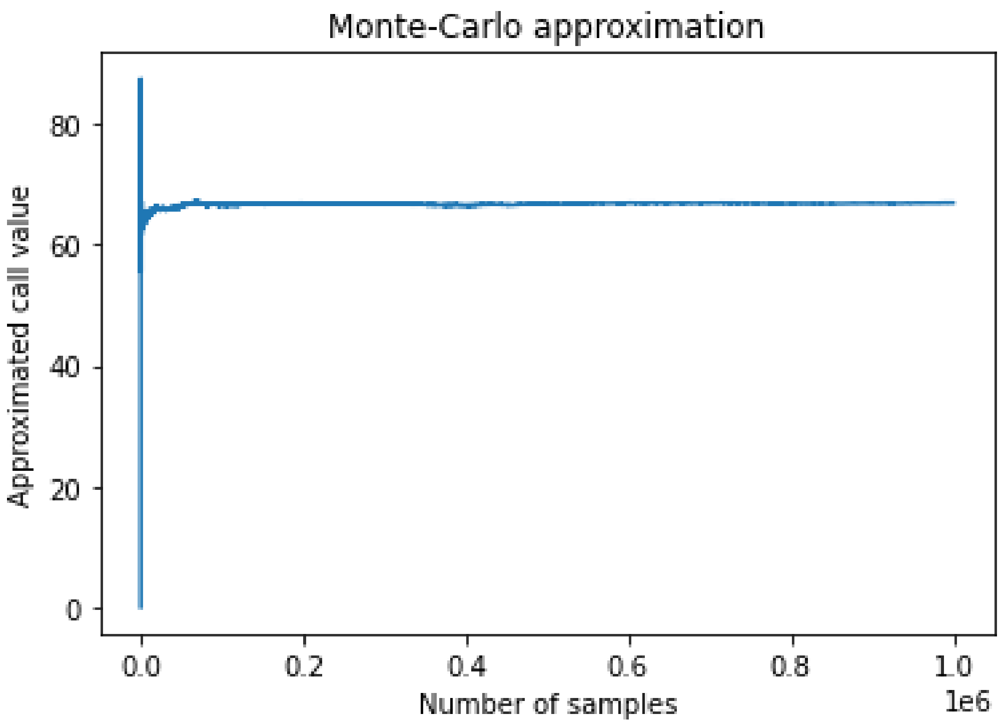

Figure 1 Spread of the values of call options simulated by Monte-Carlo Method in periods of two.

3.3.An improved method to price trinomial tree option with multiple time periods

```
In [4]:     # Define Functions
            def stock(Y):
            X = np.zeros(len(Y));
            for i in range(0,len(Y)):
                X[i] = 1;
            for j in range(0,len(Y[0])):
            X[i]*=(Y[i])[j];
            return X;

            def payoff_call_t(S, K):
            return np.maximum(S-K, 0)


            # Define Variables
T = 2
            n_simulation = 1_000_000
            Y =
np.random.choice([d,m,u],[n_simulation,T],
replace=True, p=[pd,pm,pu])
            St = stock(Y) * S0
            f = payoff_call_t(St, strike)
            option_price = f.mean() / R**T


            print(f'The Monte-Carlo estimate
of the price equals {option_price}.')
            print('Those are the shocks occur
in each time periods: ')
            print(Y)
            print('By multipling all shocks in
each simulation, the following values are obatined: ')
            print(stock(Y))
            print('The stock price at the end
of maturity in each simulations: ')
            print ('St = ',St)
```

The Monte-Carlo estimate of the price equals 48.53913878941392.

Those are the shocks occur in each time periods:

```
[[0.5 2. ]
[0.5 0.5]
[2.  0.5]
...
[0.5 2. ]
[2.  2. ]
[0.5 0.5]]
```

By multipling all shocks in each simulation, the following values are obatined:

```
[1.   0.25 1.   ... 1.   4.   0.25]
```

The stock price at the end of maturity in each simulations:

```
St =  [100.  25. 100. ... 100. 400.  25.]
```

3.4.Ploting the error

```
In [5]:     import matplotlib.pyplot as plt

            # Define Functions
            St.mean() / R**T # dummy check
            rolling_average =
(np.cumsum(f)/R) / (np.arange(n_simulation) + 1)
```

plt.plot((np.arange(n_simulation) + 1), rolling_average);
plt.xlabel('Number of samples')
plt.ylabel('Approximated call value')
plt.title('Monte-Carlo approximation');

print('Assuming the option payoff is equal to the stock price at maturity, the value returned should be close to S0 ')
print('St.mean() / R**T ', St.mean() / R**T)

Assuming the option payoff is equal to the stock price at maturity, the value returned should be close to S0
St.mean() / R**T
99.93685037994584

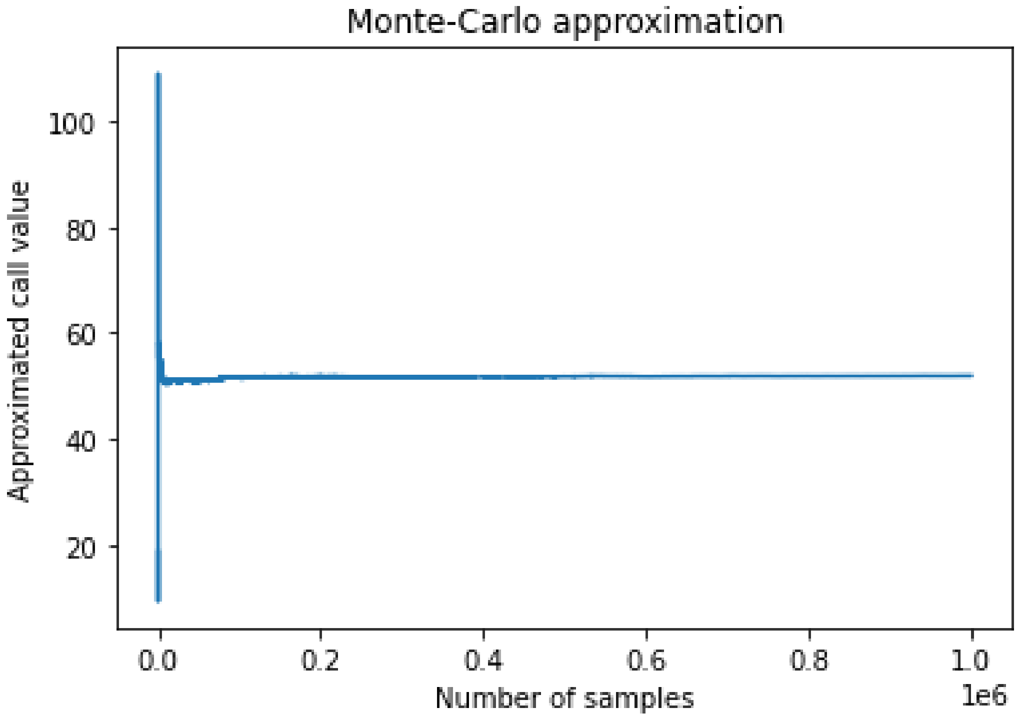

Figure 2 Spread of the values of call options simulated by Monte-Carlo Method in multiple periods. With a flatter line, showing that the values gained are more accurate than the previous method.

3.5.Replicating the call option using bonds and shares

In [6]:*# Define Variables*

```
N_sim = 1000
Samples = []
for i in range (N_sim):
    X_j = option_price # Assume the initial portfolio value is equal to the option price
    Sj = S0    # The initial stock price is S0
    for t in range (T):
        Y_j = np.random.choice([d,m,u], replace=True, p=[pd,pm,pu])  # Shock at time J
        NS_j = (np.maximum(Sj * u - strike, 0) - np.maximum(Sj * d - strike, 0)) / (Sj * (u - d))  # Number of stock invested in time J
        Sj *= Y_j    # Stock price at time J
        X_j = R * X_j + NS_j * Sj * (Y_j - R)   # Portfolio value at time J
        Replication_error = X_j - np.maximum(Sj - strike, 0)

    Samples.append(Replication_error)
np.mean(Samples)
plt.hist(Samples, bins = 100,density=True);
print('X_j = ',X_j)
print('oprtion_price = ',option_price)
print('Y_j = ',Y_j)
print('NS_j = ',NS_j)
plt.xlabel('Error')
plt.ylabel('Frequency')
plt.title('Replication Error');
```

X_j = 27.376459999999998
oprtion_price = 48.53913878941392
Y_j = 0.5
NS_j = 0.26666666666666666

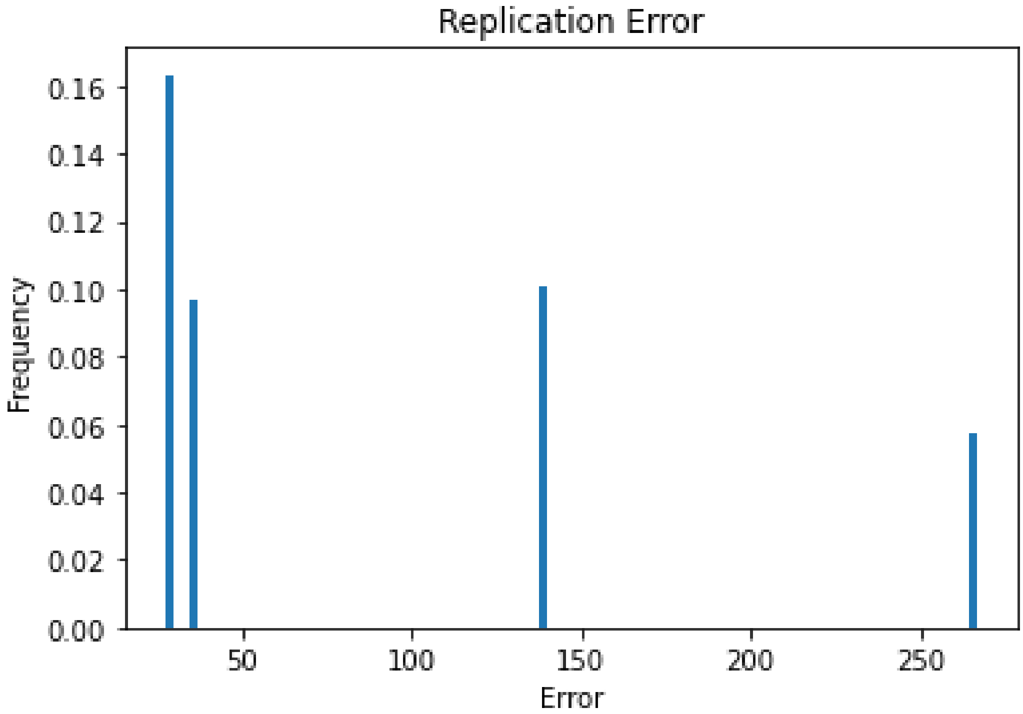

Figure 3 Frequency of replication errors.

In [7]: Replication_error
27.376459999999998

4.DISCUSSION

For the trinomial tree option pricing, the price values obtained are checked by assuming the option payoff is equal to the stock price at maturity, then divide it by R ** T, if a value close to the initial stock price is returned, then it proves the simulation of stock price in each three probability conditions in different time periods is correct [3].

In the simulation process of trinomial tree option pricing with two periods, the value is not close enough to S0; this is casued by model error, the mathmatical model in two-period-price calcuation is not fully presented with codes, and the method of Monte-Carlo simulation is not used.

In the second attempt, again with a two period model, the changes in stock prices and shocks at each periods is not only better presented using matrix but also more accurate by using the Monte-Carlo simulation, and the option price calcualted is well improved compared to the firse attempmt, with a dummy check result very close to the initial stock price. By plotting the errors in the two attempts, it is clear that the second attempt has less error [4]. However, the error can be further reduced by increasing the number of simulations, in our model, only 1,000,000 simulations are made due to limitions of computer, our results can be improved if number of simulations is more.

In our final step of trying to replicate the option by finding a delta hedge stragety, it is hard to find a stragety to replicate an option with three probabilities [5]. We have used the following stragety, Number of stock invested = (payoff when stock goes up - payoff when stock goes down) / (Stock price * (u - d)). By plotting the error, this stragety is clearly not suitable for a trinomial tree option pricing. The replication would be better if an improved stragety is found.

5.CONCLUSION

We have successfully built up a model for trinomial tree option pricing with multiple time periods by using Monte-Carlo estimation and Python. However, the delta hedging strategy needs to be improved to minimize the replication error.

REFERENCES

[1]. Cunningham P. The Apparition at Medjugorje: A Transpersonal Perspective - Part I[J]. Journal of Transpersonal Psychology, 2011, 43.

[2]. Madouasse A, Huxley J N, Denborne B H P V, et al. Presenting uncertainty and variability to the decision maker: A computer program that uses Monte Carlo simulations to improve the management of the dry period.

[3]. Tomá Tich. The convergence of binomial and trinomial option pricing models.

[4]. Collin-Dufresne P, Daniel K D, Moallemi C C, et al.

Dynamic Asset Allocation with Predictable Returns and Transaction Costs[J]. Ssrn Electronic Journal, 2015.

[5]. Arnold W C, Chess D M, Morar J F, et al. Method and Apparatus for Determination of the Non-Replicative Behavior of a Malicious Program[J]. 2008.