ISSN: 2692-7608 DOI: 10.6981/FEM.202012_1(12).0018

Strategies and Replication Errors of Option Pricing Using Trinomial Model

Fanfei Chen[1], Haoran Deng[2], Mingheng Guo[3]

[1]University College London, London WC1E 6BT, United Kingdom;

[2]Wenzhou-Kean University, Wenzhou, Zhejiang 325060, China;

[3]London School of Economics, London WC2A 2AE, United Kingdom.

Abstract

Under the framework of the binomial model towards option pricing, this work aims to find a way to price options in the trinomial model, with detailed procedures using Python. Based on the limitation of the trinomial model, investors can't price an option perfectly by replicating portfolio. And this research constructs three different hedging strategies by adjusting the value of Ns (number of underlying assets). Monte-Carlo simulation was used in this report to compare the payoff errors and standard deviations between our strategies and the initial binomial model. The result shows all three methods produce similar tiny errors thus the authors think in the giving situation there's no obvious difference between these three models. The previous researches were focusing on figuring out the delta value by creating non-arbitrage models, to find the neutral probability, in the trinomial tree; this research is trying to modify the Ns model according to the giving nature probability in the trinomial tree.

Keywords

Binominal Tree; Trinomial Tree; Delta Hedge; Option Pricing; Python.

1. Introduction

The Trinomial model is an option pricing model, modified from the Binomial model. In contrast to the Binomial model, an option in the Trinomial model cannot be priced by a replicating portfolio. This report aims to find a good strategy with only two primary assets, to calculate the replication error, and to analyze the result. The 'Delta hedge' from the title is to find the corresponding Ns (number of underlying assets) of the replicating portfolios at each short period.

In some ideal situations, the initial wealth (X_0) of the replicating portfolio should be equal to the option value. This study is going to find the error by comparing the portfolio with the corresponding payoff of the option. In order to find the error, different hedging strategies (i.e., different Ns) and pricing methods were set to find X_0. And this work suggests and discusses three different strategies including constructing Ns from a binomial model, trying to include all three nodes of the trinomial model, and taking an average of Ns as well as the combined method. Using these three methods with the X_0 from an appropriately calibrated (see below) Binomial model, constructing and comparing the errors.

In order to improve the overall error, a sophisticated calibration was used to determine the hedging strategy and the initial capital X_0. 'Calibrating' here means to choose u_B such that the variance in the Binomial model is the same as in the Trinomial model. Indeed if the variances are matched, option prices are more accurate.

2. Setting the parameters

```
T = 1000   # number of periods
```

```
import math
u = math.exp(1.5/math.sqrt(T))
R = 1
S0 = 1
m = R
d=m**2/u
# We need to make sure u*d=m*m in order to ensure that the tree is recombining.
```

3. Preparatory steps

3.1. Import the packages that we need

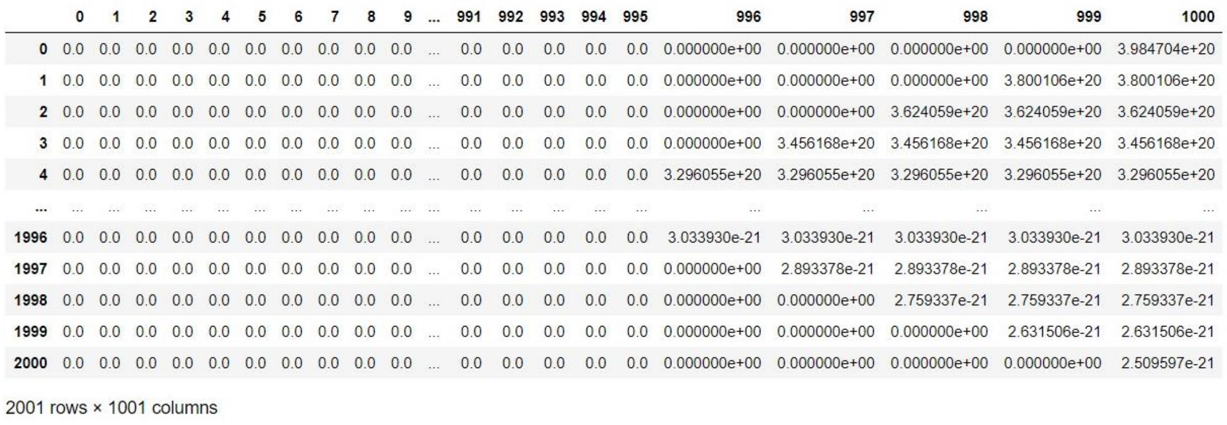
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import sympy as sp
import seaborn as sns
sns.set()
```

3.2. Present the metrics into a dataframe with their names

```
def variablename(name):
return [tpl[0] for tpl in filter(lambda x: name is x[1], globals().items())]
def print_matrix(x):
   x_df = pd.DataFrame(x)
   display(x_df, variablename(x))
```

3.3. Implementation of the trinomial stock price tree

```
stock = np.zeros([2*T + 1, T + 1])
for t in range(T + 1):     # t represents the time periods
for i in range(t + 1):
      stock[-i-1-T,t] = S0 * (m ** (t - i)) * (u ** i)
# compute the stock price that goes up
      stock[i+T, t] = S0 * (m ** (t - i)) * (d ** i)
# compute the stock price that goes down
print_matrix(stock)
```



	0	1	2	3	4	5	6	7	8	9	...	991	992	993	994	995	996	997	998	999	1000
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	3.984704e+20
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00	3.800106e+20	3.800106e+20
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	3.624059e+20	3.624059e+20	3.624059e+20
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	3.456168e+20	3.456168e+20	3.456168e+20	3.456168e+20
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	3.296055e+20	3.296055e+20	3.296055e+20	3.296055e+20	3.296055e+20
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1996	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	3.033930e-21	3.033930e-21	3.033930e-21	3.033930e-21	3.033930e-21
1997	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	2.893378e-21	2.893378e-21	2.893378e-21	2.893378e-21
1998	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	2.759337e-21	2.759337e-21	2.759337e-21
1999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00	2.631506e-21	2.631506e-21
2000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.509597e-21

2001 rows × 1001 columns

Table 1. Stork price after 1000 short periods

Volume 1 Issue 12, 2020
DOI: 10.6981/FEM.202012_1(12).0018

3.4. Auxiliary: Binomial option pricer

The following function prices the option in the trinomial model as if it were a Binomial model without the m node.

```
def option_Binomial(payoff, p, u):
price_Bin = np.zeros([2*T + 1, T + 1])
  price_Bin[:, T] = payoff
  u_b = calibrate_ub(p, u)
#using the variance from the Trinomial model to find suitable ub and db
d_b = m**2/u_b
  q = (R-d_b)/(u_b-d_b)[1]
 # probability of going up in calibrated Binomial model
  for t in range(T - 1, -1, -1):
    for i in range(0, 2*t + 1):
      price_Bin[i+(T-t), t] = (
        1 / R*(q*price_Bin[i+(T-t) - 1, t+ ] + (1-q) * price_Bin[i+(T-t) + 1, t+1]))[2]
  return price_Bin
```

3.5. To find the initial capital

In order to find X_0, the following function finds the u in the situation where binomial variance = trinomial variance.

Here we assume, by using the variance from the Trinomial model, the Binomial method would provide a more accurate price.[3]

```
def calibrate_ub(p, u):
var = (p[0]/sum(p))*u**2 + (p[1]/sum(p))*m**2 + (p[2]/sum(p))*d**2 - ((p[0]/sum(p))*u + (p[1]/sum(p))*m + (p[2]/sum(p))*d)**2 #variance of trinomial model
 x = sp.Symbol('x')
  f = (R*x**2)/(x+1) + R/(x*(x+1)) - R**2 - var      #Var.Tri(u) = Var.Bin(x)
  x = sp.solve(f)
  ub = max(x)
  return ub
```

Trinomial tree variance :

$$Var(Y) = E[Y^2] - (E[Y])^2 = p_u u^2 + p_m m^2 + p_d d^2 - (p_u u + p_m m + p_d d)^2$$

Binomial tree variance:

$$Var(Y) = E[Y^2] - (E[Y])^2 = q_u u^2 + (1 - q_u) d^2 - (q_u u + (1 - q_u) d)^2$$
$$= \frac{R - 1/u}{u - 1/u} u^2 + \frac{u - R}{u - 1/u} 1/u^2 - R^2 = \frac{Ru^2}{u + 1} + \frac{R/u}{u + 1} - R^2$$

3.6. Function to find the error via Monte-Carlo [4]

```
def find_error(X0, ns, payoff, N_of_sims, p):
   #randomly generate 1,0,-1 according to the probability we set above
c = np.random.multinomial(1, p, [N_of_sims, T])
  aux = [1, 0, -1]
  Y_raw = np.dot(c, aux)
  Y = ((u-d)/2 * (Y_raw + 1) + d - m) * Y_raw**2 + m
#turns 0 to m, -1 to d, and 1 to u
```

```
π = ns * stock             #Now we have N random draws of the stock price
  error_mc = np.zeros(N_of_sims)
  stock_mc = np.zeros(N_of_sims)

  for i in range(0, N_of_sims):
    X = X0
    idx = T                #where we are currently on the grid
    for t in range(0, T):
      X = R*X + π[idx, t] * (Y[i, t] - R) # Self-financing[5,6]
      idx -= int(Y_raw[i, t])
    stock_mc[i] = math.log(stock[idx, t])
    error_mc[i] = X - payoff[idx]
#Compare X(Wealth) with corresponding payoff of the option

  return [error_mc, stock_mc]
```

$$Self-financing: X_j = N_j^B B_j + N_j^S S_j$$

$$X_{j+1} = N_j^B B_{j+1} + N_j^S S_{j+1}$$

$$= N_j^B R B_j + N_j^S S_j \frac{S_{j+1}}{S_j}$$

$$= R(N_j^B B_j + N_j^S S_j) + N_j^S S_j (\frac{S_{j+1}}{S_j} - R)$$

$$= R X_j + \pi_j (Y_{j+1} - R)$$

3.7. Strategy to find Delta (ns)

3.7.1. Strategy 1:

construct Ns by forgetting about the middle branch using the Binomial strategy: $N_s = \frac{V_u - V_d}{S_u - S_d}$

```
def ns_binomial(payoff, p, u):
o_price = option_Binomial(payoff, p, u)

  ns = np.zeros([2*T + 1, T + 1])

  for t in range(T - 1, -1, -1):
    for i in range(0, 2*t + 1):
      Vu = o_price[i+(T-t)-1, t+1]
       #option value when stock price goes up
      Vd = o_price[i+(T-t)+1, t+1]
      #option value when stock price goes down

      Su = stock[i+(T-t)-1, t+1]
      Sd = stock[i+(T-t)+1, t+1]

      ns[i+(T-t), t] = (Vu-Vd)/(Su - Sd)
    return ns
```

3.7.2. Strategy 2:

construct Ns by taking the average of Ns , then we are able include all three notes to get Ns rather than only using u and d: $N_s = (\frac{V_u - V_d}{S_u - S_d} + \frac{V_u - V_m}{S_u - S_m} + \frac{V_m - V_d}{S_m - S_d})/3$

```
def ns_average(payoff, p, u):
o_price = option_Binomial(payoff, p, u)

    ns = np.zeros([2*T + 1, T + 1])

    for t in range(T - 1, -1, -1):
      for i in range(0, 2*t + 1):
        Vu = o_price[i+(T-t)-1, t+1]
        Vd = o_price[i+(T-t)+1, t+1]
        Vm = o_price[i+(T-t),t+1]          #option value when stock price stays constant
        Su = stock[i+(T-t)-1, t+1]
        Sd = stock[i+(T-t)+1, t+1]
        Sm = stock[i+(T-t),t+1]

        ns[i+(T-t), t] = ((Vu-Vd)/(Su - Sd)+(Vu-Vm)/(Su-Sm)+(Vm-Vd)/(Sm-Sd))/3

    return ns
```

3.7.3. Strategy 3:

In Trinomial model, it is not possible to exactly replicate the option with only one risky asset in some cases(nodes) when the payoff of option is not linear to the stock price. In this case, we use the average method to find ns.

However, there are some cases when Su > Sm > Sd > Strike Price and the payoff of option is linear to the stock price. In this case, ns is same as the corresponding ns in binomial model. In other words, the middle branch doesn't change our delta and we use the same strategy as in binomial price model.

```
def ns_combine(payoff, p, u):
o_price = option_Binomial(payoff, p, u)

    ns = np.zeros([2*T + 1, T + 1])

    for t in range(T - 1, -1, -1):
      for i in range(0, 2*t + 1):
        Vu = o_price[i+(T-t)-1, t+1]
        Vd = o_price[i+(T-t)+1, t+1]
        Vm = o_price[i+(T-t),t+1]
        Su = stock[i+(T-t)-1, t+1]
        Sd = stock[i+(T-t)+1, t+1]
        Sm = stock[i+(T-t),t+1]
        if Sd > strike:
            ns[i+(T-t), t] = (Vu-Vd)/(Su - Sd)          #when Su > Sm > Sd > k
        else :
            ns[i+(T-t), t]= ((Vu-Vd)/(Su - Sd)+(Vu-Vm)/(Su-Sm)+(Vm-Vd)/(Sm-Sd))/3

    return ns
```

4. Errors from different strategies

4.1. Setting parameters and price the option

```
strike = 110
call_payoff = np.maximum(stock[:, T] - strike, 0)
N = 500000            # number of MC simulations
p =[2/9, 5/9, 2/9]     #p_u, p_m, p_d
X0 = option_Binomial(call_payoff, p, u)[T, 0]
```

4.2. Strategy 1

```
ns1 = ns_binomial(call_payoff, p, u)
error1 = find_error(X0, ns1, call_payoff, N, p)[0]
ST1 = find_error(X0, ns1, call_payoff, N, p)[1]
```

4.2.1. Mean and standard deviation

```
print('mean = ', np.mean(error1)) #The mean represents the accuracy of the option price.
print('standard deviation = ', np.std(error1))  #The standar deviation represents the accuracy of the strategy (Ns).
```

```
mean =  0.012067156963789558
standard deviation =  0.04101871216632948
```

4.2.2. Histogram

```
plt.hist(error1, bins = 100, range = (0,0.1))
plt.show()
```

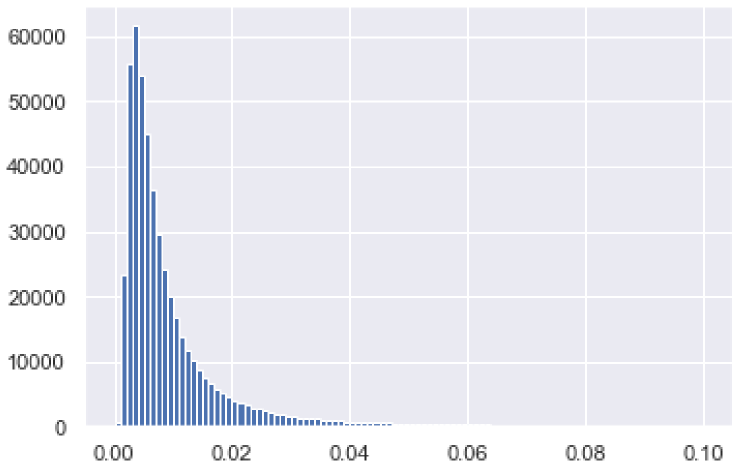


Figure 1. Error frequency graph of Strategy 1

4.2.3. Error distribution

```
plt.scatter(ST1, error1)
plt.xlabel('Log Stock Price')
plt.ylabel('Errors')
plt.show()
```

Volume 1 Issue 12, 2020
DOI: 10.6981/FEM.202012_1(12).0018

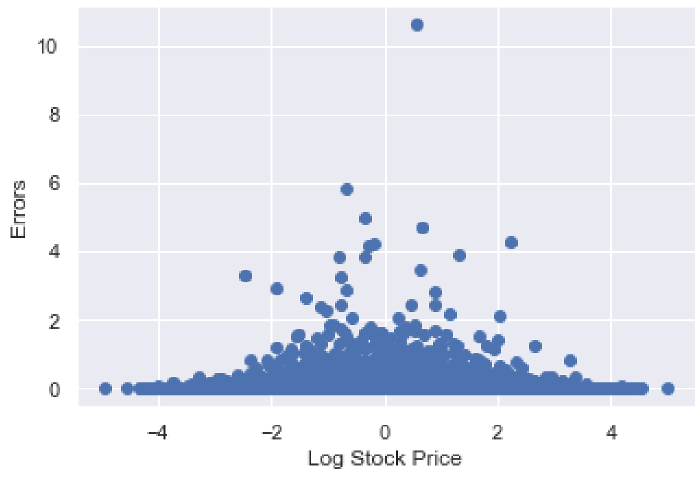


Figure 2. Error distribution of Strategy 1

4.3. Strategy 2

```
ns2 = ns_average(call_payoff, p, u)
error2 = find_error(X0, ns2, call_payoff, N, p)[0]
ST2 = find_error(X0, ns1, call_payoff, N, p)[1]
```

4.3.1. Mean and standard deviation

```
print('mean = ', np.mean(error2)) #The mean represents the accuracy of the option price.
print('standard deviation = ', np.std(error2))  #The standar deviation represents the accuracy of the strategy (Ns).
mean = 0.01205979059224262
standard deviation = 0.039159863125921124
```

4.3.2. Histogram

```
plt.hist(error2, bins = 100, range = (0,0.1))
plt.show()
```

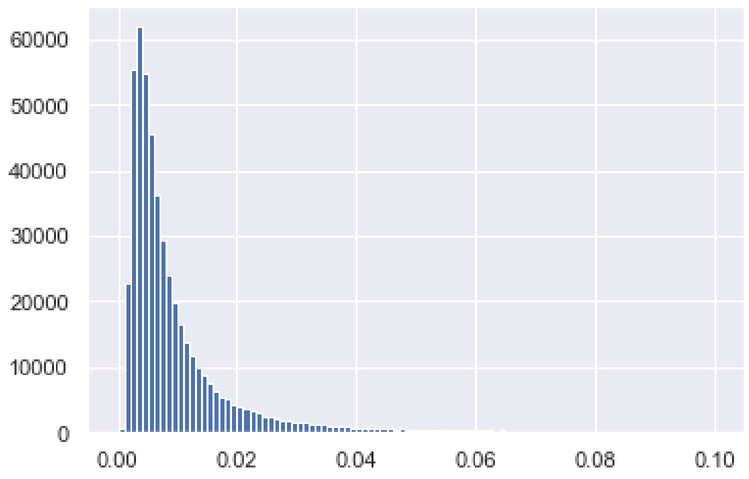


Figure 3. Error frequency graph of Strategy 2

4.3.3. Error distribution

```
plt.scatter(ST2, error2)
plt.xlabel('Log Stock Price')
plt.ylabel('Errors')
plt.show()
```

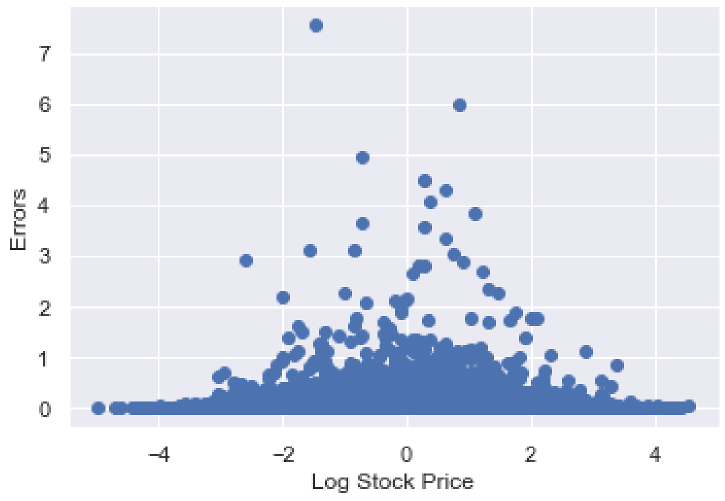


Figure 4. Error distribution of Strategy 2

4.4. Strategy 3

```
ns3 = ns_combine(call_payoff, p, u)
error3 = find_error(X0, ns3, call_payoff, N, p)[0]
ST3 = find_error(X0, ns1, call_payoff, N, p)[1]
```

4.4.1. Mean and standar deviation

```
print('mean = ', np.mean(error3)) #The mean represents the accuracy of the option price.
print('standard deviation = ', np.std(error3))  #The standar deviation represents the accuracy of the strategy (Ns).
```

```
mean = 0.012230871858268566
standard deviation = 0.04727977952724117
```

4.4.2. Histogram

```
plt.hist(error3, bins = 100, range = (0,0.1))
plt.show()
```

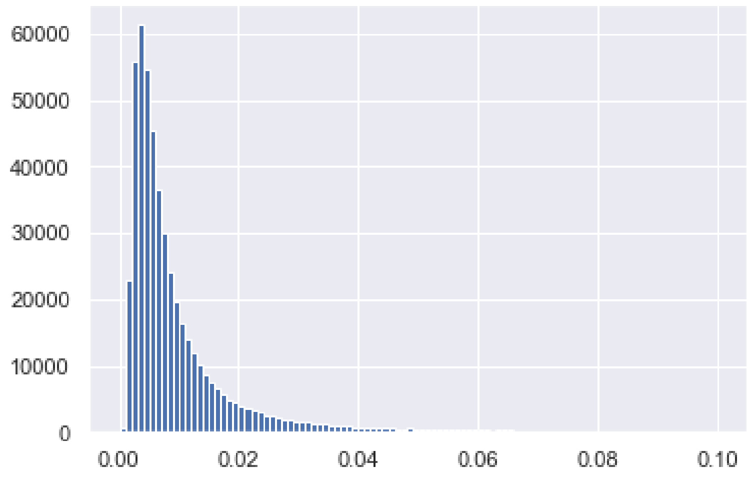


Figure 5. Error frequency graph of Strategy 3

4.4.3. Error distribution

```
plt.scatter(ST3, error3)
plt.xlabel('Log Stock Price')
plt.ylabel('Errors')
plt.show()
```

ISSN: 2692-7608 DOI: 10.6981/FEM.202012_1(12).0018

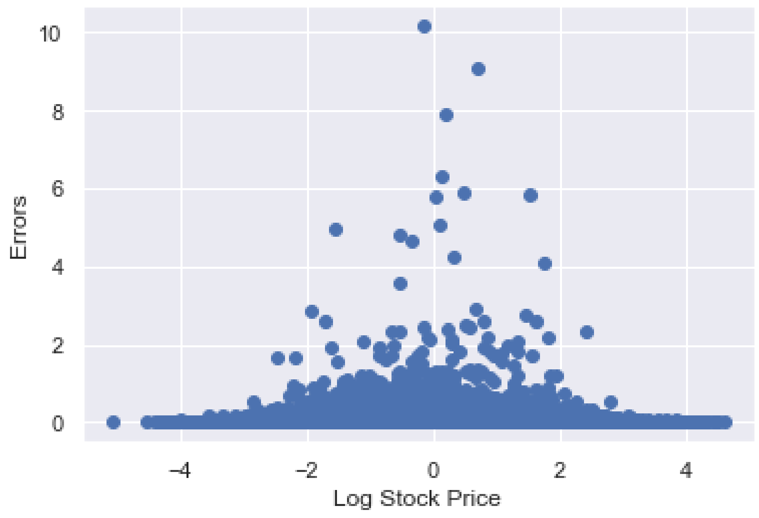


Figure 6. Error distribution of Strategy 3

5. Conclusion

Assuming T = 1000 and N = 500000, the mean of error1 is 0.01206, mean of error2 is 0.01205 and mean of error3 is 0.01205 (Error2 ≈ Error3 ≈ Error1); the standard deviation values for these three strategies are 0.04101, 0.03915 and 0.04727 respectively (StandardDeviation2 < StandardDeviation1 < StandardDeviation3) Then running Monte-Carlo simulation for times, the result shows their mean values stably stay around 0.012 and the standard deviation values are between 0.03 and 0.05. Comparing these three error distribution scatters, strategies 1 and 2 produced the least abnormal points, all the other errors are less than 6.0 except one abnormal point; at the same time, strategy 3 produced more errors greater than 6.0 with a higher upper limit around 10.0. Recording to these data, Strategy 1 and 2 whose delta equal to $\frac{V_u-V_d}{S_u-S_d}$ and $(\frac{V_u-V_d}{S_u-S_d}+\frac{V_u-V_m}{S_u-S_m}+\frac{V_m-V_d}{S_m-S_d})/3$ are better than strategy 3. But in effect, the results from these three strategies are similar and in this situation there's no obvious difference among these three strategies.

This research provides a different researching orientation towards option trinomial model and a detailed Python coding description. While other researchers are managing to find the neutral probability. This work avoids this problem, to be replaced by adjusting the Ns value to make the error near to zero.

References

[1] Cox, J.C, Ross, S.A, Rubinstein, M. (1979) Option Pricing: A Simplified Approach. Journal of Financial Economics 7: 229-263.

[2] Shreve, S.E. (2004) Stochastic Calculus for Finance I: The Binomial Asset Pricing Model. Springer-Verlag, New York.

[3] Haahtela, T. (2010). Recombining trinomial tree for real option valuation with changing volatility. In 14th Annual International Conference on Real Options theory meets practice, Rome, Italy.

[4] Dar, A.A, Anuradha, N., Rahman, B.S.A. (2017) Option Pricing Using Monte Carlo Simulation. British Journal of Economics, Finance and Management Sciences: March 2017, Vol. 13 (2)

[5] Bergman, Y.Z. (1981) A Characterization of Self-Financing Portfolio Strategies. Research Program in Finance Working Papers from University of California at Berkeley: No 113.

[6] Macdonald, A. S. (1997). The hypotheses underlying the pricing of options: a note on a paper by Bartels. Paper presented at Proceedings of the 7th International AFIR Colloquium, Cairns: 617-630.