112 選擇權-評價與應用
期末報告

林祥恩

January 7, 2024

Contents

1 Various option pricing methods

- 通用函數

```
# Utils
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats

def call(ST,K):
    return np.maximum(ST-K, 0)

def put(ST,K):
    return np.maximum(K-ST, 0)

def BS(S, K, T, r, div, sigma, option):
    d1 = (np.log(S/K) + (r-div+0.5*sigma**2)*T)/(sigma*T**0.5)
    d2 = d1 - (sigma*T**0.5)
    if option == 'call':
        return S*np.exp(-div*T)*norm.cdf(d1) - K*np.exp(-r*T)*norm.cdf(
            d2)
    elif option == 'put':
        return -S*np.exp(-div*T)*norm.cdf(-d1) + K*np.exp(-r*T)*norm.cdf
            (-d2)
```

1.1 Binomial tree model

(1) 二項數模型是透過 backward induction 的方式算出選擇權價格。其中又有兩種在參數上設定的方式，CRR 與 JR。

(2) CRR

$$u = e^{\sigma\sqrt{\Delta t}}$$
$$d = \frac{1}{u}$$
$$q = \frac{1}{2} + \frac{r\sqrt{\Delta t}}{2\sigma}$$

(3) JR

$$u = e^{(r-\frac{1}{2}\sigma^2)\Delta t+\sigma\sqrt{\Delta t}}$$
$$d = e^{(r-\frac{1}{2}\sigma^2)\Delta t-\sigma\sqrt{\Delta t}}$$
$$q = 0.5$$

(4) 定義二項數模型函數

```
def binomial_tree(S, K, T, r, div, sigma, N, method, option, Astyle=None
    ):
    dt = T/N # time increament
    df = np.exp(-r*dt) # discount factor
```

```
    payoff = {'call': call, 'put': put}[option]

    if method == 'CRR':
        u = np.exp(sigma*np.sqrt(dt))
        d = 1/u
        q = (np.exp((r-div)*dt)-d) / (u-d)
    elif method == 'JR':
        u = np.exp((r-div-0.5*sigma**2)*dt + sigma*dt**0.5)
        d = np.exp((r-div-0.5*sigma**2)*dt - sigma*dt**0.5)
        q = 0.5

    stock_path = np.zeros((N+1, N+1))
    stock_path[0,0] = S
    for i in range(1, N+1):
        for j in range(i+1):
            stock_path[i,j] = S * u**(i-j) * d**j

    option_value = np.maximum(payoff(stock_path[N], K), 0)
    for i in range(N, 0, -1):
        for j in range(i):
            option_value[j] = (q * option_value[j] + (1-q) *
                option_value[j+1]) * df
            if Astyle == True:
                option_value[j] = max(option_value[j], payoff(stock_path
                    [i-1, j], K))
    return option_value[0]
```

(5) 設定以下參數，進行模擬

$$S = 120, K = 120, T = 1, r = 0.01, div = 0.005, sigma = 0.2$$

其中，S: 標的物現價、K: 履約價格、T: 據到期日（年）、r: 無風險利率、div: 連續股利率、sigma: 標的物波動度。可以看出我們是將選擇權設定在價平，而後面根據不同計算價格的方法也會依從此參數設定。

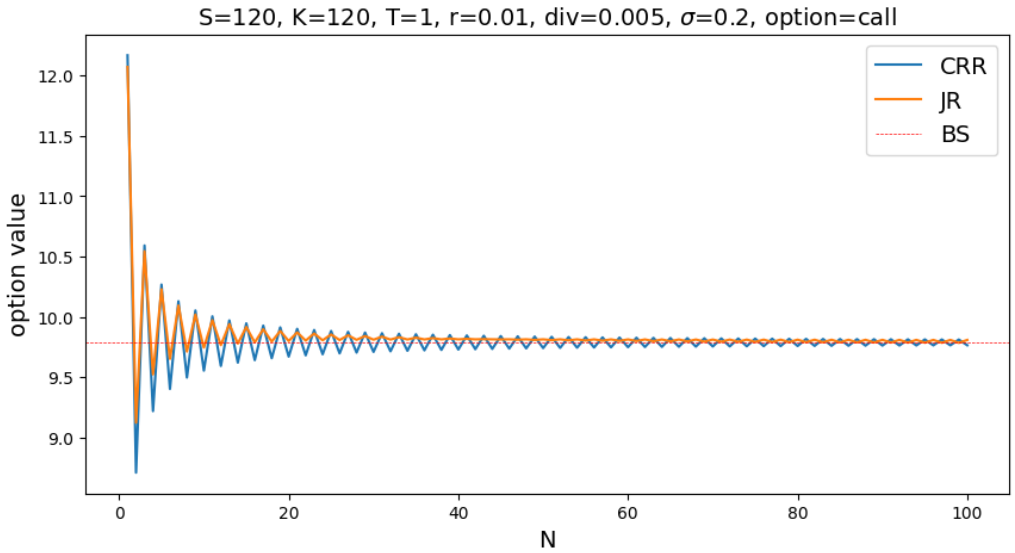
```
# (A) call
S = 120
K = 120
T = 1
r = 0.01
div = 0.005
sigma = 0.2
option = 'call'
N = np.arange(1, 101)
crr_res = []
jr_res = []
for n in N:
    crr_res.append(binomial_tree(S, K, T, r, div, sigma, n, 'CRR',
        option, True))
    jr_res.append(binomial_tree(S, K, T, r, div, sigma, n, 'JR', option,
        True))

plt.figure(figsize=(10,5))
plt.plot(N, crr_res, label='CRR')
```

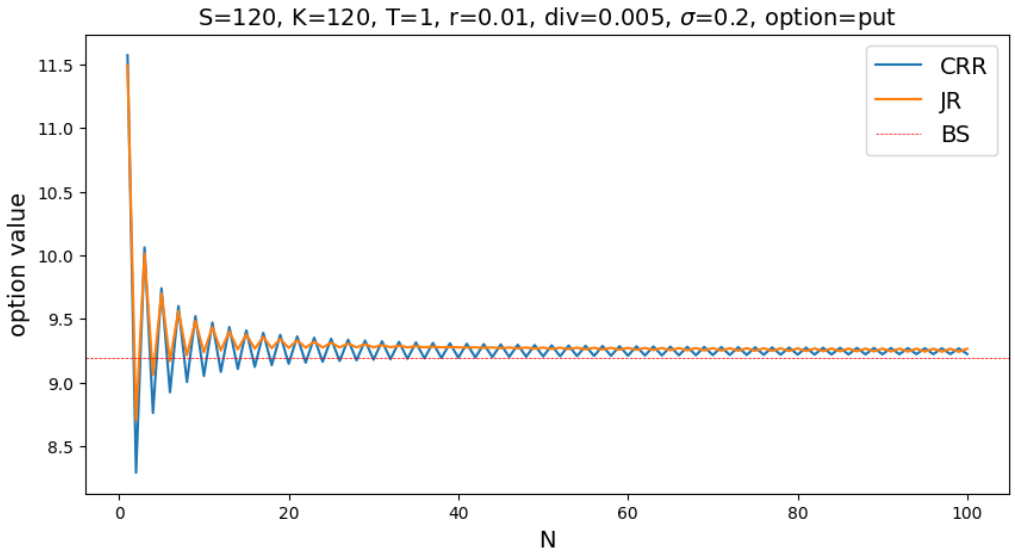
```
plt.plot(N, jr_res, label='JR')
plt.title(f'S={S}, K={K}, T={T}, r={r}, div={div}, $\sigma$={sigma}, option={option}', fontsize=14)
plt.xlabel('N', fontsize=14)
plt.ylabel('option value', fontsize=14)
bs_value = BS(S, K, T, r, div, sigma, option)
plt.axhline(y=bs_value, color='r', linestyle='--', label=f'BS', linewidth=0.5)
plt.legend(fontsize=14)
print(f'CRR={crr_res[-1]}, JR={jr_res[-1]}, BS={bs_value}')
crr_error_percentage = (crr_res[-1] - bs_value) / bs_value * 100
jr_error_percentage = (jr_res[-1] - bs_value) / bs_value * 100
print(f'CRR={crr_error_percentage:.2f}%, JR={jr_error_percentage:.2f}%')
print(f'Note: BS are European options')
"""
賣權同理
"""
```

```
========================
outcome
========================
CRR=9.764307675566378, JR=9.808156874880577, BS=9.788002127460274
CRR=-0.24%, JR=0.21%

CRR=9.223007917661175, JR=9.264655020195468, BS=9.192484674238571
CRR=0.33%, JR=0.79%

Note: BS are European options
```

(6) 圖形呈現

從圖形中可以清楚出美式賣權比歐式賣權來得貴，在嘗試不同價格後，此結論依然成立，符合我們的經濟直覺。買權的部分在價格上差距就不大。





1.2 Efficient analytic approximation(BAW)

(1) 此方法透過牛頓法找出臨界股票價格，用以估算美式選擇權的價格。

(2) 根據文章中的方法，將臨界股價的計算過程寫成函數。其中，文章中在數值方法上只有給買權的部分，賣權的部分由我們自己推導。如下：

* We know that

$$P_A(S^{**}) = P_E(S^{**}) - \frac{S^{**}}{q_1}(1 - e^{-(r-b)T}N(-d_1))$$

* Let

$$LHS(S^{**}) = P_A(S^{**}) = X - S^{**}$$
$$RHS(S^{**}) = P_E(S^{**}) - \frac{S^{**}}{q_1}(1 - e^{-(r-b)T}N(-d_1))$$

* Using Newton's method to solve

$$F(S_i) = LHS(S_i) - RHS(S_i)$$
$$S_{i+1} = S_i - \frac{F(S_i)}{F'(S_i)}$$

$$\begin{aligned}
F'(S_i) &= (X - S_i) - P_E(S_i) + \frac{S_i}{q_1}(1 - e^{-(r-b)T}N(-d_1)) \\
&= -1 + e^{-(r-b)T}N(-d_1) + \frac{1}{q_1}(1 - e^{-(r-b)T}N(-d_1)) + \frac{S_i}{q_1}(1 - e^{-(r-b)T}\frac{\partial N(-d_1)}{\partial d_1}\frac{\partial d_1}{\partial S_i}) \\
&= -1 + e^{-(r-b)T}N(-d_1) + \frac{1}{q_1} - \frac{e^{-(r-b)T}N(-d_1)}{q_1} + \frac{S_i}{q_1} + \frac{1}{q_1}\frac{e^{-(r-b)T}N'(d_1)}{\sigma\sqrt{T}} \\
&= -1 + e^{-(r-b)T}N(-d_1)(1 - \frac{1}{q_1}) + \frac{1}{q_1}(1 + S_i + \frac{e^{-(r-b)T}N'(d_1)}{\sigma\sqrt{T}}) \\
&= -1 + b_{i(put)}
\end{aligned}$$

* Therefore

$$\begin{aligned} S_{i+1} &= S_i - \frac{F(S_i)}{F'(S_i)} \\ &= \frac{-X + b_i S_i + RHS(S_i)}{-1 + b_{i(put)}} \end{aligned}$$

完成以上推導後，就可以開始刻程式。

```
def _Kc(X, t, r, b, Sigma, option):
    _ITERATION_MAX_ERROR = 0.00001
    if option == 'call':
        N = 2 * b / Sigma**2
        M = 2 * r / Sigma**2
        q2u = (-1 * (N - 1) + ((N - 1)**2 + 4 * M)**0.5) / 2
        su = X / (1 - 1 / q2u)
        h2 = -1 * (b * t + 2 * Sigma * np.sqrt(t)) * X / (su - X)
        Si = X + (su - X) * (1 - np.exp(h2))
        K = (1 - np.exp(-1 * r * t))
        d1 = (np.log(Si/X) + (b + Sigma**2 / 2) * t)/(Sigma * np.sqrt(t)
            )
        q2 = (-1 * (N - 1) + ((N - 1)**2 + 4 * M / K)**0.5) / 2
        LHS = Si - X
        RHS = BS(Si, X, t, r, r-b, Sigma, option) + (1 - np.exp((b - r)
            * t) * stats.norm.cdf(d1,0.0,1.0)) * Si / q2
        bi = np.exp((b - r) * t) * stats.norm.cdf(d1,0.0,1.0) * (1 - 1 /
             q2) + (1 - np.exp((b - r) * t) * stats.norm.pdf(d1,0.0,1.0)
            / (Sigma * np.sqrt(t))) / q2

        E = _ITERATION_MAX_ERROR
        print(f'=======Call======')
        print(f'seed value誤差{(LHS - RHS)/X}')
        while np.abs(LHS - RHS) / X > E:
            Si = (X + RHS - bi * Si) / (1 - bi)
            d1 = (np.log(Si / X) + (b + Sigma**2 / 2) * t) / (Sigma * np
                .sqrt(t))
            LHS = Si - X
            RHS = BS(Si, X, t, r, r-b, Sigma, option) + (1 - np.exp((b -
                 r) * t) * stats.norm.cdf(d1,0.0,1.0)) * Si / q2
            bi = np.exp((b - r) * t) * stats.norm.cdf(d1,0.0,1.0) * (1 -
                 1 / q2) + (1 - np.exp((b - r) * t) * stats.norm.cdf(d1
                ,0.0,1.0) / (Sigma * np.sqrt(t))) / q2

        return Si

    elif option == 'put':
        N = 2 * b / Sigma**2
        M = 2 * r / Sigma**2
        q1u = (-1 * (N - 1) - ((N - 1)**2 + 4 * M)**0.5) / 2
        su = X / (1 - 1 / q1u)
        h1 =  (b * t - 2 * Sigma * np.sqrt(t)) * X / (X - su)
        Si = su + (X - su) * np.exp(h1)

        K = (1 - np.exp(-1 * r * t))
        d1 = (np.log(Si/X) + (b + Sigma**2 / 2) * t)/(Sigma * np.sqrt(t)
            )
        q1 = (-(N - 1) - ((N - 1)**2 + 4 * M / K)**0.5)/ 2
```

```
        LHS = X - Si
        RHS = BS(Si, X, t, r, r-b, Sigma, option) - (1 - np.exp((b - r)
            * t) * stats.norm.cdf(-d1,0.0,1.0)) * Si / q1
        b2i = np.exp((b - r) * t) * stats.norm.cdf(-d1,0.0,1.0) * (1 + 1
             / q1) + (1 + Si + np.exp((b - r) * t) * stats.norm.pdf(d1
            ,0.0,1.0) / (Sigma * np.sqrt(t))) / q1

        E = _ITERATION_MAX_ERROR
        print(f'=======Put=======')
        print(f'seed value誤差{(LHS - RHS)/X}')
        while np.abs(LHS - RHS) / X > E:
            Si = (-X + RHS + b2i * Si) / (-1 + b2i)
            d1 = (np.log(Si / X) + (b + Sigma**2 / 2) * t) / (Sigma * np
                .sqrt(t))
            LHS = X - Si
            RHS = BS(Si, X, t, r, r-b, Sigma, option) - (1 - np.exp((b -
                 r) * t) * stats.norm.cdf(-d1,0.0,1.0)) * Si / q1
            b2i = np.exp((b - r) * t) * stats.norm.cdf(-d1,0.0,1.0) * (1
                 + 1 / q1) + (1 + Si + np.exp((b - r) * t) * stats.norm.
                pdf(d1,0.0,1.0) / (Sigma * np.sqrt(t))) / q1  # p25.

        return Si
```

(3) 接著計算出臨界股價，參數與過去相同。

```
S = 120
X = 120  # K
t = 1  # 選擇權到期時間 (年)
r = 0.01
b = 0.005  # 常數 (持有成本 b=r-q, 設q=0.005)
Sigma = 0.2  # 波動率

option = 'call'
S_star = _Kc(X, t, r, b, Sigma, option)

print("買權臨界股價 S* =", S_star)

option = 'put'
S_star = _Kc(X, t, r, b, Sigma, option)
print("賣權臨界股價 S* =", S_star)
========================
outcome
========================
買權臨界股價 S* = 276.20601135913154
賣權臨界股價 S* = 82.35592758113108
```

(4) 最後根據文章的近似解公式，求算美式選擇權價格。

```
def _approximateAmericanOption(S, X, t, r, b, Sigma, option):
    if option == 'put':
        # 美式賣權有無股利都比歐式賣權來得大
```

```
        Sk = _Kc(X, t, r, b, Sigma, option)
        N = 2 * b / Sigma**2
        M = 2 * r / Sigma**2
        K = (1 - np.exp(-1 * r * t))
        d1 = (np.log(Sk / X) + (b + (Sigma**2) / 2) * t) / (Sigma * (t
            **0.5))
        q1 = (-(N - 1) - ((N - 1)**2 + 4 * M / K)**0.5) / 2
        # print(q1)
        A1 = -(Sk / q1) * (1 - np.exp((b - r) * t) * stats.norm.cdf(-d1
            ,0.0,1.0))  # P21.
        if S > Sk:
            return BS(S, X, t, r, r-b, Sigma, option) + A1 * (S / Sk)**
                q1  # 大於臨界股價 P21.
        else:
            return X - S
    """
    買權同理
    """
option = 'call'
Amcall_price = _approximateAmericanOption(S, X, t, r, b, Sigma, option)
print("美式買權價格 =", Amcall_price)

option = 'put'
Amput_price = _approximateAmericanOption(S, X, t, r, b, Sigma, option)
print("美式賣權價格 =", Amput_price)
========================
outcome
========================
美式買權價格 = 9.788365616957941
美式賣權價格 = 9.235596741592884
```

(5) 回頭看二項數的結果，會發現 BAW 方法不論是買權或是賣權都介在 CRR 與 JR 之間。

1.3 Least-squares Monte Carlo simulation

(1) 此方法透過 regression analysis 模擬未來的價格路徑，在每個時間點計算執行和不執行選擇權，以找出最佳履約時點。

```
    模擬股價路徑
        def stock_simu(S, T, r, div, sigma, N, M, seed):
            dt = T / N
            np.random.seed(seed)
            paths = np.zeros((N + 1, M))
            paths[0] = S

            for t in range(1, N + 1):
                Z = np.random.standard_normal(int(M/2))
                Z = np.concatenate((Z, -Z))
                paths[t] = paths[t - 1] * np.exp((r - div - 0.5 *
                    sigma**2) * dt + sigma * np.sqrt(dt) * Z)
            return paths
```

(2) 使用三種不同的 regression analysis 方式，OLS、WLS 與 LASSO。

i. OLS

```
def least_squares_MC(S, K, T, r, div, sigma, M, N, order, option, seed):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N))
    stock_paths = stock_simu(S, T, r, div, sigma, N, M, seed)

    payoffs = payoff(stock_paths, K)

    exercise_values = np.zeros_like(payoffs)
    exercise_values[-1] = payoffs[-1]

    for t in range(N-1, 0, -1):
        in_the_money = payoffs[t]>=0
        reg = np.polyfit(stock_paths[t][in_the_money], exercise_values[t
            +1][in_the_money]*df, order)

        C = np.polyval(reg, stock_paths[t][in_the_money])
        exercise_values[t][in_the_money] = np.where(payoffs[t][
            in_the_money] > C,
        payoffs[t][in_the_money],exercise_values[t+1][in_the_money] * df
            )
        exercise_values[t][~in_the_money] = payoffs[t+1][~in_the_money]
            * df
    return np.mean(exercise_values[1]*df), np.std(exercise_values[1]*df)
        /M**0.5
```

ii. WLS

```
import statsmodels.api as sm

def least_squares_MC_with_WLS(S, K, T, r, div, sigma, M, N, order,
    option, seed):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N))
    stock_paths = stock_simu(S, T, r, div, sigma, N, M, seed)
    payoffs = payoff(stock_paths, K)
    exercise_values = np.zeros_like(payoffs)
    exercise_values[-1] = payoffs[-1]

    for t in range(N-1, 0, -1):
        in_the_money = payoffs[t] >= 0
        x = stock_paths[t][in_the_money]
        y = exercise_values[t+1][in_the_money] * df

        # WLS
        wls_model = sm.WLS(y, sm.add_constant(x), weights=1/x)
        results = wls_model.fit()
        C = results.predict(sm.add_constant(x))

        exercise_values[t][in_the_money] = np.where(payoffs[t][
            in_the_money] > C,
        payoffs[t][in_the_money],exercise_values[t+1][in_the_money] * df
            )
```

```
        exercise_values[t][~in_the_money] = payoffs[t+1][~in_the_money]
            * df

    return np.mean(exercise_values[1] * df), np.std(exercise_values[1] *
        df) / M**0.5
```

iii. LASSO

```
from sklearn.linear_model import Lasso

def least_squares_MC_with_LASSO(S, K, T, r, div, sigma, M, N, order,
    option, seed):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r * (T / N))
    stock_paths = stock_simu(S, T, r, div, sigma, N, M, seed)

    payoffs = payoff(stock_paths, K)

    exercise_values = np.zeros_like(payoffs)
    exercise_values[-1] = payoffs[-1]

    for t in range(N - 1, 0, -1):
        in_the_money = payoffs[t] >= 0

        # 使用 LASSO
        reg = Lasso(alpha=0.5)
        reg.fit(stock_paths[t][in_the_money].reshape(-1, 1),
            exercise_values[t + 1][in_the_money] * df)
        C = reg.predict(stock_paths[t][in_the_money].reshape(-1, 1))

        exercise_values[t][in_the_money] = np.where(payoffs[t][
            in_the_money] > C,
        payoffs[t][in_the_money],exercise_values[t + 1][in_the_money] *
            df)
        exercise_values[t][~in_the_money] = payoffs[t + 1][~in_the_money
            ] * df
    return np.mean(exercise_values[1] * df), np.std(exercise_values[1] *
        df) / M**0.5
```

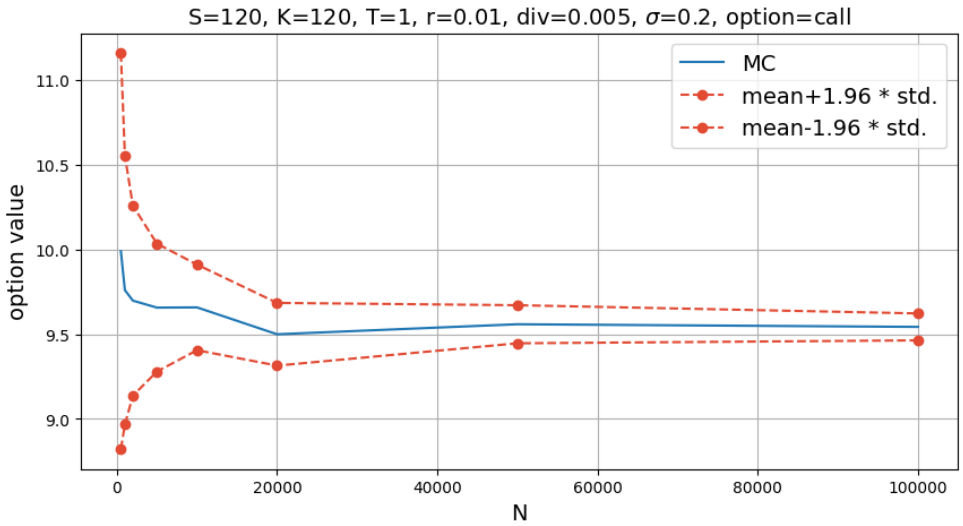
(3) 使用與前面相同的參數，輸出模擬結果。
可以看出除了 OLS 方法以外，其他方法模擬出的價格都嚴重低估了應有的價值，甚至比歐式選擇權價值還小。此結果似乎不讓人感到意外，因找不到適合的經濟意涵合理化 WLS 與 LASSO 在 regression analysis 過程中不把🄵斷面資料等同對待的理由。

```
# OLS
==================
Call
------------------
Monte Carlo mean: 9.543
Monte Carlo std: 0.041
Monte Carlo CI: [9.462, 9.624]
```

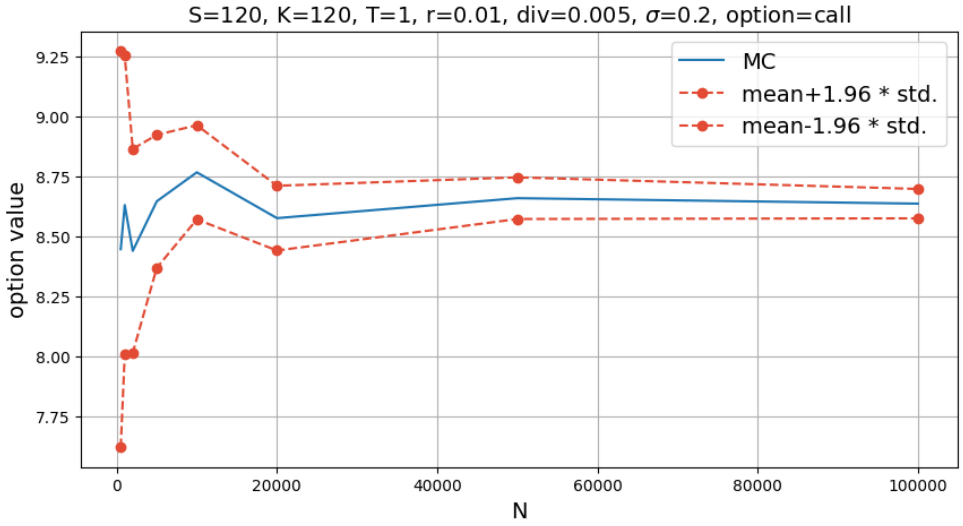
```
------------------
Put
------------------
Monte Carlo mean: 9.060
Monte Carlo std: 0.032
Monte Carlo CI: [8.996, 9.124]

# WLS
==================
Call
------------------
Monte Carlo mean: 8.636
Monte Carlo std: 0.031
Monte Carlo CI: [8.574, 8.699]
------------------
Put
------------------
Monte Carlo mean: 8.188
Monte Carlo std: 0.027
Monte Carlo CI: [8.134, 8.243]

# LASSO
==================
Call
------------------
Monte Carlo mean: 8.624
Monte Carlo std: 0.033
Monte Carlo CI: [8.559, 8.690]
------------------
Put
------------------
Monte Carlo mean: 8.197
Monte Carlo std: 0.026
Monte Carlo CI: [8.146, 8.248]
```

(4) 圖形呈現

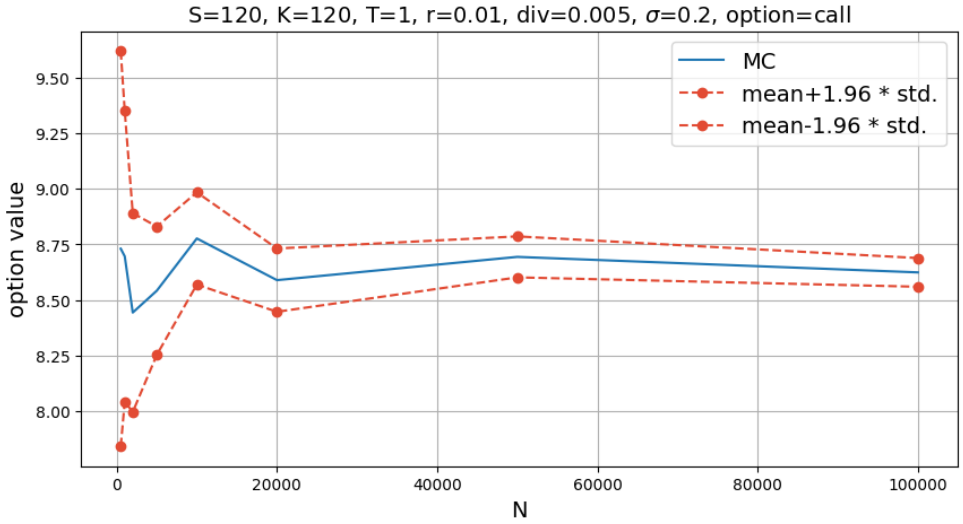
i. OLS



ii. WLS



iii. LASSO



1.4 Summary

	Binomial tree model(CRR)	Efficient analytic approximation(BAW)	Least-squares Monte Carlo simulation
Call	9.7643	9.7883	[9.462, 9.624]
Put	9.2230	9.2355	[8.996, 9.124]
Speed	0.8s	0.2s	2.5s

2 Extension works

2.1 Volatility surface of TXO

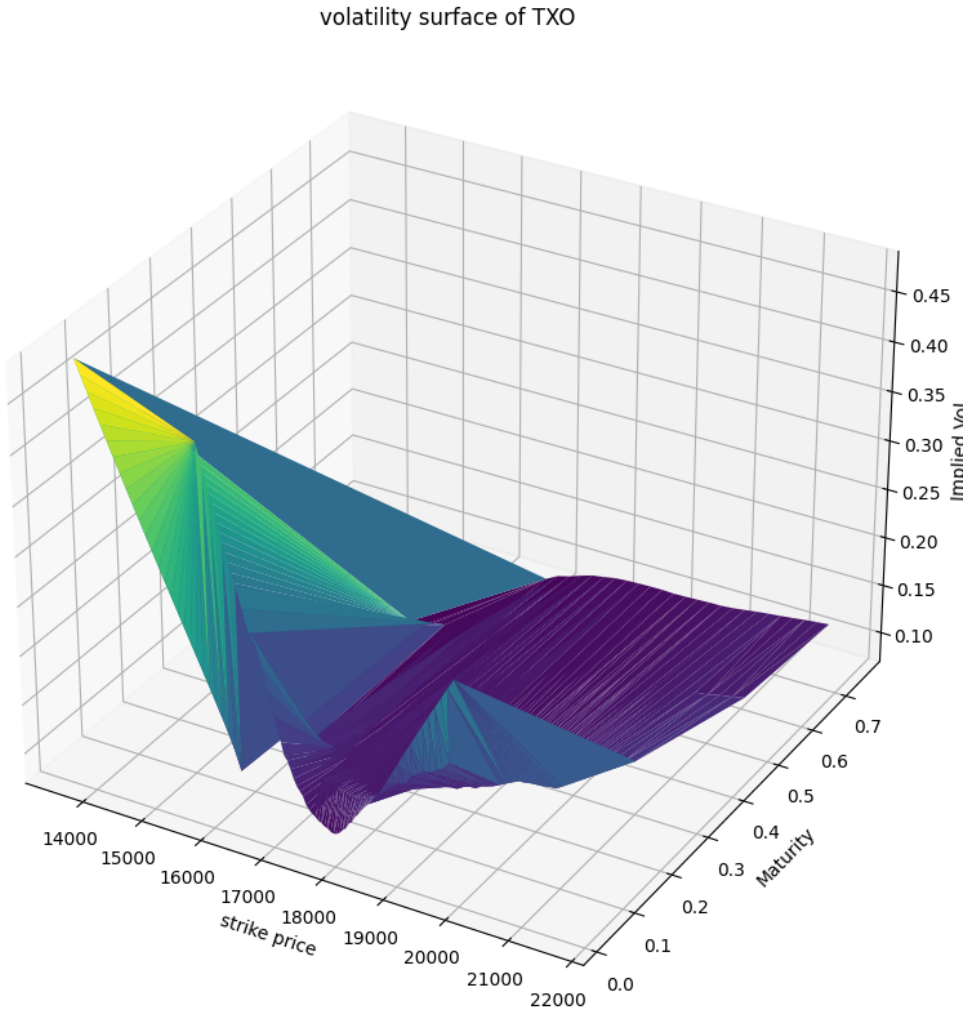
(1) 資料來源：台灣期貨交易所 https://www.taifex.com.tw/cht/3/optDailyMarketReport

(2) 標的：臺指選擇權 TXO

(3) 到期月分：202401W2、202401、202402、202403、202406、202409

(4) 標的商品現價：17510

(5) Volatility surface



(6) 發現

首先，觀察到不同履約價 (strike price) 下的隱含波動率有所不同，這可能是因 Skew 或 Smile 效應。Skew 指的是相同到期期限下不同履約價的隱含波動率不同，而 Smile 效應則指相同到期期限下隱含波動率隨著履約價的不同而呈現的 U 型的 curve。其次，觀察不同到期期限下的隱含波動率，顯示出 Term Structure 的變化。長期期限下的隱含波動率較高，這可能是由於長期風險和不確定性選擇權價格的影響。最後，將隱含波動率與實際過去一段時間中的波動率進行比較。隱含波動率與實際波動率之間的差🄵可能意味著市場對未來波動性的預期與實際情形不同。這些結果提供了有關市場選擇權隱含波動率曲面的洞察，有助於理解市場對未來波動性的預期以及選擇權價格形成的過程。

(7) Proof of work

i. 使用 scipy 套件中提供的牛頓法計算隱含波動度

```
from scipy import optimize
from scipy.stats import norm

def calculate_implied_volatility(option_price, S, K, r, T, option_type):
    def black_scholes(sigma):
        d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.
            sqrt(T))
        d2 = d1 - sigma * np.sqrt(T)
        if option_type == 'Call':
            return S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
                - option_price
        else:  # Put option
            return K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1
                ) - option_price

    return optimize.newton(black_scholes, 0.5, tol=1e-10, maxiter=1000)
        # 牛頓法
```

ii. 利用期交所提供的選擇權報價，計算各條件下的隱含波動度

```
# 台指期現價:17510
import pandas as pd
a = pd.read_csv('/Users/shawn/Github/M1/選擇權-評價與應用/Final_project/
    call.csv')
implied_volatilities = []
for index, row in a.iterrows():
    option_price = row['結算價']
    S = 17510
    K = row['履約價']
    r = 0.016  # 台灣銀行一年定存利率
    T = row['到期時間（年）']
    implied_volatility = calculate_implied_volatility(option_price, S, K
        , r, T, 'Call')
    implied_volatilities.append(implied_volatility)
    # print(implied_volatilities)

a['implied_vola'] = implied_volatilities
```

iii. 繪圖

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


a['履約價'] = pd.to_numeric(a['履約價'], errors='coerce')
a['implied_vola'] = pd.to_numeric(a['implied_vola'], errors='coerce')
```

```
a['到期時間（年）'] = pd.to_numeric(a['到期時間（年）'], errors='coerce'
    )

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

x = a['履約價']
y = a['到期時間（年）']
z = a['implied_vola']

ax.plot_trisurf(x, y, z, cmap='viridis', edgecolor='none')

ax.set_xlabel('strike␣price')
ax.set_ylabel('Maturity')
ax.set_zlabel('Implied␣Vol')

plt.title('volatility␣surface␣of␣TXO')

plt.show()
```