選擇權評價期末報告

金融碩一 劉冠銘 112352014

(一) 報告概述

本次報告內容將會分成三部分：第一部分為基本要求，比較三種評價模型，第二部分為延伸研究，第三部分則是每個主題對應之程式檔案與資料說明，另外為方便助教比較程式，已將所有程式與輸出放上來。

(二) 美式選擇權評價

使用三種評價方式來評價美式選擇權，分別為二椽樹模型、最小平方蒙地卡羅法（bonus：在此段落中也會附上將 OLS 換成其他種回歸的結果）以及 BAW 模型。

1. 參數設定（後面之評價也以皆此組參數進行試驗）：
   i. S = 40
   ii. K = 40
   iii. T = 1
   iv. r = 0.06
   v. sigma = 0.2
2. CRR：

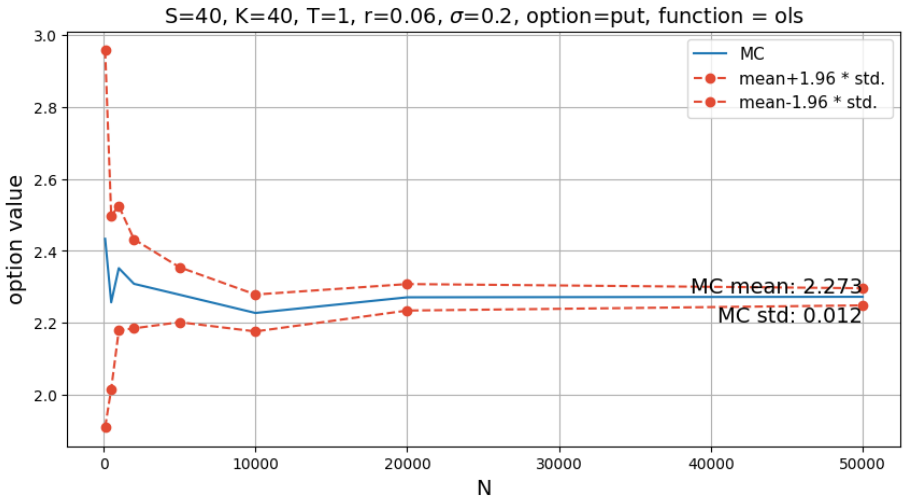
```python
def binomial_tree(S, K, T, r, sigma, N, method, option, Astyle=None): # N代表步數
    dt = T/N # time increament
    df = np.exp(-r*dt) # discount factor

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

    # u, q, d 的計算方式不同
    if method == 'CRR':
        u = np.exp(sigma*np.sqrt(dt))
        d = 1/u
        q = (np.exp(r*dt)-d) / (u-d)
    elif method == 'JR':
        u = np.exp((r-0.5*sigma**2)*dt + sigma*dt**0.5)
        d = np.exp((r-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]
```

```python
N = paths
crr_res = []
jr_res = []
crr_time = []
jr_time = []
for n in N:
    t1 = time()
    crr = (binomial_tree(S, K, T, r, sigma, n, 'CRR', option, True))
    t2 = time()
    crr_res.append(crr)
    crr_time.append(t2-t1)

    t1 = time()
    jr = (binomial_tree(S, K, T, r, sigma, n, 'JR', option, True))
    t2 = time()
    jr_res.append(jr)
    jr_time.append(t2-t1)

M = paths # path代表中間的步數
N = 50 #幾條path
mc_mean = []
mc_std = []
mc_time = []
for n in M:
    '''這邊代表用的步數，與各自對應的標準差與平均值，以及花費的時間'''
    t1 = time()
    m, s, _ = least_squares_MC_ols(S, K, T, r, sigma, n, N, 5, option) # n表示步數，N代表幾條PATH
    t2 = time()
    mc_time.append(t2-t1)
    mc_mean.append(m)
    mc_std.append(s)
```

3. 最小平方蒙地卡羅法:

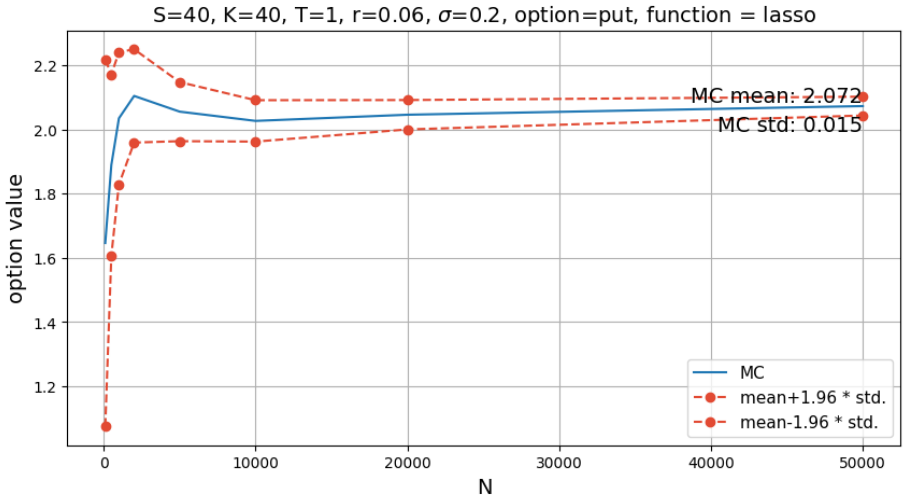
i. OLS：

```python
def least_squares_MC_ols(S, K, T, r, sigma, M, N, order = 5, option='put', seed=123): # M 表示步數，N代表共有幾條PATH
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N)) # discount factor
    stock_paths = stock_simu(S, T, r, 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, least_squares_MC_ols.__name__
```



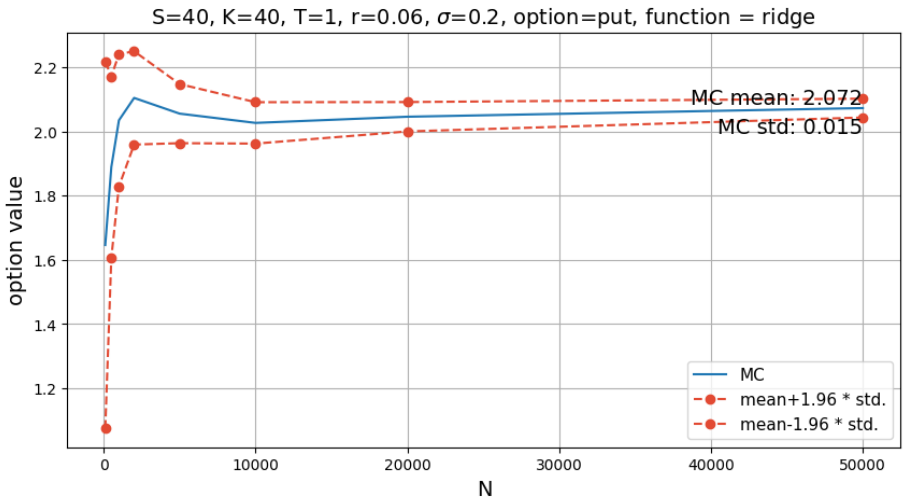
ii. Lasso：

```python
def least_squares_MC_lasso(S, K, T, r, sigma, M, N, option='put', seed=123):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N)) # discount factor
    stock_paths = stock_simu(S, T, r, 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

        lasso_model = LassoCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)  # 設定 Lasso 模型的 alpha 參數
        X_train = stock_paths[t][in_the_money].reshape(-1, 1)  # 將數據轉換為列向量
        y_train = exercise_values[t+1][in_the_money]*df
        lasso_model.fit(X_train, y_train)  # 訓練 Lasso 模型
        reg = [lasso_model.intercept_, lasso_model.coef_[0]]
        #print(f"The best alpha value for lasso is: {lasso_model.alpha_}")
        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, least_squares_MC_lasso.__name__
```



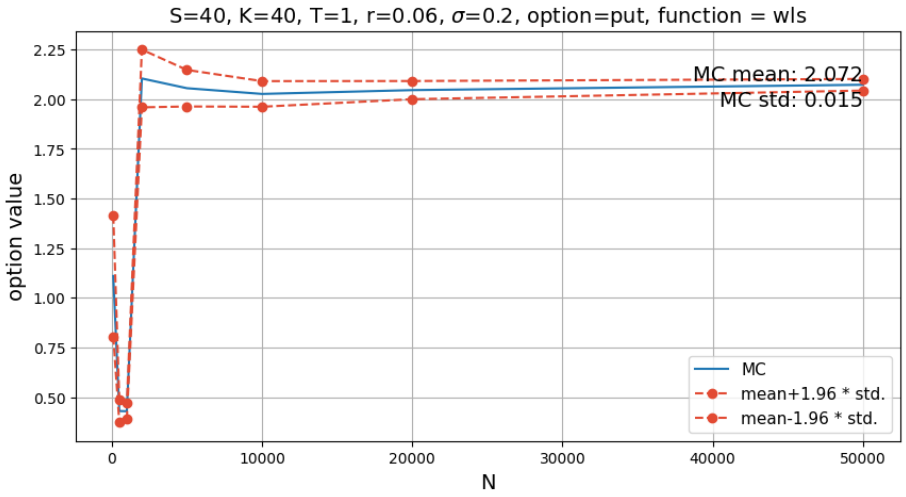
iii. Ridge：

```python
def least_squares_MC_ridge(S, K, T, r, sigma, M, N, option='put', seed=123):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N)) # discount factor
    stock_paths = stock_simu(S, T, r, 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

        ridge_model = RidgeCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)  # 設定 Lasso 模型的 alpha 參數
        X_train = stock_paths[t][in_the_money].reshape(-1, 1)  # 將數據轉換為列向量
        y_train = exercise_values[t+1][in_the_money]*df
        ridge_model.fit(X_train, y_train)  # 訓練 Lasso 模型
        reg = [ridge_model.intercept_, ridge_model.coef_[0]]
        #print(f"The best alpha value for ridge is: {ridge_model.alpha_}")
        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, least_squares_MC_ridge.__name__
```



iv. WLS：

```python
def least_squares_MC_wls(S, K, T, r, sigma, M, N, option='put', seed=123):
    payoff = {'call': call, 'put': put}[option]
    df = np.exp(-r*(T/N)) # discount factor
    stock_paths = stock_simu(S, T, r, 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
        #print(stock_paths[t][in_the_money])
        # 替換原本的 polyfit 部分
        X_train = sm.add_constant(stock_paths[t][in_the_money])  # 添加截距項
        y_train = exercise_values[t+1][in_the_money] * df
        weights = 1 / np.arange(1, len(y_train) + 1)  # 這裡使用了一個簡單的權重，你可以根據需要調整

        wls_model = sm.WLS(y_train, X_train, weights=weights)
        results = wls_model.fit()

        reg = results.params
        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, least_squares_MC_wls.__name__
```



v. 不同回歸函數之結果比較（for put）：

Function	Mean	std

OLS	2.273	0.012
Lasso	2.072	0.015
Ridge	2.072	0.015
WLS	2.072	0.015

可從上方結論表格看出，Lasso、Ridge 與 WLS 除了收斂情況有差以外，近乎都收斂到了相似的價格，而 OLS 則是與其他三種回歸函數所跑出來的結果有差異。

4. BAW：

最後得到的結果為:

Call：3.434

Put：2.781

Code：

```
ITERATION_MAX_ERROR = 0.00001

def priceEuropeanOption(option_type_flag, S, X, T, r, b, v):
    '''
    Black-Scholes
    '''

    d1 = (np.log(S / X) + (b + v**2 / 2) * T) / (v * (T)**0.5)
    d2 = d1 - v * (T)**0.5

    if option_type_flag == 'Call':
        bsp = S * np.exp((b - r) * T) * norm.cdf(d1) - X * np.exp(-r * T) * norm.cdf(d2)
    else:
        bsp = X * np.exp(-r * T) * norm.cdf(-d2) - S * np.exp((b - r) * T) * norm.cdf(-d1)

    return bsp

def priceAmericanOption(option_type_flag, S, X, T, r, b, v):
    '''
    Barone-Adesi-Whaley
    '''

    if option_type_flag == 'Call':
        return approximateAmericanCall(S, X, T, r, b, v)
    elif option_type_flag == 'Put':
        return approximateAmericanPut(S, X, T, r, b, v)
```

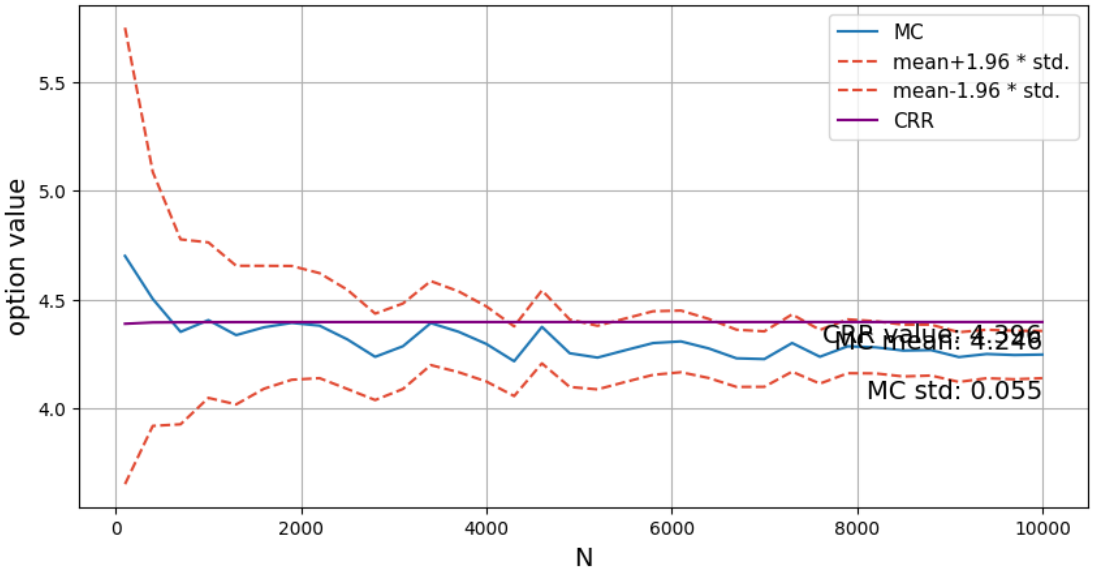
```python
def approximateAmericanCall(S, X, T, r, b, v):
    '''
    Barone-Adesi And Whaley
    '''

    if b >= r:
        return priceEuropeanOption('Call', S, X, T, r, b, v)
    else:
        Sk = Kc(X, T, r, b, v)
        N = 2 * b / v**2
        k = 2 * r / (v**2 * (1 - np.exp(-1 * r * T)))
        d1 = (np.log(Sk / X) + (b + (v**2) / 2) * T) / (v * (T**0.5))
        Q2 = (-1 * (N - 1) + ((N - 1)**2 + 4 * k))**0.5 / 2
        a2 = (Sk / Q2) * (1 - np.exp((b - r) * T) * norm.cdf(d1))
        if S < Sk:
            return priceEuropeanOption('Call', S, X, T, r, b, v) + a2 * (S / Sk)**Q2
        else:
            return S - X
def approximateAmericanPut(S, X, T, r, b, v):
    '''
    Barone-Adesi-Whaley
    '''

    Sk = Kp(X, T, r, b, v)
    N = 2 * b / v**2
    k = 2 * r / (v**2 * (1 - np.exp(-1 * r * T)))
    d1 = (np.log(Sk / X) + (b + (v**2) / 2) * T) / (v * (T)**0.5)
    Q1 = (-1 * (N - 1) - (((N - 1)**2 + 4 * k))**0.5) / 2
    a1 = -1 * (Sk / Q1) * (1 - np.exp((b - r) * T) * norm.cdf(-1 * d1))

    if S > Sk:
        return priceEuropeanOption('Put', S, X, T, r, b, v) + a1 * (S / Sk)**Q1
    else:
        return X - S
```

```python
def Kc(X, T, r, b, v):

    N = 2 * b / v**2
    m = 2 * r / v**2
    q2u = (-1 * (N - 1) + ((N - 1)**2 + 4 * m)**0.5) / 2
    su = X / (1 - 1 / q2u)
    h2 = -1 * (b * T + 2 * v * (T)**0.5) * X / (su - X)
    Si = X + (su - X) * (1 - np.exp(h2))

    k = 2 * r / (v**2 * (1 - np.exp(-1 * r * T)))
    d1 = (np.log(Si / X) + (b + v**2 / 2) * T) / (v * (T)**0.5)
    Q2 = (-1 * (N - 1) + ((N - 1)**2 + 4 * k)**0.5) / 2
    LHS = Si - X
    RHS = priceEuropeanOption('Call', Si, X, T, r, b, v) + (1 - np.exp((b - r) * T) * norm.cdf(d1)) * Si / Q2
    bi = np.exp((b - r) * T) * norm.cdf(d1) * (1 - 1 / Q2) + (1 - np.exp((b - r) * T) * norm.pdf(d1) / (v * (T)**0.5)) / Q2

    E = ITERATION_MAX_ERROR

    while np.abs(LHS - RHS) / X > E:
        Si = (X + RHS - bi * Si) / (1 - bi)
        d1 = (np.log(Si / X) + (b + v**2 / 2) * T) / (v * (T)**0.5)
        LHS = Si - X
        RHS = priceEuropeanOption('Call', Si, X, T, r, b, v) + (1 - np.exp((b - r) * T) * norm.cdf(d1)) * Si / Q2
        bi = np.exp((b - r) * T) * norm.cdf(d1) * (1 - 1 / Q2) + (1 - np.exp((b - r) * T) * norm.cdf(d1) / (v * (T)**0.5)) / Q2

    return Si
```

```python
def Kp(X, T, r, b, v):

    N = 2 * b / v**2
    m = 2 * r / v**2
    q1u = (-1 * (N - 1) - ((N - 1)**2 + 4 * m)**0.5) / 2
    su = X / (1 - 1 / q1u)
    h1 = (b * T - 2 * v * (T)**0.5) * X / (X - su)
    Si = su + (X - su) * np.exp(h1)

    k = 2 * r / (v**2 * (1 - np.exp(-1 * r * T)))
    d1 = (np.log(Si / X) + (b + v**2 / 2) * T) / (v * (T)**0.5)
    Q1 = (-1 * (N - 1) - ((N - 1)**2 + 4 * k)**0.5) / 2
    LHS = X - Si
    RHS = priceEuropeanOption( 'Put', Si, X, T, r, b, v) - (1 - np.exp((b - r) * T) * norm.cdf(-1 * d1)) * Si / Q1
    bi = -1 * np.exp((b - r) * T) * norm.cdf(-1 * d1) * (1 - 1 / Q1) - (1 + np.exp((b - r) * T) * norm.pdf(-d1) / (v * (T)**0.5)) / Q1

    E = ITERATION_MAX_ERROR

    while np.abs(LHS - RHS) / X > E:
        Si = (X - RHS + bi * Si) / (1 + bi)
        d1 = (np.log(Si / X) + (b + v**2 / 2) * T) / (v * (T)**0.5)
        LHS = X - Si
        RHS = priceEuropeanOption('Put', Si, X, T, r, b, v) - (1 - np.exp((b - r) * T) * norm.cdf(-1 * d1)) * Si / Q1
        bi = -np.exp((b - r) * T) * norm.cdf(-1 * d1) * (1 - 1 / Q1) - (1 + np.exp((b - r) * T) * norm.cdf(-1 * d1) / (v * (T)**0.5)) / Q1

    return Si
```

5. 結果比較(CRR VS LSMC)：

	CALL(mean)	PUT(mean)
CRR	2.174	4.396
LSMC	2.002	4.246



可以從上圖得出用 CRR 求出的 Call price 到 N=10000 之後，跟使用 LSMC 所算出來的 95%信心水準的上界非常相近。這種相似性可能歸因於以下幾點原因：

首先，CRR 方法在較大的時間步長（N=10000）下可能已經達到了穩定的價格估算。隨著時間步長的增加，CRR 方法對期權價格的估算能力可能得到改善，使其趨近於收斂值。這可能解釋了為什麼在較大的時間步長下，CRR 方法的結果與 LSMC 方法的上界相近。

其次，CRR 方法通常在歐式期權（European options）的定價上表現較為精確。歐式期權具有在到期日才能行使的特性，相較於美式期權，其選擇

權結構較簡單。這使得 CRR 方法能夠更有效地進行估算，並在較大的時間步長下得到準確的結果。

最後，CRR 方法在特定市場模型下可能更加高效，特別是在某些期權屬性相對簡單的情況下。這種高效性可能使得 CRR 方法在較大的時間步長下能夠更迅速而準確地計算期權價格。

（三） 衍伸研究

1、 隱含波動度（包含 Extension works 中的(1)、(2)）

使用台指期 20240102 的市場行情，進而計算隱含波動度

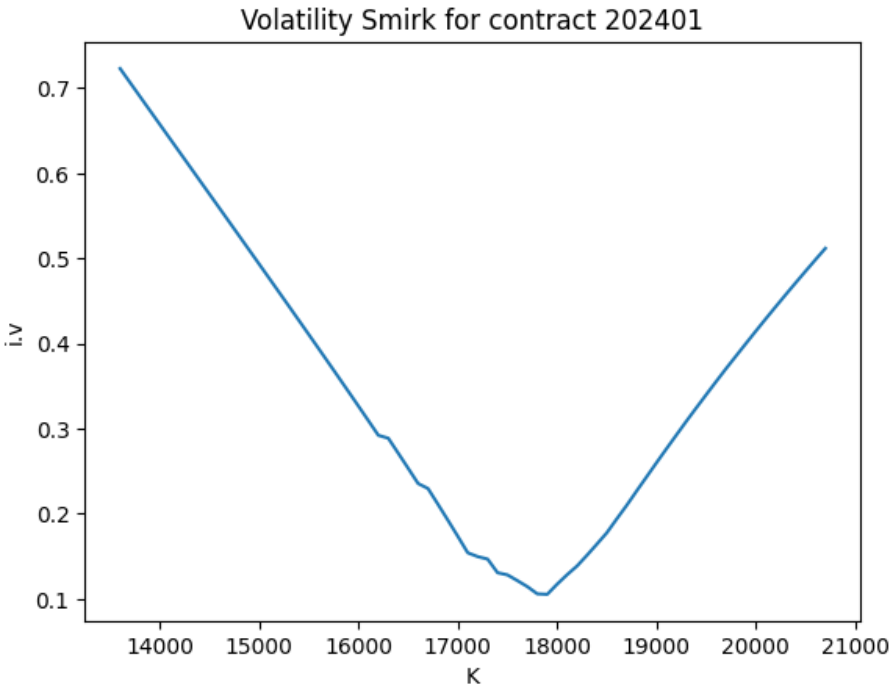
```python
def bs_call(S, K, T, r, vol):
    d1 = (np.log(S/K) + (r + 0.5*vol**2)*T) / (vol*np.sqrt(T))
    d2 = d1 - vol * np.sqrt(T)
    return S * norm.cdf(d1) - np.exp(-r * T) * K * norm.cdf(d2)

def bs_vega(S, K, T, r, sigma):
    d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
    return S * norm.pdf(d1) * np.sqrt(T)

def implied_vol(S0, K, T, r, market_price, tol=0.0001):
    max_iter = 300 #max number of iterations
    vol_old = (np.abs(np.log(S0/(K*np.exp(-1 * r * T))))*(2/T))**0.5 #initial guess

    for k in range(max_iter):
        bs_price = bs_call(S0, K, T, r, vol_old)
        Cprime =  bs_vega(S0, K, T, r, vol_old)*100
        #print(bs_price, market_price)
        C = float(bs_price) - float(market_price)
        vol_new = vol_old - C/Cprime
        #print(k, Cprime, vol_new)
        bs_new = bs_call(S0, K, T, r, vol_new)

        if (abs(vol_old - vol_new) < tol or abs(bs_new - market_price) < tol):
            break

        vol_old = vol_new

    implied_vol = vol_old
    return implied_vol
```

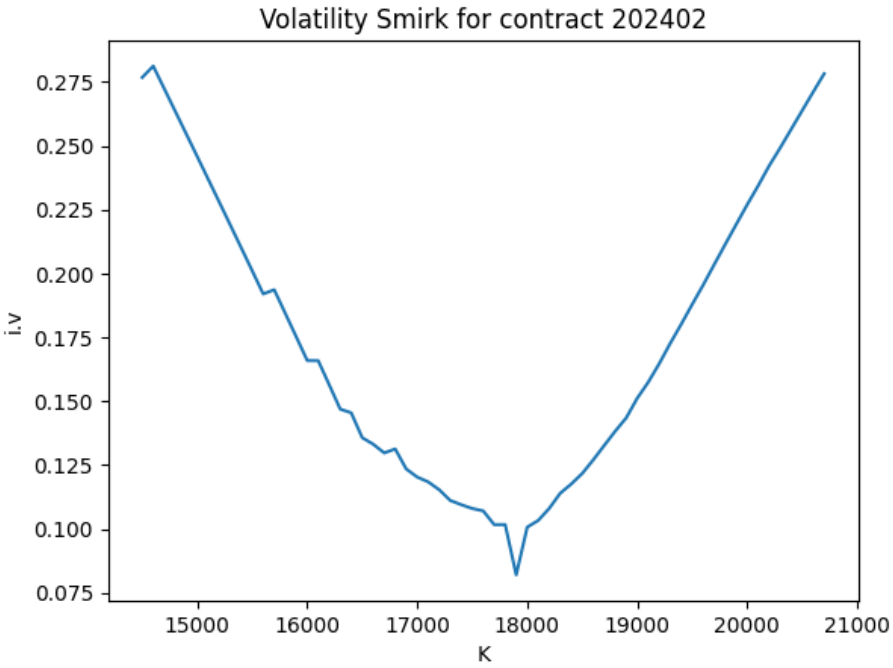
資料：

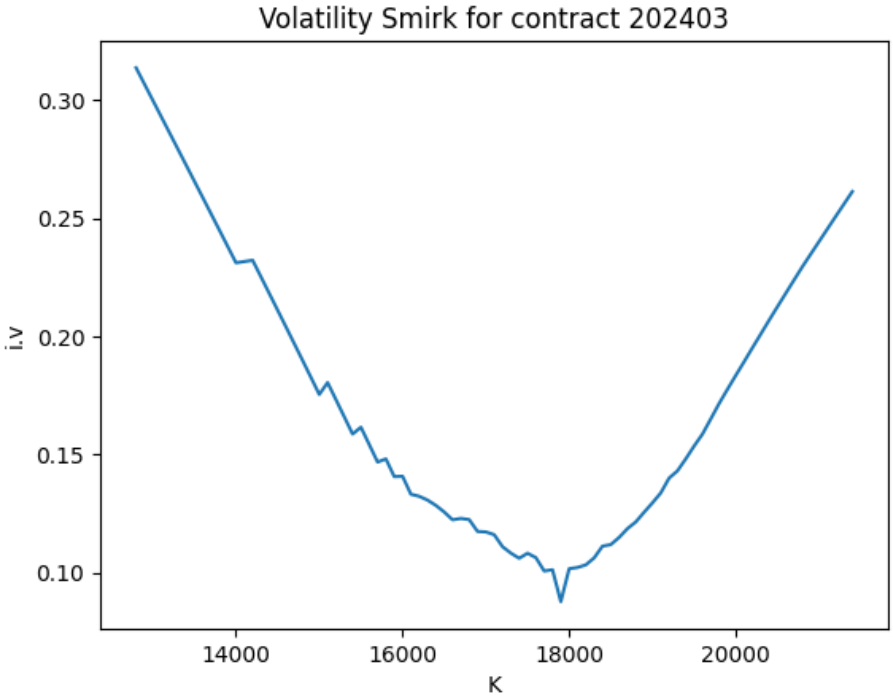
交易日期	契約	到期月份(週	履約價	買賣權	開盤價	最高價	最低價	收盤價	成交量	結算價	未沖銷契	最後最佳	最後最佳	歷史最高	歷史最低	是否因訊	交易時段	漲跌價	漲跌%
2024/1/2	TXO	202401W1	15700	買權	-	-	-	-	0	2130	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15700	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	15700	買權	-	-	-	-	0	0.1	42	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15700	買權	0.1	0.1	0.1	0.1	10	-	-	-	0.9	1.2	0.1		盤後	0	-
2024/1/2	TXO	202401W1	15800	買權	-	-	-	-	0	2030	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15800	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	15800	買權	-	-	-	-	0	0.1	174	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15800	買權	4.8	95	4.8	95	14	-	-	0.1	4.6	139	0.1		盤後	94.9	-
2024/1/2	TXO	202401W1	15900	買權	-	-	-	-	0	1930	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15900	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	15900	買權	-	-	-	-	0	0.1	6	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	15900	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16000	買權	-	-	-	-	0	1830	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16000	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16000	買權	-	-	-	-	0	0.1	13	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16000	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16100	買權	-	-	-	-	0	1730	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16100	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16100	買權	-	-	-	-	0	0.1	8	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16100	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16200	買權	-	-	-	-	0	1630	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16200	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16200	買權	-	-	-	-	0	0.1	10	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16200	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-
2024/1/2	TXO	202401W1	16300	買權	-	-	-	-	0	1530	0	-	-	-	-		一般	-	-
2024/1/2	TXO	202401W1	16300	買權	-	-	-	-	0	-	-	-	-	-	-		盤後	-	-

i. volatility smiles

使用當日不同合約，同一到期日之資料，計算出波動率微笑，並畫成圖。


Volatility Smirk for contract 202401
i.v
K
0.7
0.6
0.5
0.4
0.3
0.2
0.1
14000
15000
16000
17000
18000
19000
20000
21000

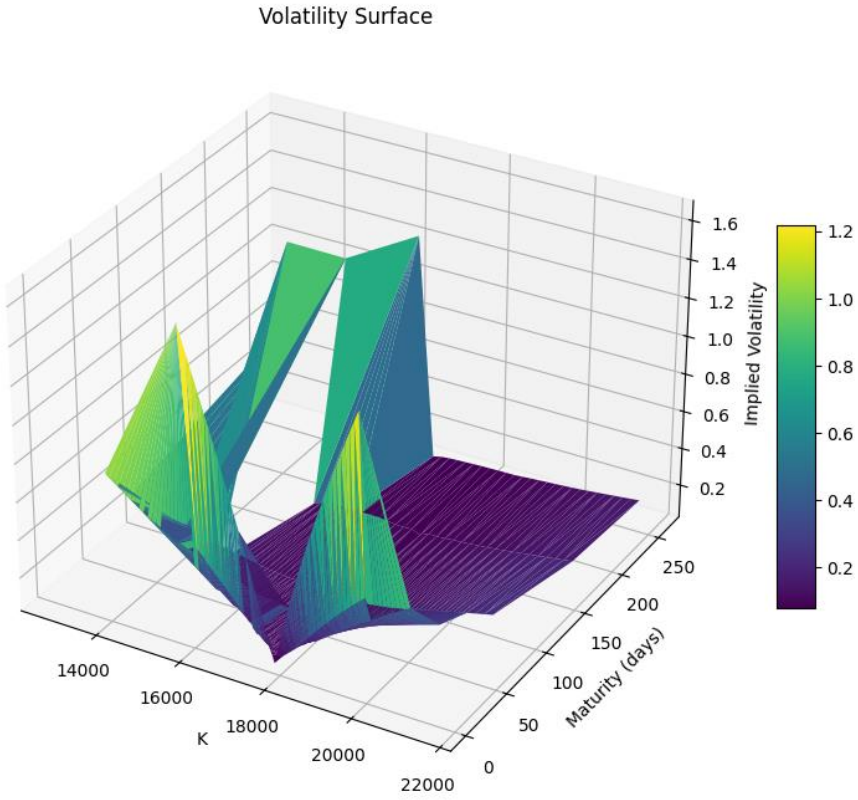

Volatility Smirk for contract 202402
i.v
K
0.275
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
15000
16000
17000
18000
19000
20000
21000



可以從以上圖中看出，當履約價距離當前現貨價格越遠的時候，隱含波動度會越大。

ii. volatility surface

使用當日不同合約，同一到期日之資料，計算出隱含波動度，並畫出波動度曲面。



可以從上圖中看出，距離到期日較近的選擇權，有出現 Volatility Smile，而距離到期日較遠的選擇權，則出現了 Volatility Smirk 。可能原因則是因為市場對於不同時間範圍內的波動性預期存在著變化。對於較短到期日的選擇權，投資者可能面臨即將發生的事件或短期不確定性，因此對未來波動性保持較高的預期，形成了微笑狀。相對地，對於較遠到期日的選擇權，投資者可能認為近期內波動性較低，但對於較長期的未來仍存在較高的不確定性，這可能受到長期內可能發生的事件或宏觀經濟因素的影響。因此，在遠期的到期日上，選擇權的隱含波動度呈現 Volatility Smirk，即在 out of the money 和 in the money 的選擇權上都有所上升。

2、 使用組合數學來解出歐式買賣權的價格（Extension works 中的(6)）

Functions：

```
u = exp(sigma * sqrt(t))
d = exp(-sigma * sqrt(t))
p = (exp((r-q) * t)-d)/(u-d)
```

✓ 0.0s

```
def C(i, j):
    result = factorial(i) // (factorial(j) * factorial(i - j))
    return result
```

✓ 0.0s

Call:

$$v_c = e^{-rT}\Sigma_{j=0}^{n} C_j^n p^{n-j}(1-p)^j max(S_0 u^{n-j} d^j - K, 0)$$

```
sigma = 0
for j in range(n+1):
    temp_part = C(n, j) * p**(n-j) * (1-p)**(j) * max(s0 * u**(n-j) * d**(j) - K, 0)
    sigma += temp_part
call = exp(-r*T) * sigma
```

Put:

$$v_p = e^{-rT}\Sigma_{j=0}^{n} C_j^n p^{n-j}(1-p)^j max(K - S_0 u^{n-j} d^j, 0)$$

```
sigma = 0
for j in range(n+1):
    temp_part = C(n, j) * p**(n-j) * (1-p)**(j) * max(K - s0 * u**(n-j) * d**(j) , 0)
    sigma += temp_part
put = exp(-r*T) * sigma
```

Results:

```
1  print(f'call = {call}, put = {put}')
✓ 0.0s
call = 3.875848325011272, put = 2.3384827361110903
```

3、 使用有限差分法解出歐式買賣權的價格（Extension works 中的(3)）

Functions：

```
def im(S, K, r, q, sigma, T, Smin, Smax, m, n, type):
    omega = 1 if type == 'Call' else -1
    dS = (Smax-Smin)/m
    dt = T/n
    Svec = np.linspace(Smin, Smax, m+1)
    Tvec = np.linspace(0, T, n+1)
    grid = np.zeros(shape=(m+1, n+1))
    # terminal
    grid[:, -1] = np.maximum(omega*(Svec - K), 0)
    # boundary
    tau = Tvec[-1] - Tvec;
    DFq = np.exp(q * tau)
    DFr = np.exp(r * tau)
    # american option boundary condition should differ from Euro style
    grid[0,  :] = np.maximum(omega*(Svec[0]  - K), 0)
    grid[-1, :] = np.maximum(omega*(Svec[-1] - K), 0)
    # set coefficient
    drift = (r-q) * Svec[1:-1]/dS
    diffusion_square = (sigma*Svec[1:-1]/dS)**2
    l = 0.5*(diffusion_square - drift)
    c = -diffusion_square - r
    u = 0.5*(diffusion_square + drift)
    # set matrix (imp)
    A = sp.diags([l[1:], c, u[:-1]], [-1, 0, 1],  format='csc')
    I = sp.eye(m-1)
    M = I - dt * A
    # solve (imp)
    _, M_lower, M_upper = sla.lu(M.toarray())
    for j in reversed(np.arange(n)):
        U = grid[1:-1, j+1].copy()
        U[0] += l[0]*dt*grid[0, j]
        U[-1] += u[-1]*dt*grid[-1, j]
        Ux = sla.solve_triangular( M_lower, U, lower=True )
        grid[1:-1, j] = sla.solve_triangular( M_upper, Ux, lower=False )
    # interpolate
    tck = spi.splrep( Svec, grid[:,0], k=3 )
    res = spi.splev( S, tck )
    return res
```

Results:

使用 implicit finite difference 所算出來的 call 為 3.262。

（四） 附檔說明

檔名	說明
BAW.ipynb	BAW 模型
Fd.ipynb	有限差分法模型
Implied vol.ipynb	計算隱含波動度曲線與平面
LSMC vs CRR.ipynb	比較 LSMC 與 CRR 模型
combi.ipynb	使用組合數學解出選擇權價格
txo.csv	20240102 台指期市場行情