Black-Scholes with Python OOP

Introduction

- 此專案參考具有權威性的Black-Sholes Model來進行實作，透過使用Python Object-Oriented Programing的方式去進行金融模型的模擬與視窗程式實現，此外為了方便測試亦編成一執行檔讓使用者可以在任何環境下皆能執行。

Black-Scholes Model:

- 用來評估歐式期權(European options)價格的數學模型，他基於幾個假設：
    1. 資產價格會根據對數常態分佈。
    2. 市場無交易成本，可以無限制地借貸。
    3. 利率與波動率為恆定。
- 在此模型中期權價格可以透過當前股票價格、執行價格、到期時間、無風險利率以及股票價格波動率來計算。

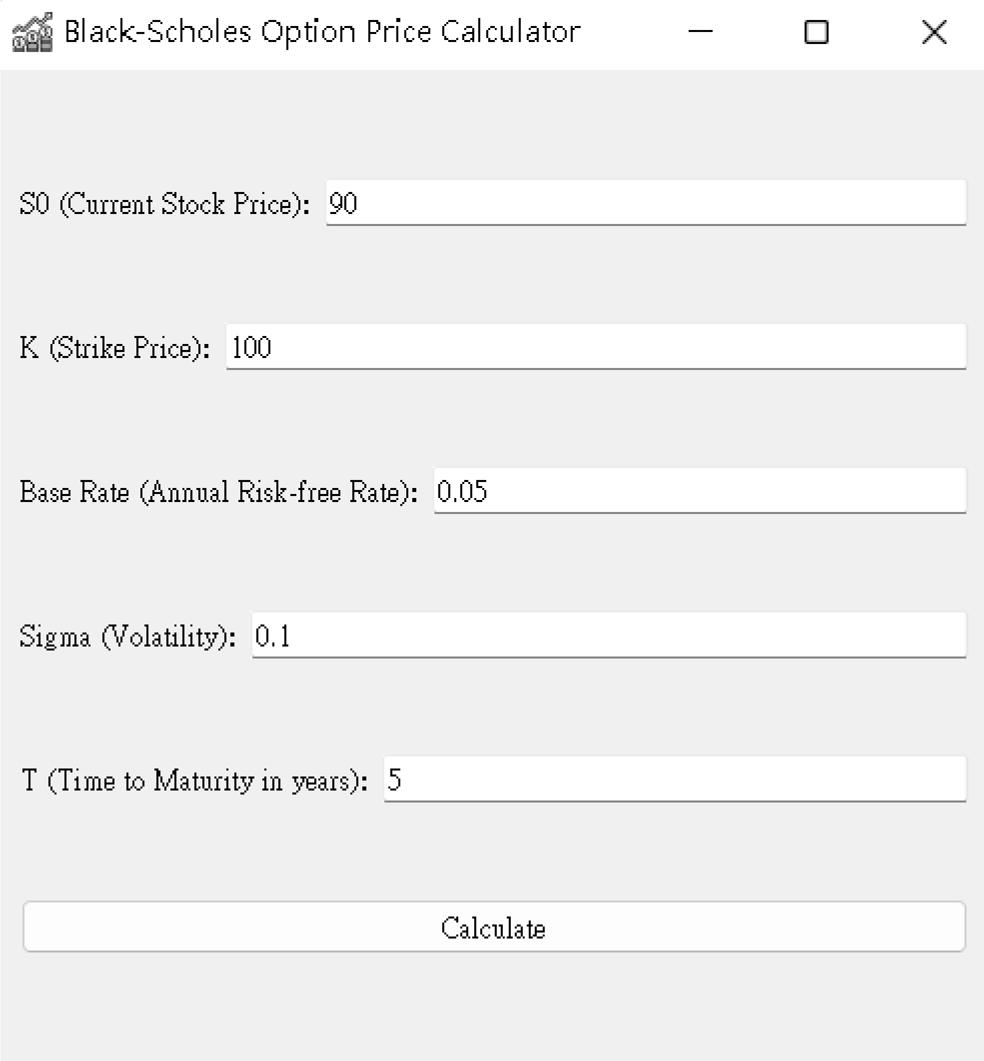
Term Structure

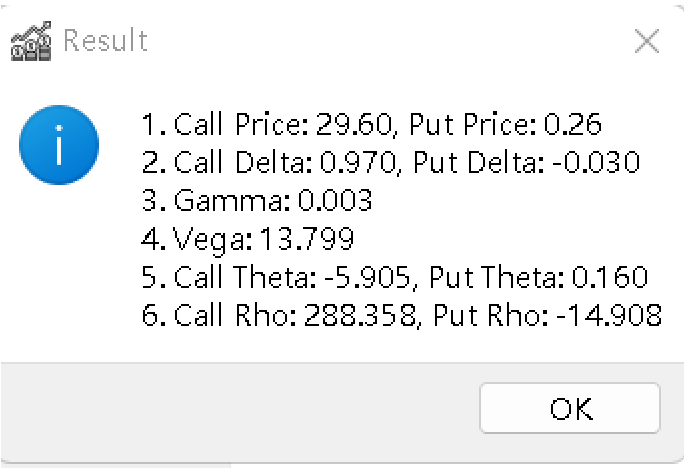
- 期限結通常指的是不同到期時間的債券或無風險利率的結構，它表明了不同時間長度的債券或者貸款的收益率。在Black-Scholes Model中通常使用一個恆定的無風險利率。然而現實中的無風險利率可能隨著時間的不同而變化，這種變化可以通過期限結構來描述。
- 通常使用Term Structure可以使模型更加準確反映當前市場情況。

Greeks

- Greeks用來描述期權價格對市場變量(價格、波動率、時間流逝等)變化的敏感程度，他們事風險管理工具，幫助投資者了解和管理期權持倉的風險。
- 以下是Greeks的解釋：
    - Delta: 表示股票價格變動對期權價值的影響。
    - Gamma: 表示股票價格變動對Delta的影響。
    - Theta: 表示隨時間推移期權價格的變化速度。
    - Vega: 表示波動率對期權價值的影響。
    - Rho: 表示利率變化對期權價值的影響。

Result





Implementation

- Black-Scholes Model:

```
import numpy as np
from scipy.stats import norm

class BlackScholesModel():
    def __init__(self, s0, k, term_structure, sigma, T):
        self.s0 = s0 # init price
        self.k = k  # start price
```

```python
        self.sigma = sigma # volatility of assets price
        self.T = T #years
        self.r = term_structure.get_rate(T)   #term structure, a rate which follow the timeline
        self.d1 = (np.log(s0/k) + (self.r + sigma**2/2)*T) / (sigma * np.sqrt(T))
        self.d2 = self.d1 - sigma * np.sqrt(T)

    def BSPrice(self):  # calculate price
        c = self.s0 * norm.cdf(self.d1) - self.k * np.exp(-self.r * self.T) * norm.cdf(self.d2)
        p = self.k * np.exp(-self.r * self.T) * norm.cdf(-self.d2) - self.s0 * norm.cdf(-self.d1)
        return c, p

# The term "Greeks" refers to the current exposure of an option to different risks.
# The numerical values of Greeks indicate how the price of the option strategy would change when facing various risks.
# There are five types of risks, which are Delta, Gamma, Vega, Theta, and Rho.
# Each represents exposure to price changes, changes in price change, volatility, time, and risk-free interest rate, respectively.


    def BSDelta(self):
        cDelta = norm.cdf(self.d1)
        pDelta = norm.cdf(self.d1) - 1
        return cDelta, pDelta

    def BSGamma(self):
        gamma = norm.pdf(self.d1) / (self.s0 * self.sigma * np.sqrt(self.T))
        return gamma  # Same for call and put

    def BSVega(self):
        vega = self.s0 * np.sqrt(self.T) * norm.pdf(self.d1)
        return vega  # Same for call and put

    def BSTheta(self):
        cTheta = -self.s0 * norm.pdf(self.d1) * self.sigma / (2 * np.sqrt(self.T)) - self.r * self.k * np.exp(-self.r * self.T) * norm.cdf(self.d2)
        pTheta = -self.s0 * norm.pdf(self.d1) * self.sigma / (2 * np.sqrt(self.T)) + self.r * self.k * np.exp(-self.r * self.T) * norm.cdf(-self.d2)
        return cTheta, pTheta

    def BSRho(self):
        cRho = self.k * self.T * np.exp(-self.r * self.T) * norm.cdf(self.d2)
        pRho = -self.k * self.T * np.exp(-self.r * self.T) * norm.cdf(-self.d2)
        return cRho, pRho
```

- Term Structure:

```python
class TermStructure:
    def __init__(self, base_rate):
```

```
        self.base_rate = base_rate

    def get_rate(self, time_to_maturity):
        # This is a simple model that linearly adjusts the base rate based on the
time to maturity.
        return self.base_rate + 0.01 * time_to_maturity
```

- main function:

```
from libs import BSM
from libs import TermStructure
from PyQt5.QtWidgets import QApplication,QWidget,QVBoxLayout, QHBoxLayout, QLabel,
QLineEdit, QPushButton, QMessageBox
from PyQt5.QtGui import QIcon,QImage
import sys

class MyWidget(QWidget):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('Black-Scholes Option Price Calculator')
        self.setGeometry(300, 300, 500, 500)
        self.setWindowIcon(QIcon('logo.ico'))
        self.layout = QVBoxLayout()
        self.setInputField()
        self.setButton()
        self.setLayout(self.layout)

    def setButton(self):
        self.btn = QPushButton('Calculate', self)
        self.btn.clicked.connect(self.calculate)
        self.layout.addWidget(self.btn)
        #self.btn.show()

    def createLineEdit(self, label_text):
        layout = QHBoxLayout()
        label = QLabel(label_text)
        line_edit = QLineEdit()
        layout.addWidget(label)
        layout.addWidget(line_edit)
        return layout, line_edit

    def setInputField(self):
        self.s0_layout, self.s0_edit = self.createLineEdit("S0 (Current Stock
Price):")
        self.k_layout, self.k_edit = self.createLineEdit("K (Strike Price):")
        self.r_layout, self.r_edit = self.createLineEdit("Base Rate (Annual Risk-
free Rate):")
        self.sigma_layout, self.sigma_edit = self.createLineEdit("Sigma
```

```
(Volatility):")
        self.T_layout, self.T_edit = self.createLineEdit("T (Time to Maturity in
years):")
        self.s0_edit.setText('90')
        self.k_edit.setText('100')
        self.r_edit.setText('0.05')
        self.sigma_edit.setText('0.1')
        self.T_edit.setText('5')

        self.layout.addLayout(self.s0_layout)
        self.layout.addLayout(self.k_layout)
        self.layout.addLayout(self.r_layout)
        self.layout.addLayout(self.sigma_layout)
        self.layout.addLayout(self.T_layout)

    def calculate(self):
        try:
            s0 = float(self.s0_edit.text())
            k = float(self.k_edit.text())
            r = float(self.r_edit.text())
            sigma = float(self.sigma_edit.text())
            T = float(self.T_edit.text())

            term_structure = TermStructure.TermStructure(r)
            bsm = BSM.BlackScholesModel(s0, k, term_structure, sigma, T)
            self.price = bsm.BSPrice()
            self.delta = bsm.BSDelta()
            self.gamma = bsm.BSGamma()
            self.vega = bsm.BSVega()
            self.theta = bsm.BSTheta()
            self.rho = bsm.BSRho()

            result_msg = (f"1. Call Price: {self.price[0]:.2f}, Put Price:
{self.price[1]:.2f}\n"
                          f"2. Call Delta: {self.delta[0]:.3f}, Put Delta:
{self.delta[1]:.3f}\n"
                          f"3. Gamma: {self.gamma:.3f}\n"
                          f"4. Vega: {self.vega:.3f}\n"
                          f"5. Call Theta: {self.theta[0]:.3f}, Put Theta:
{self.theta[1]:.3f}\n"
                          f"6. Call Rho: {self.rho[0]:.3f}, Put Rho:
{self.rho[1]:.3f}")
            QMessageBox.information(self, 'Result', result_msg)
            self.terminalShow()
        except ValueError:
            print('Error','Please input valid numbers.')
            QMessageBox.warning(self, 'Error', 'Please input valid numbers.')

    def terminalShow(self):
        print("Call Price:", self.price[0])
        print("Put Price:", self.price[0])

        print('put delta:',self.delta[0])
        print('call delta:',self.delta[1])
```

```
        print('gamma:',self.gamma)

        print("Vega:",self.vega)

        print("call Theta",self.theta[0])
        print("put Theta",self.theta[1])

        print("call Rho",self.rho[0])
        print("put Rho",self.rho[1])

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = MyWidget()
    ex.show()
    sys.exit(app.exec_())
```