Assignment 1

Iydon Liang

September 29, 2019

Contents

1 Question 1

Statement 1

Suppose that for the same asset and expiry date, you hold a European call option with exercise price E_1 and another with exercise price E_3, where $E_3 > E_1$ and also write two calls with exercise price $E_2 := (E_1 + E_3)/2$. Derive a formula for the value at expiry and draw the corresponding payoff diagram. Write a code to verify it.

I use V_h to represent two European call options that we hold, V_w to represent two calls that we write, and S_T to represent the exercise price. Therefore, we have

$$\begin{aligned} V_h &= \max(S_T - E_1, 0) + \max(S_T - E_3, 0) \\ V_w &= E_2 - \max(S_T - E_2, 0). \end{aligned} \tag{1}$$

That is,

$$V = V_h + 2V_w = \begin{cases} E_1 + E_3 & \text{where } S_T \le E_1, \\ S_T + E_3 & \text{where } E_1 < S_T \le E_2, \\ E_1 + 2E_3 - S_T & \text{where } E_2 < S_T \le E_3, \\ E_1 + E_3 & \text{where } E_3 < S_T. \end{cases} \tag{2}$$

The payoff diagram of $E_1 = 2$ and $E_3 = 4$ is Figure 1.

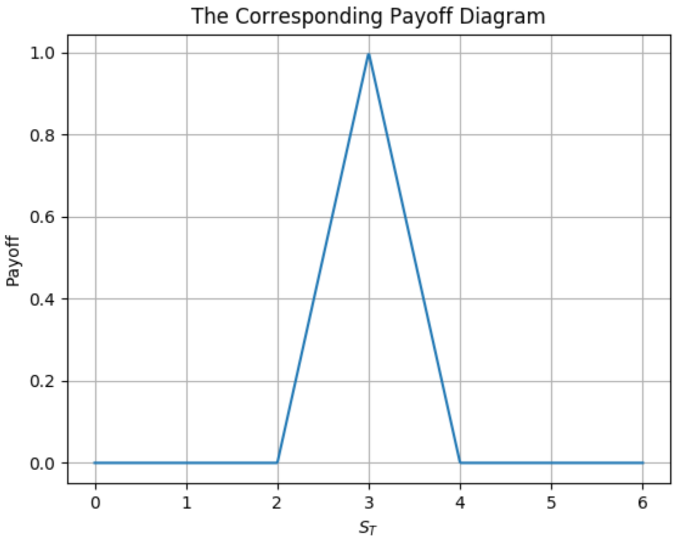

Figure 1: Payoff Diagram

And the code that verifies the result shows below.

Verification

```python
# -*- encode: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt


def question_1(E1, E3, E2=0, number=0, plot=False, figname=''):
    '''Question 1
    Hold two European call option with E1 and E2
    Write two calls with E2
    '''
    # declare variable
    E2 = E2 or (E1+E3)/2
    m, M = 0, E1+E3
    number = number or 100*M # default precision: $0.01
    ST = np.linspace(m, M, number)

```

```python
        # calculate the value at expiry
        value_hold  = (ST-E1).clip(min=0) + (ST-E3).clip(min=0)
        value_write = - (ST-E2).clip(min=0)
        value = value_hold + 2*value_write

        # plot the corresponding payoff diagram
        if plot:
                plt.plot(ST, value)
                plt.xlabel('$S_T$')
                plt.ylabel('Payoff')
                plt.title('The Corresponding Payoff Diagram')
                plt.grid()
                plt.savefig(figname) if figname else plt.show()

        # return value
        return value


if __name__ == '__main__':
        E1, E3 = 2, 4
        figname = '../figures/2019-09-27-payoff-diagram.png'
        value = question_1(E1, E3, plot=True, figname=figname)
```

2 Question 2

Statement 2

Find a systematic way to construct a portfolio of options with arbitrary payoffs.

假设我们希望的收益位于(x, y),如果y小于0,我们过收益点做斜率为$-1, -2, \ldots$的直线，直到直线交x轴于正半轴；如果y大于0，我们过收益点做斜率为$1, 2, \ldots$的直线，直到直线交x轴于正半轴。此时直线斜率为我们需要购买期权的最小数量，而与x轴正半轴所交点即为E值（当然不唯一）。

根据期权收益公式可以有如下代码，将期权的特点抽象出来，收益图展示见图2。

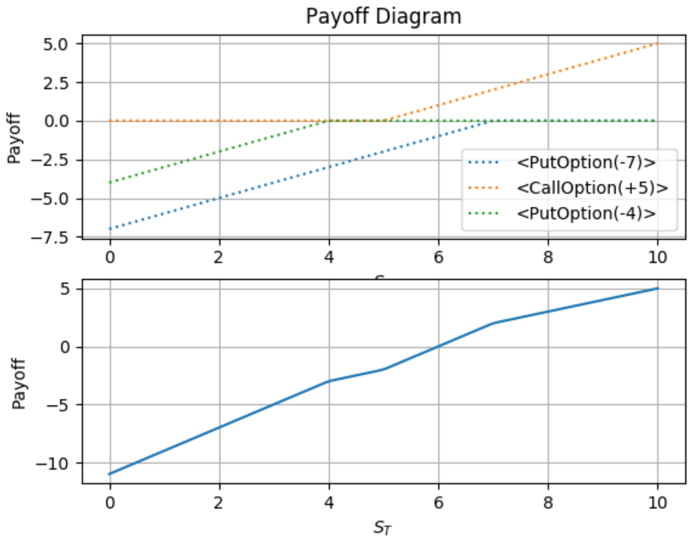

Figure 2: Payoff Diagram

Option API

```python
# -*- encode: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

from random import randint


class _Option:
    def __init__(self, E, volume=1, hold=True):
        '''
        Argument
        =======
            E: int or float, exercise price
            volume: int
            hold: bool
        '''
        self.E = E
        self.volume = volume
        self.hold = hold
        self._type = type(self)

```

```python
    def __repr__(self):
        name = self._type.__name__
        sign = '+' if self.hold else '-'
        value = self.E
        return f'<{name}({sign}{value})>'

    def __mul__(self, value):
        assert isinstance(value, int), 'Argument `value` must be int.'

        if value < 0:
            return self._type(self.E, abs(value)*self.volume, not self.hold)
        return self._type(self.E, value*self.volume, self.hold)

    def __rmul__(self, value):
        return self.__mul__(value)

    def __neg__(self):
        return self._type(self.E, self.volume, not self.hold)

    def value_at(self, S_T):
        '''

        Argument
        =======
            S_T: numpy.ndarray
        '''

        if __debug__:
            assert isinstance(S_T, np.ndarray), 'Type of `S_T` is wrong.'
        return self.value_at__(S_T)
    def value_at__(self, S_T):
        raise NotImplementedError(type(self).__name__)

    def payoff_at(self, S_T):
        '''

        Argument
        =======
```

```python
            S_T: numpy.ndarray
        '''
        if __debug__:
            assert isinstance(S_T, np.ndarray), 'Type of `S_T` is wrong.'
        return self.payoff_at__(S_T)
    def payoff_at__(self, S_T):
        raise NotImplementedError(type(self).__name__)


class CallOption(_Option):
    def value_at__(self, S_T):
        if self.hold:
            return self.volume * (S_T-self.E).clip(min=0)
        else:
            return self.volume * (self.E - (S_T-self.E).clip(min=0))

    def payoff_at__(self, S_T):
        if self.hold:
            return self.volume * (S_T-self.E).clip(min=0)
        else:
            return - self.volume * (S_T-self.E).clip(min=0)


class PutOption(_Option):
    def value_at__(self, S_T):
        if self.hold:
            return self.volume * (self.E-S_T).clip(min=0)
        else:
            return self.volume * (self.E - (self.E-S_T).clip(min=0))

    def payoff_at__(self, S_T):
        if self.hold:
            return self.volume * (self.E-S_T).clip(min=0)
```

```python
            else:
                return - self.volume * (self.E-S_T).clip(min=0)


def Option(E, type_='call', volume=1, hold=True):
    '''Return an option.
    Argument
    =======
        type_: str, type_ in {'call', 'put'}
    '''
    if type_.lower() == 'call':
        return CallOption(E, volume, hold)
    elif type_.lower() == 'put':
        return PutOption(E, volume, hold)
    else:
        raise TypeError(f'Unrecognized type: {type_}')


class Options:
    def __init__(self, *options):
        if __debug__:
            for option in options:
                assert isinstance(option, (CallOption, PutOption))

        self.options = options

    def __repr__(self):
        return '<Options ({})>'.format(len(self.options))

    def plot(self, S_T, type_='value', figname=''):
        '''plot the corresponding payoff diagram

        Argument
        =======
            S_T: np.ndarray
            type_: str, type_ in {'value', 'payoff'}
```

```python
            figname: str
        '''
        if type_.lower() == 'value':
            values = [option.value_at(S_T) for option in self.options]
            yaxis = 'Value at Expiry'
            title = yaxis + ' Diagram'
        elif type_.lower() == 'payoff':
            values = [option.payoff_at(S_T) for option in self.options]
            yaxis = 'Payoff'
            title = yaxis + ' Diagram'
        else:
            raise TypeError(f'Unrecognized type: {type_}')
        legend = [None]*len(self.options)

        fig = plt.figure()
        ax1 = fig.add_subplot(211)
        for i, value in enumerate(values):
            ax1.plot(S_T, value, ':')
            legend[i] = repr(self.options[i])
        ax1.legend(legend)

        ax2 = fig.add_subplot(212)
        ax2.plot(S_T, sum(values))

        ax1.set_title(title)
        ax1.set_xlabel('$S_T$')
        ax2.set_xlabel('$S_T$')
        ax1.set_ylabel(yaxis)
        ax2.set_ylabel(yaxis)
        ax1.grid()
        ax2.grid()
        plt.savefig(figname) if figname else plt.show()


if __name__ == '__main__':
    # variable
    number = 3
    prices = 1, 9
```

```
        figname = '../figures/2019-09-27-payoff-diagram-2.png'
        # code
        options = [None] * number
        for ith in range(number):
            E = randint(*prices)
            type_ = 'call' if randint(0, 1) else 'put'
            hold = randint(0, 1)
            options[ith] = Option(E, type_=type_, hold=hold)

        options = Options(*options)
        S_T = np.linspace(0, sum(prices), 100)
        options.plot(S_T, 'payoff', figname)
```

3 Question 3

Statement 3

Find all types of options in major global trade market.

- 行权时间:
  - 欧式期权，行权时间较固定；
  - 美式期权，行权时间较宽松；
  - 百慕大期权，可以在到期日前所规定的一系列时间行权。
- 期权权利:
  - 看涨期权；看跌期权。
- 合约标的
  - 现货期权（股票、股指、利率、外汇）；商品期权。
- 行权价格与标的证券市价的关系
  - 实值期权；平值期权；虚值期权。