HW5 Report

Kangqi Yu 121090735

December 8, 2023

1 Introduction

In this report, I use the data of S&P500 index call options matured on February 17, 2023 to detect arbitrage opportunities. I will use a optimization model to detect arbitrage opportunities. I will also consider the bid-ask spreads in the market. The report is organized as follows. In Section 2, I will introduce the data. In Section 3, I will introduce the model. In Section 4, I will show the empirical results. In Section 5, I will discuss the results that there are some arbitrage opportunities in this market under my assumptions.

2 Data

The dataset contains the prices (as of November 25, 2022) of S&P500 index call options matured on February 17, 2023 with various strikes. The risk-free rate is taken as the 3-month treasury yield which is 4.1750% and the S&P500 index is 4,026.12 as of November 25, 2022.

3 Model

3.1 Assumption

To simplify the problem, we assume the stock price will only put on the strik prices of options and we ignore the time difference between the maturity (or strike date) of 3-month treasury and options.

3.2 Model

The number of days from November 25, 2022 to February 17, 2023 is $30-25+31+31+17$. The return of treasury is given by $4.1750\% * (30-25+31+31+17)/365$. For such a sequence of call options with strike K_i, the intrisic value of them is

$$V_i = \max\{S_T - K_i, 0\} \tag{1}$$

With the assumptions above, the stock price can only take values among K_i. No arbitrage requires the existence of $y \in \mathbb{R}^n \geq 0$ such that,

$$\begin{cases} 1 = (1+r)\sum_{j=1}^n y_j \\ S_0 = \sum_{j=1}^n y_j K_j \\ C(K_i) = \sum_{j=1}^n y_j \max\{K_j - K_i, 0\} \end{cases} \tag{2}$$

where $C(K_i)$ is the price of call option with strike K_i and S_0 is the initial stock price. The first equation is the risk-free condition, the second equation is the stock price condition and the third equation is the call option price condition.

For $2 \leq l \leq n-1$,

$$C(K_{l+1}) = \sum_{j=1}^n y_j (K_j - K_{l+1})^+ = \sum_{j=l+2}^n y_j (K_j - K_{l+1})$$

$$C(K_l) = \sum_{j=1}^n y_j (K_j - K_l)^+ = \sum_{j=l+1}^n y_j (K_j - K_l)$$

$$C(K_{l-1}) = \sum_{j=1}^n y_j (K_j - K_{l-1})^+ = \sum_{j=l}^n y_j (K_j - K_l)$$

We can draw the monotonicity feature of $C(K_i)$ from the equations above. For $K_i \leq K_j$, $C(K_i) \geq C(K_j)$.

Then, we can take differences,

$$C(K_{l+1}) - C(K_l) = \sum_{j=l+1}^n y_j (K_{l+1} - K_l)$$

$$C(K_l) - C(K_{l-1}) = \sum_{j=l}^n y_j (K_l - K_{l-1})$$

Divide both sides,

$$\frac{C(K_{l+1}) - C(K_l)}{K_{l+1} - K_l} = -\sum_{j=l+1}^{n} y_j$$

$$\frac{C(K_l) - C(K_{l-1})}{K_l - K_{l-1}} = -\sum_{j=l}^{n} y_j$$

Then, convexity and slope bounds arises:

$$-\frac{1}{1+r} \leq \frac{C(K_{l+1}) - C(K_l)}{K_{l+1} - K_l} \leq \frac{C(K_l) - C(K_{l-1})}{K_l - K_{l-1}} \leq 0$$

If we detect arbitrage opportunities, we can solve a optimization problem to get the arbitrage execution plan, when we do not consider the ask-bid spreads: Let p as our price vector (I assume the price of all option is 1), x as the positions we put on each instruments.

$$\min a^\top p$$

$$s.t. a^\top X \geq 0$$

The above model is used to detect arbitrage opportunities without considering the bid-ask spread. The following model will consider it. We should have $c_b \leq c_a$. Then, we have

$$C_b(K_i) \leq C(K_i) \leq C_a(K_i)$$

$$-\frac{1}{1+r} \leq \frac{C(K_{l+1}) - C(K_l)}{K_{l+1} - K_l} \leq \frac{C(K_l) - C(K_{l-1})}{K_l - K_{l-1}} \leq 0$$

If we detect arbitrage opportunities, we can solve a optimization problem to get the arbitrage execution plan, when we consider the ask-bid spreads: let $L_{ij} = \max(K_j - K_i, 0)$,

$$\min c_a^\top x_a - c_b^\top x_b$$

$$s.t. L^\top (x_a - x_b) \geq 0, 0 \leq x_a \leq 1, 0 \leq x_b \leq 1$$

When the optimal value of the above formula is negative, there is an arbitrage opportunities and the position distribution should be x_a and x_b. If we want to do a self-financing arbitrage, we can use risk free asset to fill it.

4 Empirical results

I detect that the case with bid-ask spreads has some arbitrage opportunities. The optimal value of optimization problem is -0.1857. The absolute of it is much higher than the risk free rate, 0.0096, so we can use risk free rate to self-finance. The arbitrage strategy is long 0.4286 options with strike price of 4950, short 1 option with strike price of 5075 and 1 option with strike price of 5100. For the case without bid-ask spreads, there is some arbitrage opportunities too. One of the them is long 0.4286 options with strike price of 4950, short 1 option with strike price of 5075 and 1 option with strike price of 5100, just like the strategy in the first question.

5 Discussions

There are many arbitrage opportunities under my assumptions in this market, whether with bid-ask spreads or not. The reason is that the market is not efficient enough. But, we need to pay attention that my model just assume the prossible stock price just at the strikes prices of options. In fact, the stock price can take any value in the real world. So, the arbitrage opportunities may not exist in the real world. Maybe we can use a continuous distribution to model the prossible stock price to get a statistical arbitrage.

MAT3300 HW5 Code

December 8, 2023

[ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[ ]:
```python
options = pd.read_excel('SP500 Option Prices.xlsx', sheet_name='Sheet1')
options.head()
```

[ ]:

	Strike	Bid price	Mid price	Ask price
0	2400	1633.2	1644.70	1656.2
1	2450	1584.9	1595.95	1607.0
2	2500	1535.8	1547.00	1558.2
3	2550	1486.6	1497.60	1508.6
4	2600	1437.5	1448.70	1459.9

[ ]:
```python
new_data = {'Strike': 0, 'Bid price': 4026.12, 'Mid price': 4026.12, 'Ask␣
↪price': 4026.12}
options.loc[-1] = new_data
```

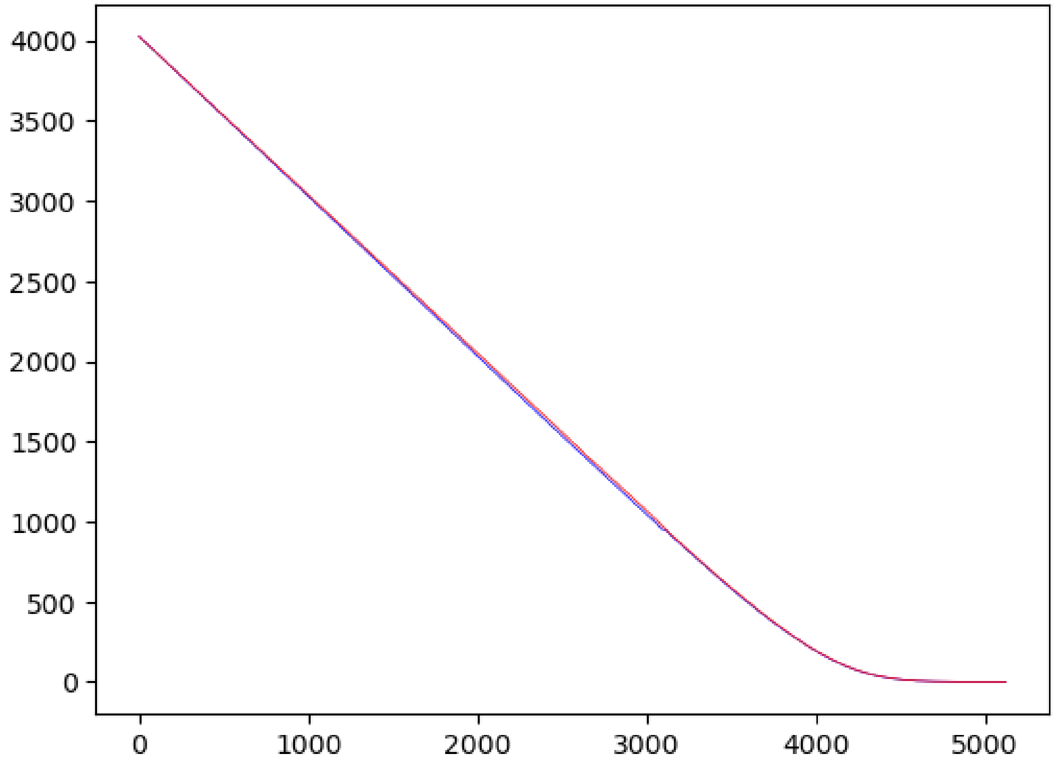
[ ]:
```python
options.sort_index(inplace=True)
options.head()
```

[ ]:

	Strike	Bid price	Mid price	Ask price
-1	0	4026.12	4026.12	4026.12
0	2400	1633.20	1644.70	1656.20
1	2450	1584.90	1595.95	1607.00
2	2500	1535.80	1547.00	1558.20
3	2550	1486.60	1497.60	1508.60

[ ]:
```python
plt.plot(options.loc[:, "Strike"], options.loc[:, "Bid price"], color='blue',␣
↪label='Bid', linewidth=0.5)
plt.plot(options.loc[:, "Strike"], options.loc[:, "Ask price"], color='red',␣
↪label='Ask', linewidth=0.5)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x192d5017eb0>]
```



1 Consider bid-ask spread

[ ]:
```
r=4.175*0.01*(30-25+31+31+17)/365

# For the arbitrage in the case with bid-ask spread, we need to find a set of␣
↪C(K) that satisfy the following conditions:
CK = [4026.12]
slope = []
bound = -1/(1 + r)
temp_slope = bound
flag = False
for i in range(0, len(options) - 1):
    temp = CK[-1] + temp_slope * (options['Strike'][i] - options['Strike'][i -␣
↪1])
    while temp < options['Bid price'][i]:
        temp_slope += 0.000006
        if temp_slope > 0:
            flag = True
            print("slope is positive")
            print("Out of bound")
            break
```

```
        temp = CK[-1] + temp_slope * (options['Strike'][i] -␣
↪options['Strike'][i - 1])
    if flag or temp > options['Ask price'][i]:
        print(str(temp) + '>' + str(options['Ask price'][i]))
        print("Out of bound")
        break
    print(i)
    CK.append(temp)
    print(temp)
    slope.append(temp_slope)

print("Finish")
```

```
0
1648.960271690904
1
1599.4361106844644
2
1549.911949678025
3
1500.3877886715854
4
1450.863627665146
5
1401.3394666587064
6
1351.815305652267
7
1327.0532251490472
8
1302.2911446458274
9
1277.5290641426077
10
1252.766983639388
11
1228.0049031361682
12
1203.2428226329484
13
1178.4807421297287
14
1153.718661626509
15
1128.9565811232892
16
1104.1945006200694
17
```

```
1079.4324201168497
18
1054.67033961363
19
1029.9082591104102
20
1005.1461786071904
21
995.2413464059025
22
985.3365142046146
23
980.3840981039706
24
975.4316820033266
25
965.5268498020387
26
955.6220176007507
27
948.3000053994415
28
940.9779931981323
29
937.3169870974776
30
933.655980996823
926.3339687955138>925.1
Out of bound
Finish
```

[ ]:

```python
# Find arbitrage strategy.

import cvxpy as cp
import numpy as np

n = len(options)
x_a = cp.Variable(n)
x_b = cp.Variable(n)


K = options.loc[:, "Strike"].values

K_i = K[:, None]
K_j = K[None, :]

L = np.maximum(K_j - K_i, 0)
```

```python
c_a = options.loc[:, "Ask price"].values
c_b = options.loc[:, "Bid price"].values

objective = cp.Minimize(cp.sum(cp.multiply(c_a, x_a) - cp.multiply(c_b, x_b)))

constraints = [L.T @ (x_a - x_b) >= 0, x_a >= 0, x_b >= 0, x_a <= 1, x_b <= 1]

problem = cp.Problem(objective, constraints)
problem.solve()

print("Optimal value:", problem.value)
print("Optimal x_a:", x_a.value)
print("Optimal x_b:", x_b.value)
```

```
(CVXPY) Dec 08 10:27:58 PM: Encountered unexpected exception importing solver
OSQP:
ImportError('DLL load failed while importing qdldl:      ')
Optimal value: -0.18571428309261157
Optimal x_a: [4.99955184e-01 3.53870311e-13 3.43010508e-13 3.22590294e-13
 3.27846989e-13 3.09004783e-13 6.17327483e-13 3.04973246e-13
 2.68105581e-13 5.64676872e-13 2.63797190e-13 2.94307208e-13
 2.56041033e-13 2.80041797e-13 2.57616564e-13 2.79226063e-13
 2.54018237e-13 2.81852093e-13 2.63423614e-13 2.95063489e-13
 2.62044060e-13 5.49889540e-13 2.54144450e-13 2.54276698e-13
 5.80127253e-13 2.61265704e-13 5.71139313e-13 2.94982618e-13
 8.83284662e-13 9.80905682e-13 9.00753818e-13 9.00515946e-13
 9.82165209e-13 9.32307238e-13 9.39606971e-13 1.00755393e-12
 1.01498431e-12 9.36796833e-13 1.04617290e-12 1.03931736e-12
 9.74177816e-13 1.00387207e-12 9.90057128e-13 1.11053255e-12
 1.16164758e-12 1.03222803e-12 1.16136875e-12 1.18892375e-12
 1.10502073e-12 1.10300593e-12 1.10338345e-12 1.14743545e-12
 1.15795936e-12 1.21784907e-12 1.32837238e-12 1.19940052e-12
 1.33545193e-12 1.23195426e-12 1.39260454e-12 1.24586658e-12
 1.44876796e-12 1.26820960e-12 1.47988654e-12 1.29706467e-12
 1.29927963e-12 1.56027445e-12 1.37862033e-12 1.52610458e-12
 1.32837853e-12 1.38615776e-12 1.34154595e-12 1.56312403e-12
 1.62720030e-12 1.54841306e-12 1.36371622e-12 1.58907583e-12
 1.42641011e-12 1.63206720e-12 1.39468186e-12 1.57789636e-12
 1.42022227e-12 1.62976426e-12 1.64923752e-12 1.46733865e-12
 1.64200241e-12 1.53108496e-12 1.63146508e-12 1.68052498e-12
 1.66985216e-12 1.39992769e-12 1.61727385e-12 1.43328867e-12
 1.19425999e-12 1.37786737e-12 1.21002828e-12 1.41290121e-12
 1.20379519e-12 1.55013069e-12 1.17610983e-12 1.39718520e-12
 1.42729906e-12 1.68277145e-12 1.18797884e-12 1.44389780e-12
 1.22840941e-12 1.66561737e-12 1.24694632e-12 1.43082921e-12
 1.29118473e-12 1.73195092e-12 1.71178548e-12 1.78951211e-12
```

```
 1.30588660e-12 1.80332207e-12 1.32689194e-12 1.53462112e-12
 1.32160904e-12 1.54583018e-12 1.34405696e-12 1.59480306e-12
 1.59721725e-12 1.60955997e-12 1.36545301e-12 1.66648169e-12
 1.36456909e-12 1.97406587e-12 1.39520478e-12 1.67253004e-12
 1.39992100e-12 1.99587978e-12 2.04200432e-12 1.69864219e-12
 1.45420255e-12 1.74475861e-12 1.44292121e-12 3.76582476e-12
 3.51165149e-12 3.67638870e-12 3.54167781e-12 3.47027468e-12
 3.45484099e-12 3.49445570e-12 2.59128648e-12 4.80711212e-12
 3.30518029e-12 3.25232756e-12 3.25171130e-12 4.57270824e-12
 3.41582912e-12 3.27952251e-12 3.20555495e-12 3.18629406e-12
 3.22032323e-12 3.31190278e-12 3.17324372e-12 4.16080200e-12
 3.35261745e-12 3.38762747e-12 3.18810254e-12 3.33891786e-12
 3.96977006e-12 3.94460374e-12 3.28801272e-12 3.74868221e-12
 3.27276201e-12 3.21435289e-12 3.21732757e-12 3.28256683e-12
 3.12757494e-12 4.06416039e-12 3.28123663e-12 3.31349156e-12
 3.77568410e-12 3.27980290e-12 3.53003098e-12 3.21904033e-12
 3.28923867e-12 3.14000367e-12 3.69066653e-12 3.67591987e-12
 3.40356925e-12 6.10314475e-12 3.48015172e-12 4.34411500e-12
 3.59745266e-12 4.29109251e-12 3.42852174e-12 4.34806078e-12
 5.03944603e-12 4.98965268e-12 4.56238233e-12 4.92554426e-12
 4.60421364e-12 4.31334002e-12 3.31667309e-12 4.41532221e-12
 4.54083477e-12 4.22078122e-12 3.93239112e-12 3.66989061e-12
 4.84314894e-12 4.06895402e-12 4.12487237e-12 4.80559157e-12
 4.29745615e-12 4.98937555e-12 3.90230996e-12 4.12522901e-12
 4.10022028e-12 4.52597951e-12 5.89323372e-12 4.78155089e-12
 4.54196141e-12 4.70665334e-12 4.53520546e-12 4.74325978e-12
 4.89495289e-12 5.82692852e-12 4.94041840e-12 4.86654164e-12
 5.99690581e-12 5.42938230e-12 6.81038074e-12 6.85945335e-12
 5.50064104e-12 5.77010249e-12 6.50985624e-12 6.52488408e-12
 6.16774819e-12 6.66634488e-12 5.61758176e-12 5.98163371e-12
 5.65529857e-12 1.15861372e-11 1.04596483e-11 1.19824420e-11
 1.37241135e-11 1.47364501e-11 1.65232994e-11 1.37780225e-11
 1.47599347e-11 1.70113206e-11 1.75087582e-11 2.47671318e-11
 2.46618307e-11 2.71262510e-11 2.78346408e-11 2.40291526e-11
 3.34915980e-11 5.85810167e-11 4.88705492e-11 2.70502476e-10
 9.50929691e-11 4.28571428e-01 1.52968992e-10 3.62318298e-11
 2.94691111e-11 1.81787813e-11 1.30802872e-11 1.01345724e-11
 4.99999750e-01]
Optimal x_b: [4.99955184e-01 2.57138393e-13 2.82709836e-13 2.91293952e-13
 2.96356460e-13 3.02800056e-13 3.11182906e-13 3.16840338e-13
 3.03701108e-13 3.23696331e-13 3.08512402e-13 3.30593244e-13
 3.18434175e-13 3.32836012e-13 3.15027866e-13 3.33066923e-13
 3.28035237e-13 3.31288633e-13 3.03709559e-13 3.27873397e-13
 3.02331736e-13 3.24324953e-13 3.09698219e-13 3.23058174e-13
 1.80613033e-13 3.00966481e-13 1.80461975e-13 3.25060002e-13
 1.81886310e-12 1.49783248e-12 1.73909312e-12 1.73751228e-12
 1.49298429e-12 1.62596286e-12 1.60306564e-12 1.43690981e-12
 1.36234495e-12 1.60893206e-12 1.36470562e-12 1.37756738e-12
```

1.51075375e-12	1.51115541e-12	1.47448942e-12	1.26885287e-12
1.25343493e-12	1.45165681e-12	1.30561287e-12	1.22391022e-12
1.32948789e-12	1.38799220e-12	1.33092600e-12	1.32299286e-12
1.30845105e-12	1.23903809e-12	1.14010115e-12	1.30709364e-12
1.09537488e-12	1.26767589e-12	1.09315895e-12	1.24923311e-12
1.09431859e-12	1.22581899e-12	1.07484197e-12	1.19604642e-12
1.19180152e-12	1.06483568e-12	1.17005882e-12	1.08033551e-12
1.20409805e-12	1.15947888e-12	1.18932298e-12	1.05610813e-12
1.06089663e-12	1.06172458e-12	1.20975693e-12	1.11173492e-12
1.20544154e-12	1.05421477e-12	1.22665930e-12	1.11475899e-12
1.25273201e-12	1.05355940e-12	1.07989815e-12	1.26603551e-12
1.08265033e-12	1.22335334e-12	1.16530907e-12	1.13986454e-12
1.18615669e-12	1.37776495e-12	1.16911529e-12	1.40370966e-12
1.72649989e-12	1.45985552e-12	1.69838001e-12	1.48480978e-12
1.70979746e-12	1.29928929e-12	1.76350099e-12	1.49870346e-12
1.46423383e-12	1.30809288e-12	1.82769808e-12	1.51018814e-12
1.83589581e-12	1.37258256e-12	1.89401749e-12	1.59414642e-12
1.80373460e-12	1.27228332e-12	1.39554473e-12	1.28926267e-12
1.86843915e-12	1.28026228e-12	1.82731488e-12	1.61883856e-12
1.93676609e-12	1.60476105e-12	1.89120997e-12	1.62862171e-12
1.62533553e-12	1.61182758e-12	1.95168774e-12	1.55591406e-12
1.95064859e-12	1.34528247e-12	2.00033863e-12	1.70586366e-12
1.98831819e-12	1.45347419e-12	1.42756646e-12	1.76339667e-12
1.99655605e-12	1.80514357e-12	2.13447641e-12	2.97107060e-12
4.26571497e-12	2.79799369e-12	4.22118699e-12	2.92656108e-12
2.93713796e-12	2.90909698e-12	4.14349951e-12	2.74620721e-12
4.09045582e-12	2.86052123e-12	4.17882871e-12	2.82152968e-12
4.42090759e-12	2.83928743e-12	2.89661631e-12	2.91221183e-12
4.22695644e-12	2.81480518e-12	4.31621071e-12	2.76693003e-12
4.00612005e-12	2.98285236e-12	4.28825684e-12	3.02054407e-12
2.64544519e-12	2.86220985e-12	4.10629738e-12	2.55906716e-12
4.12748328e-12	3.12863046e-12	4.21954306e-12	3.06676023e-12
3.90157850e-12	2.80885991e-12	2.83380864e-12	3.04137151e-12
5.02989271e-12	3.07056520e-12	4.77986465e-12	2.88306456e-12
5.34846386e-12	2.94738816e-12	5.18838989e-12	2.79535571e-12
2.74924699e-12	3.37825163e-12	5.72535727e-12	3.80292481e-12
6.49810030e-12	3.47272012e-12	5.91669685e-12	3.43150122e-12
3.74996443e-12	3.77390099e-12	3.63684245e-12	3.43620955e-12
3.60919220e-12	3.81193706e-12	3.63156469e-12	3.73657751e-12
4.61201984e-12	4.36601401e-12	4.16484115e-12	3.60163953e-12
4.33440646e-12	4.53642001e-12	4.46735146e-12	4.36049034e-12
4.27838406e-12	4.22136413e-12	4.19037177e-12	4.45925819e-12
5.16295610e-12	4.60399193e-12	4.77472675e-12	4.36431314e-12
4.58190229e-12	4.42480651e-12	5.30139221e-12	4.39046506e-12
4.86786027e-12	4.79653580e-12	5.62179152e-12	5.72176436e-12
5.43668338e-12	6.00015586e-12	5.75757302e-12	5.71920224e-12
5.89714710e-12	6.76581357e-12	5.96940109e-12	7.27415760e-12
6.27344681e-12	4.97712750e-12	6.96358597e-12	8.08479570e-12

```
6.89711140e-12 1.20079509e-11 1.34824686e-11 1.15795845e-11
1.03129816e-11 1.36731874e-11 1.24223231e-11 1.44700741e-11
1.34729253e-11 1.82495705e-11 1.76626073e-11 2.18444304e-11
2.19025091e-11 2.79551503e-11 2.72147201e-11 2.22989309e-11
2.32550058e-11 2.36920768e-11 1.88835640e-11 1.77492932e-11
1.57798680e-11 1.75589078e-11 1.96778132e-11 3.14351283e-11
4.00910437e-11 3.62959832e-10 1.00000000e+00 1.00000000e+00
4.99999750e-01]
```

2 Not consider bid-ask spread

[ ]:
```
# As the mid price is in the middle of bid-ask spreads, we are easy to
↪construct arbitrage strategy than the case with bid-ask spread.
n = len(options) + 1
x = cp.Variable(n)

K = options.loc[:, "Strike"].values

K_i = K[:, None]
K_j = K[None, :]

L = np.maximum(K_j - K_i, 0)
risk_free = (1 + r)*np.ones((1, n - 1))
L = np.concatenate((L, risk_free), axis=0)

p = options.loc[:, "Mid price"].values
p = np.append(p, 1)

objective = cp.Minimize(cp.sum(cp.multiply(p, x)))

constraints = [x.T @ L >= 0, x <= 1, x >= -1]

problem = cp.Problem(objective, constraints)
problem.solve()

print("Optimal value:", problem.value)
print("Optimal x:", x.value)
```

```
Optimal value: -49.66610136640696
Optimal x: [-4.20669955e-04  9.99999999e-01 -4.99369057e-01 -9.99999998e-01
  4.99789788e-01 -9.99999998e-01  1.00000000e+00  4.99999925e-01
 -9.99999999e-01  1.00000000e+00 -9.99999997e-01  9.99999998e-01
 -9.99999999e-01  9.99999993e-01 -9.99999997e-01  9.99999991e-01
 -9.99999999e-01  8.00000015e-01 -5.99999998e-01  9.99999998e-01
 -9.99999992e-01  1.00000000e+00 -9.99999999e-01 -9.99999999e-01
  1.00000000e+00 -3.00000010e-01  1.00000000e+00  9.99999999e-01
 -9.99999997e-01  9.99999998e-01 -9.99999996e-01 -9.99999996e-01
```

9.99999997e-01	-9.99999995e-01	-9.99999996e-01	9.99999993e-01
9.99999996e-01	-9.99999998e-01	9.99999761e-01	9.50000276e-01
-9.99999998e-01	-9.99999997e-01	-9.99999997e-01	9.99999998e-01
9.99999998e-01	-9.99999997e-01	9.99999995e-01	9.99999998e-01
4.39999931e-01	-9.99999993e-01	-9.99999987e-01	-2.89999999e-01
-9.99999980e-01	9.99999993e-01	9.99999998e-01	-9.99999996e-01
9.99999998e-01	-9.99999997e-01	9.99999998e-01	-9.99999995e-01
9.99999999e-01	-9.99999981e-01	9.99999999e-01	-2.50000055e-01
-9.99999991e-01	9.99999999e-01	2.25000024e-01	9.99999998e-01
-9.99999997e-01	-9.99999994e-01	-9.99999998e-01	9.99999985e-01
9.99999847e-01	-8.92839760e-03	-9.99999998e-01	9.99999994e-01
-9.99999996e-01	9.99999998e-01	-9.99999998e-01	9.99999991e-01
-9.99999999e-01	5.33928567e-01	8.33333327e-01	-9.99999998e-01
9.99999996e-01	-9.99999997e-01	-8.33333319e-01	9.99999989e-01
9.99999987e-01	-9.99999997e-01	9.99999999e-01	-7.62499963e-01
-9.99999999e-01	9.99999966e-01	-9.99999999e-01	9.99999994e-01
-9.99999998e-01	9.99999999e-01	-9.99999999e-01	-1.37499922e-01
9.99999939e-01	9.99999999e-01	-9.99999999e-01	9.99999990e-01
-9.99999999e-01	9.99999999e-01	-9.99999999e-01	-9.99999996e-01
-9.99999999e-01	9.99999998e-01	9.73940069e-01	9.99999998e-01
-9.99999999e-01	9.99999999e-01	-9.99999999e-01	-1.47880124e-01
-9.99999999e-01	9.99999990e-01	-9.99999999e-01	9.99999987e-01
7.39400707e-02	5.66666684e-01	-9.99999999e-01	9.99999997e-01
-9.99999999e-01	9.99999999e-01	-9.99999999e-01	7.33333293e-01
-9.99999999e-01	9.99999998e-01	9.99999999e-01	-9.99999961e-01
-9.99999998e-01	9.99999991e-01	-9.99999999e-01	9.99999998e-01
-9.99999982e-01	9.99999998e-01	-9.99999996e-01	8.26478755e-01
-2.52957544e-01	6.26478774e-01	-9.99999999e-01	9.99999998e-01
-9.99999997e-01	9.99999989e-01	-9.99999997e-01	9.99999998e-01
-9.99999998e-01	6.65597127e-01	-3.31194248e-01	6.65597180e-01
-9.99999997e-01	9.99999997e-01	-9.99999994e-01	9.99999999e-01
-6.66666827e-01	9.99999996e-01	-9.99999997e-01	-8.33333230e-01
9.99999995e-01	5.00000037e-01	-9.99999998e-01	9.99999998e-01
-9.99999997e-01	9.99999970e-01	-9.99999997e-01	9.99999904e-01
-9.99999998e-01	9.99999992e-01	-9.99999806e-01	9.99999900e-01
-9.99999997e-01	9.99999993e-01	-9.99999996e-01	9.99999996e-01
-9.99999998e-01	9.99999983e-01	-9.99999998e-01	9.99999983e-01
-9.99999942e-01	9.99999994e-01	-9.99999999e-01	9.99999966e-01
-9.99999998e-01	9.99999994e-01	-9.99999998e-01	5.00000014e-01
-3.26085501e-10	2.97787402e-01	-4.46681106e-01	1.48893705e-01
-9.88314063e-10	3.33333328e-01	-9.99999994e-01	9.99999986e-01
-4.99999979e-01	1.66666662e-01	-7.23880873e-09	1.64645331e-09
7.25757873e-01	-9.99999984e-01	4.84842347e-02	4.28294392e-01
-4.05073028e-01	2.75342081e-01	-2.18416697e-01	8.12277791e-01
-9.99999988e-01	3.33333324e-01	5.77297217e-09	7.38907322e-02
-2.21672205e-01	8.14448133e-01	-9.99999992e-01	5.00127446e-01
-3.33588233e-01	5.68149478e-01	-4.95319174e-01	-6.14783092e-01
9.99999992e-01	-5.82506173e-01	2.91253087e-01	1.86112295e-01

```
 1.27775402e-01 -9.99999984e-01  8.24214485e-01 -1.51188875e-01
-1.11928844e-01  8.52927668e-01 -9.99999977e-01 -7.89157635e-01
 9.99999968e-01  2.83759072e-01 -4.45027148e-01  2.22513573e-01
 1.26278996e-09 -1.02022332e-09  1.56200561e-09 -1.33980485e-09
 1.40957833e-09 -1.01551942e-09 -2.34387720e-11  4.39120533e-10
 2.06834792e-09 -2.37499875e-09  8.27518873e-10  2.06149381e-09
-5.07440169e-09  4.99999971e-01 -9.99999935e-01  4.99999997e-01
 2.50000001e-01  9.99999981e-01  9.99999962e-01 -9.99999992e-01
-9.99999994e-01 -9.99999997e-01 -9.99999998e-01 -9.99999998e-01
 0.00000000e+00  9.99999723e-01]
```