Open in app

Sarit Maitra

1.4K Followers About

How to Solve Optimization Problem Using Convex Mathematical Optimization

Mathematical Optimization for Portfolio Management

Sarit Maitra Just now · 4 min read ★

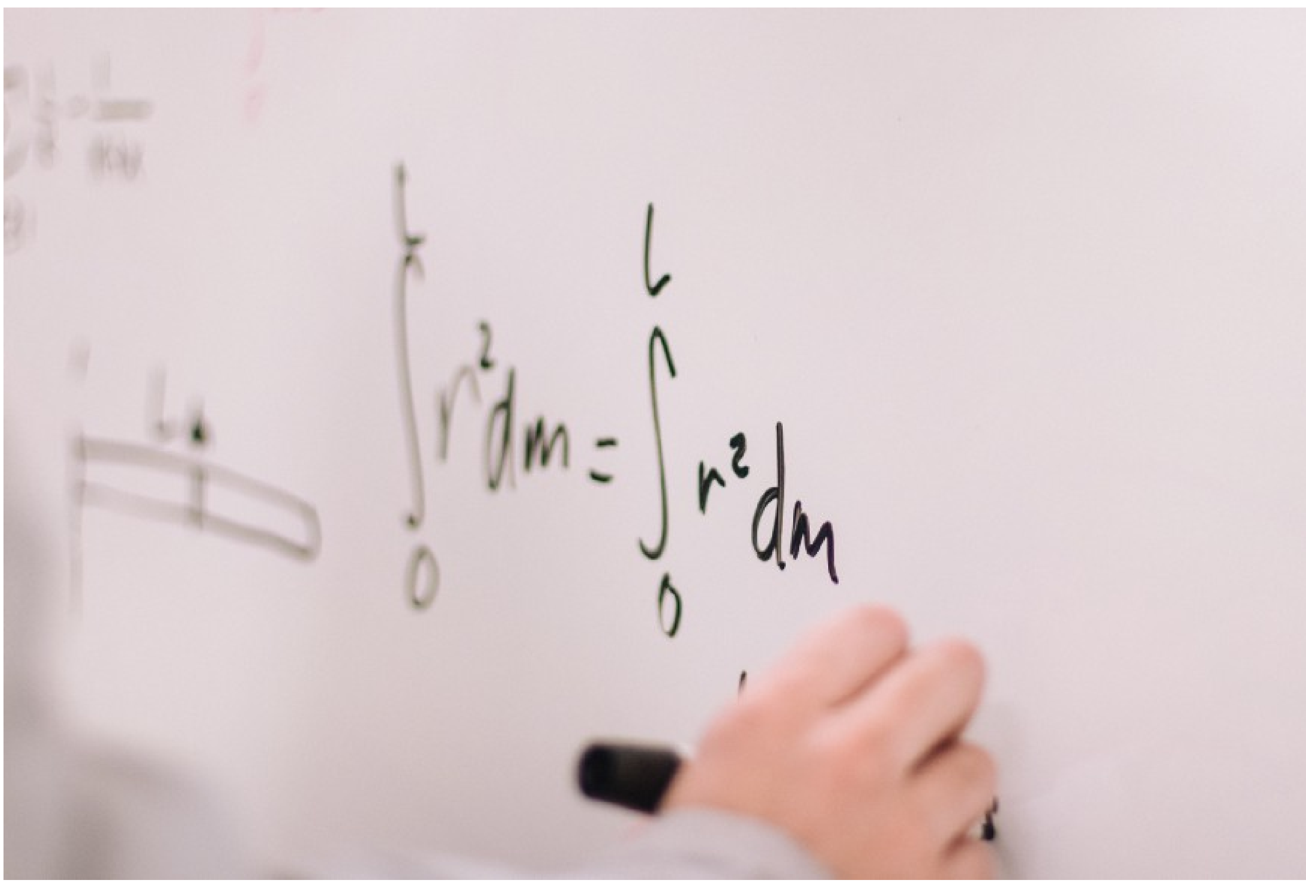

Photo by Jeswin Thomas on Unsplash

O zeroed down on the problem statement, the next step is to solve the problem with the best available options.

To simplify, the idea is to find the best available solution which is at least as good and any other possible solution. If we want to quantify and express the problem in mathematics, we need to come with an objective of solving the problem which is the objective function in mathematics. This will be followed by condition or conditions of the problem which are constraints.

Let us load the data. Data is taken from Stanford and covers 10 years, from January 2006 to December 2016, and comprises a set of 52 popular exchange traded funds (ETFs) and the US central bank (FED) rate of return. We have taken the daily close prices.

```
df = pd.read_csv("prices.txt", parse_dates = True, index_col=0)
df.info()
```

```
28  KRE    2521 non-null   float64
29  LQD    2521 non-null   float64
30  OIL    2521 non-null   float64
31  SDS    2521 non-null   float64
32  SH     2521 non-null   float64
33  SLV    2521 non-null   float64
34  SPY    2521 non-null   float64
35  USO    2521 non-null   float64
36  VGK    2521 non-null   float64
37  VNQ    2521 non-null   float64
38  VTI    2521 non-null   float64
39  VWO    2521 non-null   float64
40  XHB    2521 non-null   float64
41  XLB    2521 non-null   float64
42  XLE    2521 non-null   float64
43  XLF    2521 non-null   float64
44  XLI    2521 non-null   float64
45  XLK    2521 non-null   float64
46  XLP    2521 non-null   float64
47  XLU    2521 non-null   float64
48  XLV    2521 non-null   float64
49  XLY    2521 non-null   float64
50  XME    2521 non-null   float64
```

```
memory usage: 1.0 MB
```

Problem formulation:

So, here our objective is to find the optimal portfolio allocation. Our problem here is maximization problem and the solution is likely by maximizing the expected portfolio fractional return and minimizing the portfolio standard deviation.

> Nothing in the world takes place without optimization, and there is no doubt that all aspects of the world that have a rational basis can be explained by optimization methods. .....Leonhard Euler(1744)

Moreover, we have a quadratic programming problem with a quadratic objective and affine equality and inequality constraints. So, our problem with *n* variables and *m* constraints and this can be formulated as follows.

$$\underset{x}{maximize}\ \mu^t x - \frac{1}{2} x^T \Sigma x$$

(**x**T denotes the vector auto response of x)

subject to

$$x \preccurlyeq b$$
$$x \geq 0$$

(it is the portfolio allocation vector where xi is the number of shares in asset i to buy. We have kept x ≥ 0 to prevent shorting).

Similarly, as equivalent optimization problem is to set an upper bound σ2σ2 on the portfolio variance and maximize the expected portfolio fractional return.

subject to

$$x \preccurlyeq b$$

$$x \geq 0$$

$$x^T \sum x \leq \sigma^2$$

Let us do some initial pre-processing before we start working on the data.

```
data = df.reset_index()
data.rename(columns = {'index': 'date'}, inplace=True)

# last price for each fund
last_price = data.drop(['date'], axis=1)
last_price = last_price.tail(1).to_numpy()
pd.DataFrame(last_price)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	108.09	15.61	53.56	192.49	49.36	35.6	57.66	25.56	20.83	23.25	30.9	30.67	44.34	53.79	32.

We have taken the return data set to calculate expected return and create covariance matrix.

```
# weekly returns of last 1 year
week_ret = data[(data['date'].dt.weekday == 4) & (data['date'] >= '2016-01-01')]
week_ret = week_ret.drop(['date'], 1)
week_ret = week_ret.pct_change().dropna()

# expected return and covariance matrix
sigma = week_ret.cov().to_numpy()
mu = week_ret.mean().to_numpy()
```

```
sigma
```

```
array([[ 2.35881215e-05, -5.33718342e-06, -1.69862970e-06, ...,
         1.15603102e-06, -5.41856358e-05, -1.65060865e-08],
       [-5.33718342e-06,  6.17014992e-04,  2.06275660e-04, ...,
         5.32059114e-04,  8.77476869e-04, -1.01309598e-08],
       [-1.69862970e-06,  2.06275660e-04,  6.11676262e-04, ...,
         7.26300465e-04,  5.57369061e-04,  3.62049739e-08],
       ...,
       [ 1.15603102e-06,  5.32059114e-04,  7.26300465e-04, ...,
```

```
       7.90551336e-08,  6.93593116e-08,  1.60691771e-10]])
```

```
mu
```

```
array([-1.16825552e-04,  4.61282409e-03, -3.06933991e-05,  3.61370837e-03,
        2.06857966e-03,  4.13533580e-03,  9.07251461e-04,  6.67517418e-04,
        2.79534200e-03,  1.92186528e-02,  2.76770553e-02,  5.19993616e-04,
       -4.13223744e-04,  3.14698387e-03,  1.23578573e-02,  2.15335908e-04,
       -1.94364742e-04,  4.00486600e-03,  1.06906789e-02,  1.55097271e-03,
```

Solution to optimization:

We have the constant, parameters (maximum allocation in one fund which is calculated based on maximum std. deviation), number of funds as in variables. Our objective function is the maximization of the the return with the constraints: portfolio valuation should not be negative and volatility (std. deviation) of a fund at a given point of time should be ≤ maximum deviation.

Let us formulate our solution based on the hypotheses and work towards an feasible solution.

```
# optimization variable and parameters
x = cp.Variable(shape = df.shape[1], integer=True)
threshold = cp.Parameter(nonneg=True) # maximum portfolio variance
k = cp.Parameter(nonneg=True) # maximum allocation into one fund
# portfolio mean and variance
mean = mu.T*x
variance = cp.quad_form(x, sigma)
objective = cp.Maximize(mean) # objective function
# constraints
constraints = [x >= 0, variance <= threshold]
for pi in last_price:
    constraints = constraints + [pi*x <= k] # upper bound on single-fund allocation
prob = cp.Problem(objective, constraints)
# Solving optimization problem for each parameter combination
z_values = []
k_values = np.arange(1000, 5000, 1000)
threshold_values = np.arange(1, 5.5, 0.5)
for threshold_value in threshold_values:
    for k_value in k_values:
        threshold.value = threshold_value
        k.value = k_value
        prob.solve()
        if prob.status != 'optimal': continue
        counts = x.value.round()
        investments = last_price*counts
        returns = mu@investments[0]
        z_values.append(returns)
# The optimal objective
```

Open in app

```
status: optimal
optimal value 0.9822826057876909
```

```
optimal_funds = (df.columns).values[np.where(counts > 0)]
print('Funds=>', optimal_funds);
counts_optimal = counts[counts > 0]
print('Counts=>', counts_optimal);
prices_optimal = np.around(last_price, 2)[0][np.where(counts > 0)]
print('Prices=>', prices_optimal);
investments_optimal = np.around(investments, 2)[investments > 0]
print('Investments=>', investments_optimal); print()
capital_optimal = np.around(counts_optimal@prices_optimal, 2)
print('Capital=>', capital_optimal);
risk_optimal = np.around(counts.T@sigma@counts, 2)
print('Risk=>', risk_optimal)
return_optimal = np.around(52*returns/capital_optimal, 3)
print('Return=>', return_optimal); print()
```

```
Funds=> ['DBC' 'EWT' 'GDX' 'KBE' 'SH' 'XLE' 'XLK' 'XME']
Counts=> [ 1.  7.  3.  1.  4. 25. 19. 22.]
Prices=> [15.61 30.9  21.2  42.76 37.1  75.35 47.28 33.18]
Investments=> [  15.61  216.3    63.6    42.76  148.4  1883.75  898.32  729.96]

Capital=> 3998.7
Risk=> 4.99
Return=> 0.519
```

```
Funds = pd.DataFrame(optimal_funds)
Funds.rename(columns = {0: 'PortfolioName'}, inplace=True)
Counts = pd.DataFrame(counts_optimal)
Counts.rename(columns = {0: 'Counts'}, inplace=True)
Prices = pd.DataFrame(prices_optimal)
Prices.rename(columns = {0: 'OptimalPrices'}, inplace=True)
Investments = pd.DataFrame(investments_optimal)
Investments.rename(columns = {0: 'OptimalInvestments'}, inplace=True)
pd.concat([Funds, Counts, Prices, Investments], 1)
```

	PortfolioName	Counts	OptimalPrices	OptimalInvestments
0	DBC	1.0	15.61	15.61
1	EWT	7.0	30.90	216.30
2	GDX	3.0	21.20	63.60

5	XLE	25.0	75.35	1883.75
6	XLK	19.0	47.28	898.32
7	XME	22.0	33.18	729.96

So, we can make an inference from above that, XLE (energy sector), XLK (technology sector) and XME (metal & mining) are dominating the portfolio with > 80% of total investments. However, it can be noted that, the expected return is based on past prices which is not a reasonable indicator of future performance.

Key takeaways:

The above use case was a mixed-integer quadratic programming problem. Though I have used financial/stock data, but similar approach can be used in many other applications. In general, there are multiple solutions with an optimum objective value, but usually the aim is to find just one of them.

I can be reached here.

Optimization · Mathematical Modeling · Industry Solutions · Quadratic Equation