a2_2

October 13, 2025

1 Exercises - Constrained Optimization

1.1 Data

All the analysis below applies to the data set, * `data/spx_weekly_returns.xlsx` * The file has **weekly** returns. * For annualization, use 52 periods per year.

Consider only the following 10 stocks…

[1]:
```
TICKS =  ['AAPL','NVDA','MSFT','GOOGL','AMZN','META','TSLA','AVGO','BRK/
 ↪B','LLY']
```

As well as the ETF,

[2]:
```
TICK_ETF = 'SPY'
```

1.1.1 Data Processing

[3]:
```
import pandas as pd
```

[4]:
```
INFILE = '../data/spx_returns_weekly.xlsx'
SHEET_INFO = 's&p500 names'
SHEET_RETURNS = 's&p500 rets'
SHEET_BENCH = 'benchmark rets'
```

[5]:
```
info = pd.read_excel(INFILE,sheet_name=SHEET_INFO)
info.set_index('ticker',inplace=True)
info.loc[TICKS]
```

[5]:

ticker	name	mkt cap
AAPL	Apple Inc	3.008822e+12
NVDA	NVIDIA Corp	3.480172e+12
MSFT	Microsoft Corp	3.513735e+12
GOOGL	Alphabet Inc	2.145918e+12
AMZN	Amazon.com Inc	2.303536e+12
META	Meta Platforms Inc	1.745094e+12
TSLA	Tesla Inc	9.939227e+11
AVGO	Broadcom Inc	1.148592e+12

```
BRK/B   Berkshire Hathaway Inc  1.064240e+12
LLY              Eli Lilly & Co  7.332726e+11
```

[6]:
```
rets = pd.read_excel(INFILE,sheet_name=SHEET_RETURNS)
rets.set_index('date',inplace=True)
rets = rets[TICKS]
```

[7]:
```
bench = pd.read_excel(INFILE,sheet_name=SHEET_BENCH)
bench.set_index('date',inplace=True)
rets[TICK_ETF] = bench[TICK_ETF]
```

[8]:
```
bench.head()
```

[8]:
```
                 SPY       BTC       USO       TLT       IEF       IYR  \
date
2015-01-09 -0.005744 -0.079179 -0.080945  0.029453  0.013517  0.029953
2015-01-16 -0.012827 -0.281115  0.002735  0.016175  0.010188  0.019471
2015-01-23  0.016565  0.137612 -0.072559  0.011863  0.001558  0.007958
2015-01-30 -0.026931 -0.030969  0.048235  0.026044  0.011992 -0.013361
2015-02-06  0.030584 -0.027431  0.092593 -0.051020 -0.022724 -0.013173

                 GLD
date
2015-01-09  0.027875
2015-01-16  0.044858
2015-01-23  0.013957
2015-01-30 -0.006279
2015-02-06 -0.038963
```

[9]:
```
rets.head()
```

[9]:
```
                AAPL      NVDA      MSFT     GOOGL      AMZN      META  \
date
2015-01-09  0.024514 -0.009315  0.009195 -0.054445 -0.037534 -0.009055
2015-01-16 -0.053745  0.000836 -0.020131  0.019448 -0.020880 -0.032931
2015-01-23  0.065950  0.037578  0.020329  0.061685  0.074431  0.035255
2015-01-30  0.036997 -0.072636 -0.143706 -0.008130  0.134900 -0.024669
2015-02-06  0.019114  0.062269  0.049753 -0.006812  0.055737 -0.018967

                TSLA      AVGO     BRK/B       LLY       SPY
date
2015-01-09 -0.057685  0.047971  0.002011 -0.001855 -0.005744
2015-01-16 -0.065760 -0.010268 -0.001739  0.010726 -0.012827
2015-01-23  0.042575  0.030500 -0.000603  0.020514  0.016565
2015-01-30  0.011476 -0.038331 -0.034938 -0.001802 -0.026931
2015-02-06  0.067589  0.018037  0.043569 -0.022778  0.030584
```

2 1 Constrained Optimization for Mean-Variance

Continue working with the data above. Suppose we want to constrain the weights such that * there are no short positions beyond negative 20%, $w_i \geq -.20$ for all i * none of the positions may have weight over 35%, $w_i \leq .35$ for all i. * all the asset weights must sum to 1

Furthermore, * The targeted mean return is 20% per year. * Be careful; the target is an annualized mean.

Consider using the code below as a starting point.

2.1 1.1.

Report the weights of the constrained portfolio.

Report the mean, volatility, and Sharpe ratio of the resulting portfolio.

[10]:
```
import polars as pl
import numpy as np
from scipy.optimize import minimize

pl.Config.set_tbl_rows(-1)
pl.Config.set_tbl_cols(-1)
```

[10]:
```
polars.config.Config
```

[11]:
```
rets_df = pl.from_pandas(rets.reset_index())
print(rets_df.head(3))
```

```
shape: (3, 12)


 date     AAPL     NVDA     MSFT    GOOGL    AMZN    META    TSLA    AVGO
BRK/B   LLY     SPY
 ---      ---      ---      ---     ---      ---     ---     ---     ---     ---
 ---     ---
 dateti   f64      f64      f64     f64      f64     f64     f64     f64     f64
 f64     f64
 me[ns]




 2015-0   0.0245   -0.009   0.009   -0.05    -0.03   -0.00   -0.05   0.047
0.002   -0.00   -0.00
 1-09     14       315      195     4445     7534    9055    7685    971     011
 1855    5744
 00:00:

 00
```

```
 2015-0   -0.053   0.0008   -0.02   0.019   -0.02   -0.03   -0.06   -0.01
-0.00   0.010   -0.01
 1-16     745      36       0131    448     088     2931    576     0268
1739    726     2827
 00:00:

 00

 2015-0   0.0659   0.0375   0.020   0.061   0.074   0.035   0.042   0.030
-0.00   0.020   0.016
 1-23     5        78       329     685     431     255     575     5
0603    514     565
 00:00:

 00
```

```
[12]: FREQ = 52
      TARGET_MEAN = 0.20
```

```
[13]: # mean return per week
      mean_ret = rets_df.select(pl.col(pl.Float64)).mean() * FREQ
      print(mean_ret)
```

```
shape: (1, 11)

 AAPL     NVDA     MSFT     GOOGL    AMZN     META     TSLA     AVGO     BRK/B
 LLY      SPY
 ---      ---      ---      ---      ---      ---      ---      ---      ---
 ---      ---
 f64      f64      f64      f64      f64      f64      f64      f64      f64
 f64      f64


 0.2387   0.6455   0.2614   0.2168   0.2934   0.2619   0.4697   0.3948   0.1350
 0.2815   0.1312
 14       8        02                47       24       54       54       25
 42       64
```

Optimization setup: - objective function - constraints - bounds - initialization

```
[14]: ret_mat = rets_df.select(pl.col(pl.Float64)).to_numpy()
      cov_mat = np.cov(ret_mat.T) * FREQ # annualized
```

```python
# Define obj func
def objective(w):
    m = mean_ret @ w
    return -m / (w.T @ cov_mat @ w)

# Define constraints
def fun_constraint_capital(w):
    """Constraint: weights sum to 1"""
    return np.sum(w) - 1

def fun_constraint_mean(w):
    """Constraint: portfolio return equals target"""
    return (mean_ret.to_numpy()[0] @ w) - TARGET_MEAN

# Build constraints
constraint_capital = {'type': 'eq', 'fun': fun_constraint_capital}
# constraints = [constraint_capital]
constraint_mean = {'type': 'eq', 'fun': fun_constraint_mean}
constraints = [constraint_capital, constraint_mean]

# Build bounds
n_assets = ret_mat.shape[1]
bounds = tuple([(-0.20, 0.35) for _ in range(n_assets)])

# Set initial equal weights
w0 = np.array([1. / n_assets] * n_assets)
```

[15]:
```python
# Run optim
result = minimize(
    objective, w0, method='SLSQP', bounds=bounds, constraints=constraints,
    options={'disp': True, 'maxiter': 1000}
)
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: -7.356172491512938
            Iterations: 17
            Function evaluations: 216
            Gradient evaluations: 17

            Current function value: -7.356172491512938
            Iterations: 17
            Function evaluations: 216
            Gradient evaluations: 17
```

[16]:
```python
# Weights
w = result.x
w
```

```
[16]: array([ 0.02958052, -0.01359567,  0.14517307,  0.00886273,  0.09341806,
        0.00238082, -0.01616236,  0.03617394,  0.35      ,  0.21331584,
        0.15085305])
```

```
[17]: # Calc metrics
port_mean = mean_ret @ w
port_var = w.T @ cov_mat @ w
port_std = np.sqrt(port_var)
sharpe = port_mean / port_std
print(
    pl.DataFrame({
        "mean": port_mean, "vol": port_std, "sharpe": sharpe
    })
)
```

shape: (1, 3)

mean	vol	sharpe
---	---	---
f64	f64	f64
0.2	0.164888	1.212945

2.1.1 1.2.

Compare these weights to the assets' Sharpe ratios and means.

Do the most extreme positions also have the most extreme Sharpe ratios and means?

Why?

The asset with the max weight is BRK/B (0.35), one with the min weight is TSLA (-0.015).

```
[18]: print(
    pl.DataFrame(w.reshape(-1, 1), schema=mean_ret.schema)
)
```

shape: (1, 11)

AAPL	NVDA	MSFT	GOOGL	AMZN	META	TSLA	AVGO	BRK/B	LLY	SPY
---	---	---	---	---	---	---	---	---	---	---
f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
0.02958	-0.013	0.1451	0.0088	0.0934	0.0023	-0.016	0.0361	0.35		

```
0.2133   0.1508
1         596      73       63       18       81       162      74
16       53
```

[19]:
```python
print(mean_ret)
```

```
shape: (1, 11)

AAPL     NVDA     MSFT     GOOGL    AMZN     META     TSLA     AVGO     BRK/B
LLY      SPY
---      ---      ---      ---      ---      ---      ---      ---      ---
---      ---
f64      f64      f64      f64      f64      f64      f64      f64      f64
f64      f64


0.2387   0.6455   0.2614   0.2168   0.2934   0.2619   0.4697   0.3948   0.1350
0.2815   0.1312
14       8        02                47       24       54       54       25
42       64
```

[20]:
```python
vol = rets_df.select(pl.col(pl.Float64) * FREQ).std()
sharpe = mean_ret / vol
print(sharpe)
```

```
shape: (1, 11)

AAPL     NVDA     MSFT     GOOGL    AMZN     META     TSLA     AVGO     BRK/B
LLY      SPY
---      ---      ---      ---      ---      ---      ---      ---      ---
---      ---
f64      f64      f64      f64      f64      f64      f64      f64      f64
f64      f64


0.1196   0.1932   0.1510   0.1074   0.1329   0.1033   0.1110   0.1459   0.0982
0.1379   0.1065
68       42       54       31       86       83       84       66       13
7        28
```

Through comparison we find, the most extreme positions does not have the most extreme Sharpe ratios and means. The key insight is that mean-variance optimization considers the correlation structure and diversification benefits, not just individual asset metrics.

2.1.2 1.3.

Compare the bounded portfolio weights to the unbounded portfolio weights (obtained from optimizing without the inequality constraints, keeping the equality constraints.)

Report the mean, volatility, and Sharpe ratio of both.

[21]:
```python
result_unb = minimize(
    objective, w0, method='SLSQP', constraints=constraints,
    options={'disp': True, 'maxiter': 1000}
)
w_unb = result_unb.x
```

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: -7.358552021327915
            Iterations: 18
            Function evaluations: 237
            Gradient evaluations: 18
```

[22]:
```python
print(
    pl.DataFrame({
        "tickers": rets_df.select(pl.col(pl.Float64)).schema.names(),
        "bounded": w,
        "unbounded": w_unb
    })
)
```

shape: (11, 3)

tickers	bounded	unbounded
---	---	---
str	f64	f64
AAPL	0.029581	0.029871
NVDA	-0.013596	-0.014425
MSFT	0.145173	0.146105
GOOGL	0.008863	0.009667
AMZN	0.093418	0.093798
META	0.002381	0.002992
TSLA	-0.016162	-0.015357
AVGO	0.036174	0.036339
BRK/B	0.35	0.373036
LLY	0.213316	0.211059
SPY	0.150853	0.126914

[23]:
```python
def calc_metrics(w, mean_ret=mean_ret, cov_mat=cov_mat):
    port_mean = mean_ret @ w
    port_var = w.T @ cov_mat @ w
    port_std = np.sqrt(port_var)
    sharpe = port_mean / port_std
    return (
        pl.DataFrame({
            "mean": port_mean, "vol": port_std, "sharpe": sharpe
        })
    )
```

[24]:
```python
print(
    pl.DataFrame({
        "metrics": ["mean", "vol", "sharpe"],
        "bounded": calc_metrics(w).to_numpy()[0],
        "unbounded": calc_metrics(w_unb).to_numpy()[0]
    })
)
```

```
shape: (3, 3)

 metrics   bounded    unbounded
 ---       ---        ---
 str       f64        f64

 mean      0.2        0.2
 vol       0.164888   0.164861
 sharpe    1.212945   1.213141
```

2.2 Code Help

The `minimize` function will be how we optimize.

[25]:
```python
from scipy.optimize import minimize
```

Build the objective functions.

Before doing this, you will need to define * `TARGET_MEAN` * `FREQ` * `cov` * `mean`

[26]:
```python
# def objective(w):
#     return (w.T @ cov @ w)

# def fun_constraint_capital(w):
#     return np.sum(w) - 1

# def fun_constraint_mean(w):
#     return (mean @ w) - TARGET_MEAN
```

Build the constraints * sum of weights add to one * weighted average of means is the target mean

[27]:
```
# constraint_capital = {'type': 'eq', 'fun': fun_constraint_capital}
# constraint_mean = {'type': 'eq', 'fun': fun_constraint_mean}

# constraints = ([constraint_capital, constraint_mean])
```

Build the upper and lower bounds on each asset.

You will need to use the `minimize` function along with these contraints, bounds, and an initial guess.
