Deterministic Optimization

Illustration of the optimization process

Shabbir Ahmed
Anderson-Interface Chair and Professor
School of Industrial and Systems Engineering

Solving the portfolio optimization model

GTx

Solving the model

Learning objectives

- Discover how to set up data for the portfolio optimization problem
- Recognize how to optimize the problem using CVXPY

Portfolio Optimization model

Sum notation:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{3}\sum_{j=1}^{3} x_i x_j \sigma_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^{3} x_i \leq 1000.00, \\
& \sum_{i=1}^{3} \bar{r}_i x_i \geq r_{\min}, \\
& x_i \geq 0 \qquad i = 1, 2, 3.
\end{aligned}
$$

Matrix notation:

$$
\begin{aligned}
\min \quad & \mathbf{x}^\top Q \mathbf{x} \\
\text{s.t.} \quad & \mathbf{e}^\top \mathbf{x} \leq 1000.00, \\
& \bar{\mathbf{r}}^\top \mathbf{x} \geq r_{\min}, \\
& \mathbf{x} \geq \mathbf{0}
\end{aligned}
$$

Modified model

$$\begin{aligned} \min \quad & \mathbf{x}^\top Q \mathbf{x} \\ \text{s.t.} \quad & \mathbf{e}^\top \mathbf{x} \leq 1000.00, \\ & \bar{\mathbf{r}}^\top \mathbf{x} \geq r_{\min}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

$$\begin{aligned} \min \quad & \mathbf{x}^\top Q \mathbf{x} \\ \text{s.t.} \quad & \mathbf{e}^\top \mathbf{x} = 1.0, \\ & \bar{\mathbf{r}}^\top \mathbf{x} \geq r_{\min}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

The new variables denote the proportion of the budget invested. Given a solution of the modified model we can just multiply by 1000 to get the actual $ investments. The new desired return parameter is the $-per-$ return instead of the total return.

Data Collection

- Before we can attempt to solve the model we need to specify the required data or model parameters: $\bar{\mathbf{r}}$, Q and $r_{\min}$
- Reliable quantification of model parameters can only be done through careful observation and analysis of the actual data processes underlying the decision problem
- This is one of the most crucial steps in the optimization approach. One of the key reasons why optimization models can fail to provide a useful decision (or even provide a wrong decision) in practice is improper data specification. This is **the garbage in garbage out (GIGO) principle**
- Specification of model parameters again involves approximations and assumptions

Available Data

- File: monthly_prices.csv
- Closing price of each stock on the first day of each month for the last 24 months
- The last row indicates the current price (i.e. buy price)

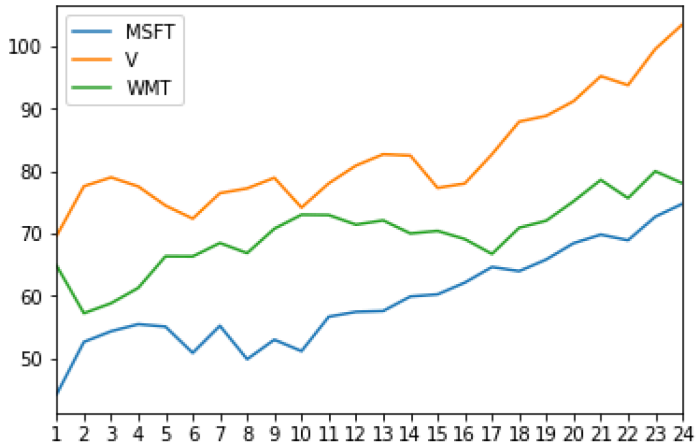
	MSFT	V	WMT
1	44.259998	69.660004	64.839996
2	52.639999	77.580002	57.240002
3	54.349998	79.010002	58.84
4	55.48	77.550003	61.299999
5	55.09	74.489998	66.360001
6	50.880001	72.389999	66.339996
7	55.23	76.480003	68.489998
8	49.869999	77.239998	66.870003
9	53	78.940002	70.779999
10	51.169998	74.169998	73.019997
11	56.68	78.050003	72.970001
12	57.459999	80.900002	71.440002
13	57.599998	82.699997	72.120003
14	59.919998	82.510002	70.019997
15	60.259998	77.32	70.43
16	62.139999	78.019997	69.120003
17	64.650002	82.709999	66.739998
18	63.98	87.940002	70.93
19	65.860001	88.870003	72.080002
20	68.459999	91.220001	75.18
21	69.839996	95.230003	78.599998
22	68.93	93.779999	75.68
23	72.699997	99.559998	79.989998
24	74.769997	103.519997	78.07

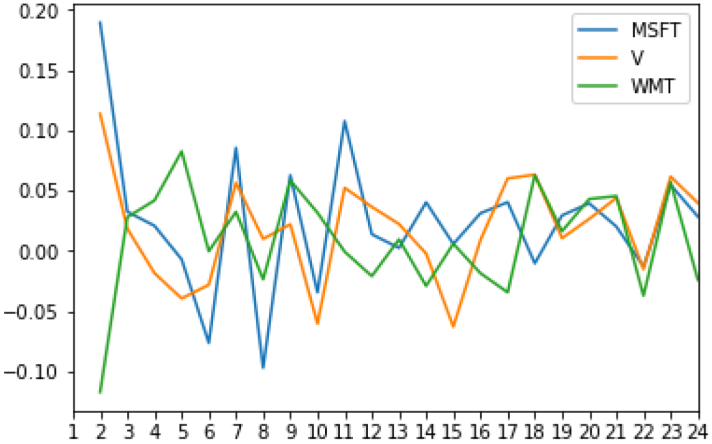
Price and Return

- Historical Monthly returns:

$$r_{jt} = \frac{p_{jt} - p_{jt-1}}{p_{jt-1}}$$

where
r_{jt} = return of stock j in month t
p_{jt} = price of stock j in month t





Estimating parameters

- We need to determine the expected values and the covariances for the end-of-month returns of the three stocks.
- **Assumption:** The monthly stock returns have a stationary probability distribution, and the historical data is sample from this distribution
- We can then estimate the expected values and covariances of this monthly return distribution through statistical analysis of historical data

Python 2.7 Code for Data Analysis

```
import pandas as pd
import numpy as np
from cvxpy import *

# read monthly_prices.csv
mp = pd.read_csv("monthly_prices.csv",index_col=0)
mr = pd.DataFrame()

# compute monthly returns
for s in mp.columns:
    date = mp.index[0]
    pr0 = mp[s][date]
    for t in range(1,len(mp.index)):
        date = mp.index[t]
        pr1 = mp[s][date]
        ret = (pr1-pr0)/pr0
        mr.set_value(date,s,ret)
        pr0 = pr1

# get symbol names
symbols = mr.columns

# convert monthly return data frame to a numpy matrix
return_data = mr.as_matrix().T

# compute mean return
r = np.asarray(np.mean(return_data, axis=1))

# covariance
C = np.asmatrix(np.cov(return_data))

# print out expected return and std deviation
print "----------------------"
for j in range(len(symbols)):
    print '%s: Exp ret = %f, Risk = %f' %(symbols[j],r[j], C[j,j]**0.5)
```

- pandas is a package for data analysis
- numpy is package for linear algebra operations
- cvxpy is the optimization package

- Output:

```
----------------------
MSFT: Exp ret = 0.024611, Risk = 0.058040
V: Exp ret = 0.018237, Risk = 0.042807
WMT: Exp ret = 0.009066, Risk = 0.044461
```

Model parameters

Expected return vector and Covariance matrix

```
In [10]: r
Out[10]: array([ 0.02461117,  0.01823726,  0.00906643])
```

$= \bar{\mathbf{r}}$

```
In [11]: C
Out[11]:
matrix([[ 0.00336865,  0.0016328 , -0.00075249],
        [ 0.0016328 ,  0.00183242, -0.00056339],
        [-0.00075249, -0.00056339,  0.00197676]])
```

$= Q$

Note that maximum expected return we can hope for is around 2.46%.
Let us set the required expected return to be 2% or 0.02 $= r_{\min}$

Remarks

- The analysis used is quite naive
- We have made a very simplistic assumption of the stock price distribution, no temporal effects are considered here
- Furthermore, the expectation and covariances are estimated based on a very small sample
- In practice, the data collection and analysis process would involve very sophisticated statistical and time series models to obtain reliable estimates of the means and covariances

Python 2.7 Code for Optimization

```
# set up optimization model
n = len(symbols)
x = Variable(n)
req_return = 0.02
ret = r.T*x
risk = quad_form(x, C)
prob = Problem(Minimize(risk),
               [sum_entries(x) = 1, ret >= req_return,
                x >= 0])

# solve problem and write solution
try:
    prob.solve()
    print "----------------------"
    print "Optimal portfolio"
    print "----------------------"
    for s in range(len(symbols)):
        print 'x[%s] = %f' %(symbols[s],x.value[s,0])
    print "----------------------"
    print 'Exp ret = %f' %(ret.value)
    print 'risk    = %f' %((risk.value)**0.5)
    print "----------------------"
except:
    print 'Error'
```

$$
\begin{array}{ll}
\min & \mathbf{x}^\top Q \mathbf{x} \\
\text{s.t.} & \mathbf{e}^\top \mathbf{x} = 1.0, \\
 & \bar{\mathbf{r}}^\top \mathbf{x} \geq r_{\min}, \\
 & \mathbf{x} \geq \mathbf{0}
\end{array}
$$

- Output:

```
----------------------
Optimal portfolio
----------------------
x[MSFT] = 0.582818
x[V] = 0.204324
x[WMT] = 0.212858
----------------------
Exp ret = 0.020000
risk    = 0.038256
----------------------
```

Summary

- We made some simplifying assumptions is setting up the model as well as identifying model parameters
- Python packages pandas, numpy and cvxpy offer great tools for data analysis, matrix operations and optimization