Portfolio Theory & Application: Final

Due on Tuesday, August 22, 2017

Professor Starer 18:15pm

Yuxuan Xia

Contents

Problem 1

Question 1.1

(Standard Time-Weighted Returns—Adjustment to Numerator). An investor gives \$1.0 million to a portfolio manager to manage for two years. Close to the end of the first year, the investor gives the manager a further \$0.1 million to manage. At the end of the first year, the portfolio value has risen to \$1.2 million. At the end of the second year, the portfolio value is \$1.44 million. Each year, the portfolio is valued after the external cash flow for that year.

a. Using the usual time-weighted method,

$$r = \prod_{t=1}^{T} \frac{W_t - C_t}{W_{t-1}} - 1 \tag{1}$$

compute the net rate of return $r = R - 1$ over the two years.

Using the time-weighted average formula and substituting the numerical values:

$$\begin{aligned} R &= \prod_{t=1}^{T} \frac{W_t - C_t}{W_{t-1}} \\ &= \frac{W_1 - C_1}{W_0} \frac{W_2 - C_2}{W_1} \\ &= \frac{1.2 - 0.1}{1} \frac{1.44}{1.2} \\ &= 1.32 \end{aligned}$$

the net rate of return over the two years:

$$r = R - 1 = 0.32 = 32\%$$

b. Why, do you think, might this method give an estimate of return that could be too high?

Because the interest rate is over 2 years, the internal rate of return over one year is the root of equation: $(1+r)^2 = 1.32$, $r = 14.9\%$. I can't declare it is too high or not, just by comparison with money-weighted annual return, using the formula in question 1.3

$$W_0 \cdot R^2 + I \cdot R = W_2$$

we get the answer $R = 1.151$, and $r = R - 1 = 15.1\%$, actually higher than what we get here. So, it's not supposed to say the estimate return is too high without other information.

Question 1.2

(Alternative time-weighted method—Adjustment to Denominator). The basic conditions as in Question 1.1 apply. That is, an investor gives \$1.0 million to a portfolio manager to manage for two years. Close to the end of the first year, the investor gives the manager a further \$0.1 million to manage. At the end of the first year, the portfolio value has risen to \$1.2 million. At the end of the second year, the portfolio value is \$1.44 million. Each year, the portfolio is valued after the external cash flow for that year. Show all of your working.

a. Write an expression for the gross return $R = W_2 = W_0$ in terms of a product of the gross returns over the two periods using only the quantities defined above, in which you adjust only denominators for cash flows.

$$\begin{aligned} R &= \frac{W_1^-}{W_0} \cdot \frac{W_1^+}{W_1^- + C} \cdot \frac{W_2}{W_1^+} \\ &= \frac{W_1^-}{W_0} \cdot \frac{W_2}{W_1^- + C} \end{aligned}$$

b. Substitute the numerical values into your expression, and find the net rate of return $r = R - 1$ over the two years. Comment on any difference between the return computed this way, and the return you computed in Question 1.1.

$$\begin{aligned} R &= \frac{1.1}{1} \cdot \frac{1.2}{1.1 + 0.1} \cdot \frac{1.44}{1.2} \\ &= 1.32 \end{aligned}$$

It's the same as what we get in question 1.1. Why?
In question 1.1, we adjust only numerator and in question 1.2, we adjust only denominator. Comparing these 2 cases in the split form:
1. Adjust denominator

$$\begin{aligned} R &= \frac{W_1^-}{W_0} \cdot \frac{W_1^+}{W_1^- + C} \cdot \frac{W_2}{W_1^+} \\ &= \frac{W_1^-}{W_0} \cdot \frac{W_2}{W_1^- + C} \end{aligned}$$

2. Adjust numerator

$$\begin{aligned} R &= \frac{W_1^-}{W_0} \cdot \frac{W_1^+ - C}{W_1^-} \cdot \frac{W_2}{W_1^+} \\ &= \frac{W_1^+ - C}{W_0} \cdot \frac{W_2}{W_1^+} \end{aligned}$$

Because $W_1^+ - C = W_1^-$,these two equations are equal.

Question 1.3

(Money-Weighted Returns, IRR). An investor gives \$1.0 million to a portfolio manager to manage for two years. Close to the end of the first year, the investor gives the manager a further \$0.1 million to manage. At the end of the first year, the portfolio value has risen to \$1.2 million. At the end of the second year, the portfolio value is \$1.32 million.Using the method outlined below, compute the manager's annual Internal Rate of Return. Show all of your working.

a.

$$W_0 \cdot R^2 + I \cdot R = W_2$$

b.
Since $R > 0$:

$$R = \frac{-I + \sqrt{I^2 + 4W_0W_2}}{2W_0}$$

c.

$$R = \frac{-0.1 + \sqrt{0.1^2 + 4 \times 1 \times 1.32}}{2 \times 1} = 1.1$$

Problem 2

Show mathematically (i.e., not graphically or geometrically) how a portfolio's return can be separated into a sum of four terms representing asset allocation return r_{AA}, security selection return r_{SS}, benchmark return r_B, and an interaction effect r_{INT}: In the geometric representation, give expressions for r_{AA}, r_{SS}, r_B, and r_{INT}:

The definition of asset allocation return, benchmark return, security selection return and interaction effect return are listed below:

$$\begin{aligned}
r_{AA} &= \sum_{j=1} M(h_j - h_j^B)r_j^B \\
r_B &= \sum_{j=1} M h_j^B r_j^B \\
r_{SS} &= \sum_{j=1} M h_j^B (r_j - r_j^B) \\
r_{INT} &= \sum_{j=1} M(h_j - h_j^B)(r_j - r_j^B)
\end{aligned}$$

$$\begin{aligned}
r_P &= \sum_{j=1} M h_j r_j \\
&= \sum_{j=1} M(h_j - h_j^B + h_j^B)(r_j - r_j^B + r_j^B) \\
&= \sum_{j=1} M h_j^B r_j^B + \sum_{j=1} M h_j^B (r_j - r_j^B) + \sum_{j=1} M(h_j - h_j^B) r_j^B + \sum_{j=1} M(h_j - h_j^B)(r_j - r_j^B)
\end{aligned}$$

by definition:

$$r_P = r_B + r_{SS} + r_{AA} + r_{INT}$$

Problem 3

Question 3.1

The optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^T Q x - \mu^T x \\ \text{subject to} \quad & Bx \geq c \\ & Dx < e \\ & Fx = g \end{aligned}$$

or equivalently:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^T Q x - \mu^T x \\ \text{subject to} \quad & Bx \geq c \\ & -Dx > -e \\ & Fx \geq g \\ & -Fx \geq -g \end{aligned}$$

However, the 2nd constraint cannot be transformed to non-negativity constraint, it might be an error or we should back test our optimized results to check this inequation to make sure it to be satisfied.
In matrix form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & x^T Q x - \mu^T x \\ \text{subject to} \quad & \begin{pmatrix} B \\ -D \\ F \\ -F \end{pmatrix} x \geq \begin{pmatrix} c \\ -e \\ g \\ -g \end{pmatrix} \end{aligned}$$

let

$$A = \begin{pmatrix} B \\ -D \\ F \\ -F \end{pmatrix}$$

$$b = \begin{pmatrix} c \\ -e \\ g \\ -g \end{pmatrix}$$

we have:

$$Ax \geq b$$

Question 3.2

In optimization problem, we want to convert a linear programmer to standard form, whereas:

1. Non-negative constraints for all variables
2. All remaining constraints are expressed as equality constraints
3. The right hand side vector b is non-negative.

Based on these conditions, the terms are introduced:

free variable: variable which is unconstrained in sign. And it can be canceled by substitution

slack variable: To convert a $\leq$ constraint into standard form, we add a new variable in the left hand side of the inequality called slack variable. In this case the inequality constraint becomes equality constraint. We also requires that the slack variable is non-negative.

surplus variable: Similar to $\leq$ constraint, to convert a $\geq$ constraint to standard form, we also add a new variable in the left hand side of the inequality called surplus variable. In this case, the surplus variable is non-negative.

Problem 4

Question 4.1

Assume that every stock's return can be expressed in the single-factor form$ri = a_i + x_i\phi$, where a_i is a constant for stock i; ϕ is a random variable, and x_i is the exposure of stock i to the random variable. Assume that you construct a portfolio with two different stocks, i and j: You use weights w_i and w_j whose sum is 1. Find an expression for the portfolio's return in terms of a risk-less part and a risky part that involves the random variable ϕ:

$$\begin{aligned}
r_i &= a_i + x_i\phi \\
r_j &= a_j + x_j\phi \\
r_p &= w_i r_i + w_j r_j \\
&= w_i a_i + w_i x_i\phi + w_j a_j + w_j x_j\phi \\
&= (w_i a_i + w_j a_j) + (w_i x_i + w_j x_j)\phi
\end{aligned}$$

$$\begin{aligned}
&\text{riskless part: } w_i a_i + w_j a_j \\
&\text{risky part: } (w_i x_i + w_j x_j)\phi
\end{aligned}$$

Question 4.2

Create a risk-free portfolio by setting the coefficient of ϕ equal to zero. Find expressions for w_i and w_j in terms of x_i and x_j :

In order to create a risk-free portfolio, the risky part of portfolio should be equal to zero, that is:

$$w_i x_i + w_j x_j = 0$$

substitute to the budget constraint:

$$w_i = \frac{w_j x_j}{x_i} = 1 - w_j$$

$$w_j(1 + \frac{x_j}{x_i}) = 1$$

$$w_j = \frac{x_i}{x_i + x_j}$$

Similarly:

$$w_i = \frac{x_j}{x_i + x_j}$$

Problem 5

Inference and Clarification before Question 5

To be start with, I thought it might be ambiguity of terminology in this question. So, first of all what is the (α, β, ω) corresponding to? The benchmark return or market return, because they both occured in the question.

First of all, I assume they are with respect to benchmark return as mentioned in textbook explicitly. It seems make sense because all the formula derived are based on this assumption.

However in this case, we have benchmark weights' value b and if (α, β, ω) are based on benchmark return, the following equation should always be hold.

Multiply benchmark's weights b_i in both side, and by summation with respect to index i,

$$\sum_i b_i \mu_i = \sum_i b_i \alpha_i + \sum b_i \beta_i \mu_b$$

equivalent to

$$\mu_b = \sum_i b_i \alpha_i + \sum_i b_i \beta_i \mu_b$$

transformation

$$\sum_i b_i \alpha_i = \mu_b (1 - \sum_i b_i \beta_i)$$

since the above equation always held

both sides should equal to zero, that is

$$\sum_i b_i \beta_i \equiv 1$$

$$\sum_i b_i \alpha_i \equiv 0$$

however, substitute the numerical values into expressions

$$\sum_i b_i \beta_i = 0.888 \neq 1$$

$$\sum_i b_i \alpha_i = 0.0737 \neq 0$$

or

$$\mu_b \equiv \sum_i \frac{b_i \alpha_i}{1 - \sum_i b_i \beta_i}$$

however, by substitution

$$\mu_b = \frac{0.0737}{1 - 0.888} = 65.8\%$$

Problem 5 [Inference and Clarification before Question 5] continued on next page...

Apparently it's insane to say benchmark's return rate is 65.8% while the market's return rate is only 7%. What a happy era!! Why don't they hit the bank?
So I assume (α, β, ω) is based on market return in following derivatives.
The expected return is expressed as μ or "mu" as below:

```
data:
   alpha  beta  omega     b     mu
0   0.10   1.2   0.30  0.10  0.184
1   0.09   1.1   0.25  0.12  0.167
2   0.11   1.3   0.27  0.12  0.201
3   0.05   0.8   0.20  0.06  0.106
4   0.08   0.9   0.22  0.21  0.143
5   0.07   0.7   0.20  0.08  0.119
6   0.05   0.6   0.20  0.06  0.092
7   0.04   0.6   0.29  0.08  0.082
8   0.08   0.8   0.24  0.06  0.136
9   0.03   0.5   0.20  0.11  0.065
market's standard deviation: 0.2
risk-free interest rate: 0.03
expected market's return: 0.07
expected benchmark's return: 0.13586
```

Check whether our expected return is equal. If not, the definition of $\alpha, \beta, and \omega$ might be different as mentioned above.

Question 5.1

What is the minimum variance portfolio, and what is the standard deviation of that portfolio's return?

optimization problem:

$$\begin{aligned} \underset{h}{\text{minimize}} \quad & h^T Q h \\ \text{subject to} \quad & h^T \iota = 1 \end{aligned}$$

Using single-index covariance model, the covariance matrix:

$$Q = \beta \sigma_M^2 \beta^T + \Omega$$

Scheme 1: numerical method

```
Q5.1, minium variance portfolio:
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 0.011652713471109345
            Iterations: 8
            Function evaluations: 96
            Gradient evaluations: 8
numerical result:
h:
[-0.0387583  -0.024372   -0.0870158   0.13903705  0.07134432  0.19105031
  0.24306358  0.1124022   0.09817179  0.29507684]
expected return: 0.0830935137587
expected alpha: 0.0437471877547
var:0.0233054269422
std:0.15266115073
```

Scheme 2: matrix form

$$h_v = \frac{1}{\iota^T Q^{-1} \iota} Q^{-1} \iota$$

Result:

```
using matrix formula
h:
[-0.04047276 -0.02236738 -0.08075632  0.13339501  0.06386804  0.18950968
  0.24562436  0.11682491  0.09263542  0.30173904]
expected return:0.0827988305643
expected alpha: 0.0435185551791
var:0.0232924975291
std:0.152618798086
```

Question 5.2

Use the Elton-Gruber-Padberg algorithm with Sharpe's Single Index Model to choose a long-only portfolio. What is that portfolio?

Using Elton et al.'s Method and Sharpe's Single Index Model mentioned in class 7, and convert the matlab script to python script. I got the following result:(EGP portfolio weights see column x)

Q5.2, EGP with Sharpe's single index model, long-only:

	alpha	beta	omega	b	mu	c	cstar	z	x
8	0.08	0.8	0.24	0.06	0.136	0.062500	0.042581	0.276652	0.180652
2	0.11	1.3	0.27	0.12	0.201	0.061538	0.040889	0.338060	0.220751
0	0.10	1.2	0.30	0.10	0.184	0.058333	0.022764	0.210030	0.137148
5	0.07	0.7	0.20	0.08	0.119	0.057143	0.042370	0.254831	0.166403
4	0.08	0.9	0.22	0.21	0.143	0.055556	0.040813	0.241261	0.157542
1	0.09	1.1	0.25	0.12	0.167	0.054545	0.032958	0.210573	0.137503
6	0.05	0.6	0.20	0.06	0.092	0.033333	0.041778	-0.138716	0.000000
3	0.05	0.8	0.20	0.06	0.106	0.025000	0.038335	-0.351621	0.000000
7	0.04	0.6	0.29	0.08	0.082	0.016667	0.041020	-0.184883	0.000000
9	0.03	0.5	0.20	0.11	0.065	0.000000	0.040909	-0.532263	0.000000

expected return: 0.159468497187
expected alpha: 0.0890765081327
variance:0.0510649568459
std:0.225975566922

Question 5.3

Extend Question 5.2 above to produce a long-short portfolio using the Lintnerian definition of shorts. What is your optimal portfolio now? What are the expected value and standard deviation of that portfolio's return

	alpha	beta	omega	b	mu	c	cstar	z	x
8	0.08	0.8	0.24	0.06	0.136	0.062500	0.042581	0.276652	0.101009
2	0.11	1.3	0.27	0.12	0.201	0.061538	0.040889	0.338060	0.123430
0	0.10	1.2	0.30	0.10	0.184	0.058333	0.022764	0.210030	0.076684
5	0.07	0.7	0.20	0.08	0.119	0.057143	0.042370	0.254831	0.093042
4	0.08	0.9	0.22	0.21	0.143	0.055556	0.040813	0.241261	0.088087
1	0.09	1.1	0.25	0.12	0.167	0.054545	0.032958	0.210573	0.076883
6	0.05	0.6	0.20	0.06	0.092	0.033333	0.041778	-0.138716	-0.050647
3	0.05	0.8	0.20	0.06	0.106	0.025000	0.038335	-0.351621	-0.128381
7	0.04	0.6	0.29	0.08	0.082	0.016667	0.041020	-0.184883	-0.067503
9	0.03	0.5	0.20	0.11	0.065	0.000000	0.040909	-0.532263	-0.194335

expected return: 0.0527293678347
expected alpha: 0.0323241568101
variance:0.00937349347537
std:0.0968168036829

Compared with the long-only portfolio in question 5.2. This one has even lower standard deviation as well as lower expected portfolio return.

Question 5.4

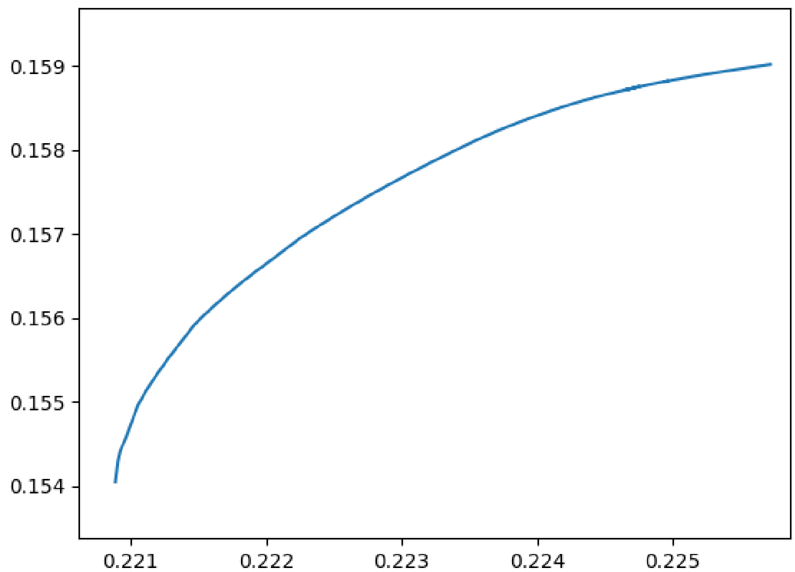
Using the data provided in the table above, construct the covariance matrix Q; and use quadratic programming to find the optimal portfolio that satisfies a long-only constraint (i.e., no short positions), the budget constraint, and the constraint $\beta_P = 1$ for a range of risk tolerances from $\tau = 0.001$ to $\tau = 1$: Plot the resulting efficient frontier.

Problem 5 [Question 5.4] continued on next page...

The optimization problem:

$$
\begin{aligned}
\underset{h}{\text{minimize}} \quad & \frac{1}{2}h^T Q h - \tau \mu^T h \\
\text{subject to} \quad & h^T \iota = 1 \\
& h^T \beta = 1 \\
& h \geq 0
\end{aligned}
$$

the efficient frontier:



Question 5.5

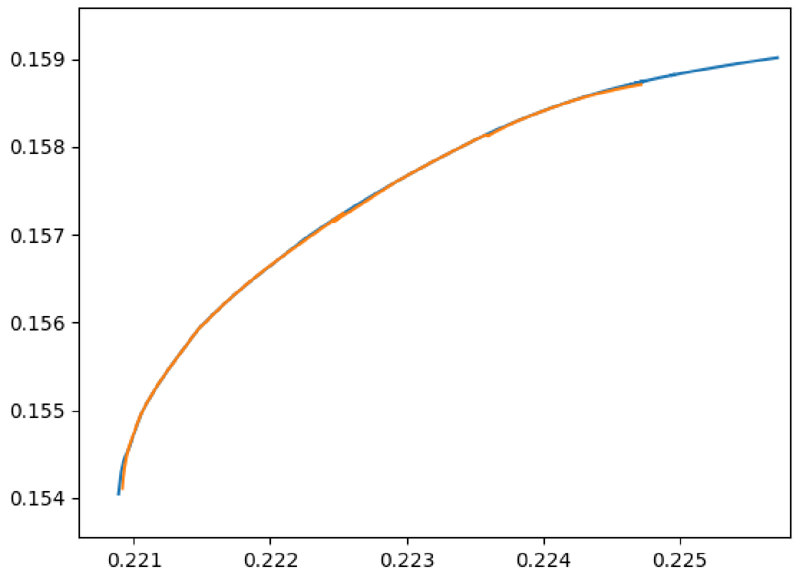
Repeat Question 5.4 above, but now include the constraint that the portfolio should have no weight should be greater that 20%. Superimpose this efficient frontier on the efficient frontier from Question 5.4 and explain any differences.

The optimization problem:

$$\begin{aligned} \underset{h}{\text{minimize}} \quad & \frac{1}{2}h^T Q h - \tau \mu^T h \\ \text{subject to} \quad & h^T \iota = 1 \\ & h^T \beta = 1 \\ & h \geq 0 \\ & -h \geq -0.2\iota \end{aligned}$$

in the last constraint, ι is the unit vector which defined as np.ones(h.shape) in python

the efficient frontier:



The orange line is the efficient frontier of question 5.5(portfolio A) whereas the blue line is the efficient frontier of question 5.4(portfolio B).
We found that they share the same curve in the middle of the efficient frontier, while slightly different at the left and right side.
The reason of this phenomenon is in the middle of the curve, they share the same portfolio weights while at the left and right side, their portfolio weights are different because of the weight constraint. In these area, portfolio A's 3rd and 9th (indexed from 1) security's weights are greater than 20% while portfolio B's cannot.

Question 5 Code

Listing 1: question 5 preprocessing

```
import numpy as np
import pandas as pd
```

```
alpha_array = np.array([10, 9, 11, 5, 8, 7, 5, 4, 8, 3]) * 0.01
beta_array = np.array([1.2, 1.1, 1.3, 0.8, 0.9, 0.7, 0.6, 0.6, 0.8, 0.5])
omega_array = np.array([30, 25, 27, 20, 22, 20, 20, 29, 24, 20]) * 0.01
b_array = np.array([10, 12, 12, 6, 21, 8, 6, 8, 6, 11]) * 0.01
sigmaM = 0.2  # market standard deviation
muM = 0.07  # expected market return
rf = 0.03  # risk-free rate
re_array = alpha_array + beta_array * muM  # expected securities return array
rb = np.sum(re_array * b_array)  # benchmark return
n_features = len(beta_array)  # number of securities
eta = np.ones(n_features)

df = pd.DataFrame(columns=['alpha', 'beta', 'omega', 'b', 'mu'])
df.alpha = alpha_array
df.beta = beta_array
df.omega = omega_array
df.mu = df.alpha + df.beta * muM
mu_array = df.mu
df.b = b_array
print("data:")
print(df)
print("market's standard deviation: " + str(sigmaM))
print("risk-free interest rate: " + str(rf))
print("expected market's return: " + str(muM))
print("expected benchmark's return: " + str(rb))


def compute_expected_alpha(h, alpha):
    return np.dot(h, alpha)


def compute_expected_return(h, alpha, beta, muM):
    return np.dot(h, alpha + beta * muM)


def compute_standard_deviation(h, cov):
    var = np.dot(np.dot(h.T, cov), h)
    std = np.sqrt(var)
    return std


def compute_covariance(beta, omega, sigmaM):
    n_features = len(omega)
    cov = np.zeros((n_features, n_features))
    for i in range(n_features):
        for j in range(n_features):
            cov[i, j] = beta[i] * beta[j] * sigmaM ** 2
            if i == j:
                cov[i, j] += omega[i] ** 2
    return cov


cov = compute_covariance(beta_array, omega_array, sigmaM)
```

```
def compute_portfolio_variance(h, Q):
    return np.dot(np.dot(h, Q), h)

# print(compute_portfolio_variance(np.array([1,2]),np.array([[1,2],[3,4]])))
```

Listing 2: question 5 source code

```
from question5_utils import *
import matplotlib.pyplot as plt
import scipy as sp
import scipy.optimize
import numpy as np

###############################################################################
# 5.1
cons = ({'type': 'eq',
         'fun': lambda h: np.dot(h, eta) - 1,
         'jac': lambda h: eta})

varP = lambda h: compute_portfolio_variance(h, Q=cov)
negativeAdjReturn = lambda h, tau: 1 / 2 * varP(h) - np.dot(h, mu_array) * tau
# print(cov)
print("Q5.1, minium variance portfolio:")
res = sp.optimize.minimize(negativeAdjReturn, np.zeros(n_features), args=(0,),
                           constraints=cons, options={'disp': True})
h_op = res.x
print("numerical result:")
print("h:")
print(h_op)
print("expected return: "
      + str(np.sum(h_op * (alpha_array + beta_array * muM))))
print("expected alpha: " + str(np.sum(h_op * alpha_array)))
var1_1 = compute_portfolio_variance(h_op, cov)
print("var:" + str(var1_1))
print("std:" + str(np.sqrt(var1_1)))
plt.scatter(np.sqrt(var1_1),
            np.sum(h_op * (alpha_array + beta_array * muM)), marker='o', c='g')
h_v = np.dot(np.matrix(cov).I, eta) / np.dot(np.dot(eta, np.matrix(cov).I), eta)
h_v = np.array(h_v)[0, :]
print("using matrix formula")
print("h:")
print(h_v)
print("expected return:" + str(np.sum(h_v * (alpha_array + beta_array * muM))))
print("expected alpha: " + str(np.sum(h_v * alpha_array)))
var1_2 = compute_portfolio_variance(h_v, cov)
print("var:" + str(var1_2))
print("std:" + str(np.sqrt(var1_2)))

#################################################################################
# 5.2
print("Q5.2, EGP with Sharpe's single index model, long-only:")

```

```python
def compute_traynor_ratio(mu, beta, rf):
    return (mu - rf) / beta


def compute_cumulative_threshold(mu_list, beta_list, omega_list, sigmaM, rf):
    mu_array = np.array(mu_list)
    beta_array = np.array(beta_list)
    omega_array = np.array(omega_list)
    numerator = sigmaM ** 2 * np.sum((mu_array - rf)
                                     / omega_array ** 2 * beta_array)
    denominator = 1 + sigmaM ** 2 * np.sum(beta_array ** 2 / omega_array ** 2)
    return numerator / denominator


c_array = compute_traynor_ratio(alpha_array, beta_array, rf)
# print(c_array)
cstar_array = np.zeros(n_features)
for i in range(n_features):
    cstar_array[i] /= compute_cumulative_threshold(
        alpha_array[:i + 1], beta_array[:i + 1],
        omega_array[:i + 1], sigmaM, rf)

# print(cstar_array)
df['c'] = c_array
df['cstar'] = cstar_array
inportfolio = c_array > cstar_array
thecstar = np.max(cstar_array)
z_array = (alpha_array - rf) / omega_array ** 2 \
          - beta_array * thecstar / omega_array ** 2
df['z'] = z_array
x_array = z_array[inportfolio] \
          / np.sum(z_array[inportfolio])
df['x'] = np.zeros(n_features)
df.x[inportfolio] = x_array
print(df.sort_values(by='c', ascending=False))
r2 = np.sum((df.alpha + df.beta * muM) * df.x)
print("expected return: " + str(r2))
print("expected alpha: " + str(np.sum(df.x * alpha_array)))
var2 = compute_portfolio_variance(
    df.x, compute_covariance(df.beta, df.omega, sigmaM))
print("variance:" + str(var2))
print("std:" + str(np.sqrt(var2)))
plt.scatter(np.sqrt(var2), r2, marker='^', c='r')
######################################################################
# 5.3
cstarN = compute_cumulative_threshold(
    alpha_array, beta_array, omega_array, sigmaM, rf)
df2 = df.copy()
x_array2 = z_array / np.sum(np.abs(z_array))
df2.x = x_array2
print(df2.sort_values(by='c', ascending=False))
r3 = np.sum((df2.alpha + df2.beta * muM) * df2.x)
```

```
print("expected return: " + str(r3))
print("expected alpha: " + str(np.sum(df2.x * alpha_array)))
var3 = compute_portfolio_variance(
    df2.x, compute_covariance(df2.beta, df2.omega, sigmaM))
print("variance:" + str(var3))
print("std:" + str(np.sqrt(var3)))
plt.scatter(np.sqrt(var3), r3, marker='s', c='b')
################################################################
# 5.4
cov = compute_covariance(beta_array, omega_array, sigmaM)
print('rb:' + str(rb))
cons4 = ({'type': 'eq',
          'fun': lambda h: np.dot(h, eta) - 1,
          'jac': lambda h: eta},
         {'type': 'eq',
          'fun': lambda h: np.dot(h, beta_array) - 1,
          'jac': lambda h: beta_array},
         {'type': 'ineq',
          'fun': lambda h: h})

re_list = []
std_list = []
h_list = []
for i in range(100):
    taui = 0.001 + i * (1 - 0.001) / 99
    res = sp.optimize.minimize(
        negativeAdjReturn,
        np.zeros(n_features), args=(taui,), constraints=cons4,
        options={'disp': False})
    h_i = res.x
    # print("h:"+str(h_i))
    re_i = np.sum(h_i * mu_array)
    # print("expected return:" +str(re_i))
    var_i = compute_portfolio_variance(h_i, cov)
    std_i = np.sqrt(var_i)
    # print("std:" + str(std_i))
    re_list.append(re_i)
    std_list.append(std_i)
    h_list.append(h_i)
print("if h_list greater than 0.2")
print(np.array(h_list) > 0.2)
plt.plot(std_list, re_list)
# plt.show()

#####################################################################
# 5.5
cov = compute_covariance(beta_array, omega_array, sigmaM)
print('rb:' + str(rb))
cons4 = ({'type': 'eq',
          'fun': lambda h: np.dot(h, eta) - 1,
          'jac': lambda h: eta},
         {'type': 'eq',
          'fun': lambda h: np.dot(h, beta_array) - 1,
```

```
          'jac': lambda h: beta_array},
         {'type': 'ineq',
          'fun': lambda h: h},
         {'type': 'ineq',
          'fun': lambda h: np.ones(n_features) * 0.2 - 1e-5 - h})

re_list2 = []
std_list2 = []
h_list2 = []
for i in range(100):
    taui = 0.001 + i * (1 - 0.001) / 99
    res = sp.optimize.minimize(negativeAdjReturn, np.zeros(n_features),
                               args=(taui,),
                               constraints=cons4,
                               options={'disp': False})
    h_i = res.x
    # print("h:"+str(h_i))
    re_i = np.sum(h_i * (alpha_array + beta_array * muM))
    # print("expected return:" +str(re_i))
    var_i = compute_portfolio_variance(h_i, cov)
    std_i = np.sqrt(var_i)
    # print("std:" + str(std_i))
    re_list2.append(re_i)
    std_list2.append(std_i)
    h_list2.append(h_i)

print(np.array(h_list2) > 0.2)
plt.plot(std_list2, re_list2)
plt.show()
```

Problem 6

Priori Analysis

Before inference, there might be some ambiguity should be clarified.

1. Since it is active portfolio management. The object function can be set as maximum expected alpha, however, that makes sense only when active beta equals to 0, $\beta^T w = 0$ satisfied. Here w defined as active weight not omega.

2. If there is no active beta constraint, the object function of active portfolio should be maximum active expected return, $w^T \mu$

Here, I classify the question into 4 cases corresponding to 4 optimization problems.
The optimization problem 1:
Literal interpretation:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & h^T \alpha \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \end{aligned}$$

The optimization problem 2:
Maximize active return:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & w^T \mu \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \end{aligned}$$

The optimization problem 3:
Maximize active alpha add active beta constraint:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & h^T \alpha \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \\ & w^T \beta = 0 \end{aligned}$$

The optimization problem 4:
Maximize active return add active beta constraint:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & w^T \mu \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \\ & w^T \beta = 0 \end{aligned}$$

Question 6.1

Find the active portfolio that maximizes your expected alpha, α_A; subject to the constraints that the sum of the active weights is zero, and that the tracking error is 3%.

The optimization problem 1:
Literal interpretation:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & h^T \alpha \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \end{aligned}$$

```
portfolio that maximize the portfolio's expected alpha under only budget constraint
and tracking error constraint:
portfolio h:[ 0.12682085  0.14652351  0.16349059  0.01851346  0.23467294  0.09825723
  0.0331393   0.05475387  0.08580988  0.03801837]
portfolio total weight:1.0
portfolio expected active return:0.0136993099443
portfolio expected active alpha:0.00858314174896
portfolio expected return:0.149559309944
benchmark expected return:0.13586
portfolio standard deviation:0.213374923117
benchmark standard deviation:0.195702733757
portfolio tracking error:0.0300000007055
portfolio beta:0.961088117077
portfolio expected residual return:0.0189858783583
portfolio information ratio:0.286104718237
```

The optimization problem 2:
Maximize active return:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & w^T \mu \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \end{aligned}$$

```
portfolio that maximize the portfolio's expected active return under only budget constraint
and tracking error constraint:
portfolio h:[ 0.13034796  0.15122415  0.16816663  0.03415388  0.23140577  0.08303503
  0.02740835  0.05683097  0.07564293  0.04178433]
portfolio total weight:1.0
portfolio expected active return:0.0139575831582
portfolio expected active alpha:0.00842452725215
portfolio expected return:0.149817583158
benchmark expected return:0.13586
portfolio standard deviation:0.214574680084
benchmark standard deviation:0.195702733757
portfolio tracking error:0.030000002666
portfolio beta:0.967043655801
portfolio expected residual return:0.0184350320811
portfolio information ratio:0.280817550117
```

The optimization problem 3:
Maximize active alpha add active beta constraint:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & h^T \alpha \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \\ & w^T \beta = 0 \end{aligned}$$

```
portfolio that maximize the portfolio's expected alpha under budget constraint,
tracking error constraint and active market beta constraint(active_weights * beta == 0)
portfolio h:[ 0.09380987  0.10818563  0.11478956 -0.02818039  0.23457338  0.15744038
  0.07643642  0.06094841  0.11662757  0.06536917]
portfolio total weight:1.0
portfolio expected active return:0.00397326102965
portfolio expected active alpha:0.00397326102966
portfolio expected residual return:0.0191895810297
portfolio expected return:0.13983326103
benchmark expected return:0.13586
portfolio standard deviation:0.19802231674
benchmark standard deviation:0.195702733757
portfolio tracking error:0.0300000395014
portfolio beta:0.888
portfolio information ratio:0.132441859934
```

The optimization problem 4:
Maximize active return add active beta constraint:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & w^T \mu \\ \text{subject to} \quad & h^T \iota = 1 \\ & \sqrt{w^T Q w} = 0.03 \\ & w^T \beta = 0 \end{aligned}$$

```
portfolio that maximize the portfolio's expected active return under budget constraint,
tracking error constraint and active market beta constraint(active_weights * beta == 0)
portfolio h:[ 0.09380987  0.10818563  0.11478956 -0.02818039  0.23457338  0.15744038
  0.07643642  0.06094841  0.11662757  0.06536917]
portfolio total weight:1.0
portfolio expected active return:0.00397326102931
portfolio expected active alpha:0.00397326102932
portfolio expected residual return:0.0191895810293
portfolio expected return:0.139833261029
benchmark expected return:0.13586
portfolio standard deviation:0.198022316732
benchmark standard deviation:0.195702733757
portfolio tracking error:0.030000039499
portfolio beta:0.888
portfolio information ratio:0.132441859933
```

Analysis:
We can see, if the object of active management is to maximize active expected return $w^T\mu$, and meanwhile, we have the active beta constraint $w^T\beta = 0$, the optimization problem can be converted to maximize expected alpha.

Question 6.2

According to the program results. The information ratio in different cases are 0.286, 0.281, 0.132, 0.132.

In this case, the influence of different object functions is tiny, while the strength of constraints plays a key role in information ratio

Question 6 Code

Listing 3: question 6 source code

```
from question5 import *

# expected return of benchmark
mub = compute_expected_return(b_array, alpha_array, beta_array, muM)
# covariance of securities
cov = compute_covariance(beta_array, omega_array, sigmaM)


def compute_expected_active_return(h, b, alpha_array, beta_array, muM):
    return compute_expected_return(h - b, alpha_array, beta_array, muM)
    # return np.dot(h-b,alpha_array + beta_array * muM)


def compute_tracking_error(h_array, b_array, cov):
    active_weight_vector = h_array - b_array
    tracking_var = \
        np.dot(np.dot(active_weight_vector, cov), active_weight_vector)
    tracking_error = np.sqrt(tracking_var)
    return tracking_error


def compute_tracking_var(h_array, b_array, cov):
    active_weight_vector = h_array - b_array
    tracking_var = \
        np.dot(np.dot(active_weight_vector, cov), active_weight_vector)
    return tracking_var


# I thought it was the supposed definition of expected alpha of the portfolio
def compute_expected_residual_return(h, b, alpha_array, beta_array, muM):
    mu_p = compute_expected_return(h, alpha_array, beta_array, muM)
    mu_b = compute_expected_return(b, alpha_array, beta_array, muM)
    beta_p = np.dot(h, beta_array)
    residual_return = mu_p - beta_p * mu_b
    return residual_return
```

```
cons6_1 = ({'type': 'eq',  # budget constraint
            'fun': lambda h: np.dot(h, eta) - 1,
            'jac': lambda h: eta
            },
           {'type': 'eq',  # tracking error constraint
            'fun':
                lambda h: compute_tracking_error(h, b_array, cov) - 0.03,

            'jac':
                lambda h: np.dot(cov, h - b_array)
                          / (compute_tracking_error(h, b_array, cov) + 1e-10)
            },
           # {'type' : 'eq',
           #  'fun' : lambda h: np.dot(h-b_array,beta_array)}
           # {'type' : 'ineq',  # long-only constraint
           #  'fun' : lambda h: h,
           #  'jac' : lambda h: np.eye(n_features)},
           # {'type': 'ineq',   # boundary constraint
           #  'fun' : lambda h: eta-h,
           #  'jac' : lambda h: -np.eye(n_features)}
           )
res6_1_0 = sp.optimize.minimize(
    lambda h: -np.dot(h, alpha_array),
    x0=eta / n_features, constraints=cons6_1,
    tol=1e-8, options={'disp': False})
h_6_1_0 = res6_1_0.x
print("########################################################################")
print("portfolio that maximize the portfolio's "
      "expected alpha under only budget constraint\n"
      "and tracking error constraint:")
print("portfolio h:" + str(h_6_1_0))
print("portfolio total weight:" + str(np.dot(h_6_1_0, eta)))
print("portfolio expected active return:"
      + str(compute_expected_active_return(h_6_1_0, b_array, alpha_array, beta_array, muM)))
print("portfolio expected active alpha:"
      + str(compute_expected_alpha(h_6_1_0 - b_array, alpha_array)))
print("portfolio expected return:"
      + str(compute_expected_return(h_6_1_0, alpha_array, beta_array, muM)))
print("benchmark expected return:"
      + str(mub))
print("portfolio standard deviation:"
      + str(compute_standard_deviation(h_6_1_0, cov)))
print("benchmark standard deviation:"
      + str(compute_standard_deviation(b_array, cov)))
print("portfolio tracking error:"
      + str(compute_tracking_error(h_6_1_0, b_array, cov)))
print("portfolio beta:"
      + str(np.dot(h_6_1_0, beta_array)))
print("portfolio expected residual return:"
      + str(compute_expected_residual_return(h_6_1_0, b_array, alpha_array, beta_array, muM)))
print("portfolio information ratio:"
      + str(compute_expected_alpha(h_6_1_0 - b_array, alpha_array)
```

```
            / compute_tracking_error(h_6_1_0, b_array, cov)))
print("######################################################################")

res6_1 = sp.optimize.minimize(
    lambda h: -compute_expected_active_return(h, b_array, alpha_array, beta_array, muM),
    x0=eta / n_features, constraints=cons6_1, tol=1e-8, options={'disp': False})
h_6_1 = res6_1.x
print("######################################################################")
print("portfolio that maximize the portfolio's expected active return "
      "under only budget constraint \n"
      "and tracking error constraint:")
print("portfolio h:" + str(h_6_1))
print("portfolio total weight:" + str(np.dot(h_6_1, eta)))
print("portfolio expected active return:"
      + str(compute_expected_active_return(h_6_1, b_array, alpha_array, beta_array, muM)))
print("portfolio expected active alpha:"
      + str(compute_expected_alpha(h_6_1 - b_array, alpha_array)))
print("portfolio expected return:"
      + str(compute_expected_return(h_6_1, alpha_array, beta_array, muM)))
print("benchmark expected return:" + str(mub))
print("portfolio standard deviation:" + str(compute_standard_deviation(h_6_1, cov)))
print("benchmark standard deviation:" + str(compute_standard_deviation(b_array, cov)))
print("portfolio tracking error:" + str(compute_tracking_error(h_6_1, b_array, cov)))
print("portfolio beta:" + str(np.dot(h_6_1, beta_array)))
print("portfolio expected residual return:"
      + str(compute_expected_residual_return(h_6_1, b_array, alpha_array, beta_array, muM)))
print("portfolio information ratio:"
      + str(compute_expected_alpha(h_6_1 - b_array, alpha_array)
            / compute_tracking_error(h_6_1, b_array, cov)))
print("######################################################################")

cons6_2 = ({'type': 'eq',  # budget constraint
            'fun': lambda h: np.dot(h, eta) - 1,
            'jac': lambda h: eta
            },
           {'type': 'eq',  # tracking error constraint
            'fun': lambda h: compute_tracking_error(h, b_array, cov) - 0.03,
            'jac':
                lambda h: np.dot(cov, h - b_array)
                          / (compute_tracking_error(h, b_array, cov) + 1e-10)
            },
           {'type': 'eq',
            'fun': lambda h: np.dot(h - b_array, beta_array)}
           )

res6_2_0 = sp.optimize.minimize(lambda h: -np.dot(h, alpha_array),
                                x0=eta / n_features,
                                constraints=cons6_2,
                                options={'disp': False})
h_6_2_0 = res6_2_0.x
print("######################################################################")
print("portfolio that maximize the portfolio's expected alpha"
      " under budget constraint,"
```

```
      " \ntracking error constraint "
      "and active market beta constraint(active_weights * beta == 0)")
print("portfolio h:" + str(h_6_2_0))
print("portfolio total weight:" + str(np.dot(h_6_2_0, eta)))
print("portfolio expected active return:"
      + str(compute_expected_active_return(h_6_2_0, b_array, alpha_array, beta_array, muM)))
print("portfolio expected active alpha:"
      + str(compute_expected_alpha(h_6_2_0 - b_array, alpha_array)))
print("portfolio expected residual return:"
      + str(compute_expected_residual_return(h_6_2_0, b_array, alpha_array, beta_array, muM)))
print("portfolio expected return:"
      + str(compute_expected_return(h_6_2_0, alpha_array, beta_array, muM)))
print("benchmark expected return:" + str(mub))
print("portfolio standard deviation:" + str(compute_standard_deviation(h_6_2_0, cov)))
print("benchmark standard deviation:" + str(compute_standard_deviation(b_array, cov)))
print("portfolio tracking error:" + str(compute_tracking_error(h_6_2_0, b_array, cov)))
print("portfolio beta:" + str(np.dot(h_6_2_0, beta_array)))
print("portfolio information ratio:"
      + str(compute_expected_alpha(h_6_2_0 - b_array, alpha_array)
            / compute_tracking_error(h_6_2_0, b_array, cov)))
print("#########################################################################")

res6_2 = sp.optimize.minimize(lambda h: -np.dot(h - b_array, mu_array),
                              x0=eta / n_features,
                              constraints=cons6_2,
                              options={'disp': False})
h_6_2 = res6_2.x
print("#########################################################################")
print("portfolio that maximize the portfolio's expected active return"
      " under budget constraint, \ntracking error constraint "
      "and active market beta constraint(active_weights * beta == 0)")
print("portfolio h:" + str(h_6_2))
print("portfolio total weight:" + str(np.dot(h_6_2, eta)))
print("portfolio expected active return:"
      + str(compute_expected_active_return(h_6_2, b_array, alpha_array, beta_array, muM)))
print("portfolio expected active alpha:"
      + str(compute_expected_alpha(h_6_2 - b_array, alpha_array)))
print("portfolio expected residual return:"
      + str(compute_expected_residual_return(h_6_2, b_array, alpha_array, beta_array, muM)))
print("portfolio expected return:"
      + str(compute_expected_return(h_6_2, alpha_array, beta_array, muM)))
print("benchmark expected return:" + str(mub))
print("portfolio standard deviation:" + str(compute_standard_deviation(h_6_2, cov)))
print("benchmark standard deviation:" + str(compute_standard_deviation(b_array, cov)))
print("portfolio tracking error:" + str(compute_tracking_error(h_6_2, b_array, cov)))
print("portfolio beta:" + str(np.dot(h_6_2, beta_array)))
print("portfolio information ratio:"
      + str(compute_expected_alpha(h_6_2 - b_array, alpha_array)
            / compute_tracking_error(h_6_2, b_array, cov)))
print("#########################################################################")
```

Problem 7

Question 7.1

What are the annual returns using the time weighted method, the Simple Deitz method, and the Modified Deitz method?

Results of 7.1 and 7.2:

```
      value  inflow  outflow    rm
date
0       520      50        0  0.10
1       560     100        0  0.08
2       600      80       20  0.05
3       540       0        0  0.12
4       480       0       10 -0.03
5       400       0       20 -0.01
6       700     250       50  0.10
7       710       0       10  0.04
8       705       0        0  0.05
9       715      10        0 -0.05
10      640      20       60 -0.08
11      670      15       15  0.07
time weighted return: -0.271693988874
simple dietz return: -0.253731343284
modified dietz return: -0.230769230769
IRR: -0.22813652495069436
```

Question 7.2

In order to solve the internal interest rate of the discounted cash flow equation, using root function in scipy.optimize package, the result was displayed in question 7.1
IRR = -0.2281
Combined with time-weighted return, Dietz return which are similar, the company did not performed well this year.

Question 7.3

Program's output:

```
      value  inflow  outflow    rm  monthly_return
date
0       520      50        0  0.10       -0.060000
1       560     100        0  0.08       -0.115385
2       600      80       20  0.05       -0.035714
3       540       0        0  0.12       -0.100000
4       480       0       10 -0.03       -0.092593
5       400       0       20 -0.01       -0.125000
6       700     250       50  0.10        0.250000
7       710       0       10  0.04        0.028571
8       705       0        0  0.05       -0.007042
9       715      10        0 -0.05        0.000000
10      640      20       60 -0.08       -0.048951
11      670      15       15  0.07        0.046875
alpha: -0.0355360897751
beta: 0.379987976611
R squared: 0.0579678204714
adjusted R squared: -0.0362353974814
Sharpe Ratio: -0.220921962044
Treynor Ratio: -0.0568523180444
IR: -0.374418784233
M squared: -0.050354719182
```

Question 7 Code

Listing 4: question 7 source code

```python
import numpy as np
import pandas as pd
import scipy as sp
import scipy.optimize

initial_value = 500
value_vector = np.array([520, 560, 600, 540, 480, 400, 700, 710, 705, 715, 640, 670])
inflow_vector = np.array([50, 100, 80, 0, 0, 0, 250, 0, 0, 10, 20, 15])
outflow_vector = np.array([0, 0, 20, 0, 10, 20, 50, 10, 0, 0, 60, 15])
rm_vector = np.array([10, 8, 5, 12, -3, -1, 10, 4, 5, -5, -8, 7]) * 0.01
n_months = 12

df = pd.DataFrame()
df['value'] = value_vector
```

```python
df['inflow'] = inflow_vector
df['outflow'] = outflow_vector
df['rm'] = rm_vector
df.index.name = "date"
print(df)


def compute_time_weighted_return(initial_value, value_vector, inflow_vector,
                                 outflow_vector, n_months):
    time_weighted_return = \
        (value_vector[0] - inflow_vector[0] + outflow_vector[0]) \
        / initial_value
    for i in range(1, n_months):
        time_weighted_return *= \
            (value_vector[i] - inflow_vector[i] + outflow_vector[i]) \
            / value_vector[i - 1]
    time_weighted_return -= 1
    return time_weighted_return


def compute_modified_monthly_return(initial_value, value_vector,
                                    inflow_vector, outflow_vector,
                                    n_months):
    modified_monthly_return = []
    modified_monthly_return.append(
        (value_vector[0] - inflow_vector[0] + outflow_vector[0])
        / initial_value)
    for i in range(1, n_months):
        modified_monthly_return.append(
            (value_vector[i] - inflow_vector[i] + outflow_vector[i])
            / value_vector[i - 1])
    modified_monthly_return = np.asarray(modified_monthly_return)
    modified_monthly_return -= 1
    return modified_monthly_return


def compute_simple_dietz_return(initial_value, end_value, inflow_vector,
                                outflow_vector):
    C_net = np.sum(inflow_vector - outflow_vector)
    return (end_value - initial_value - C_net) / (initial_value + C_net / 2)


def compute_modified_dietz_return(initial_value, end_value, inflow_vector,
                                  outflow_vector, n_months):
    C_net = np.sum(inflow_vector - outflow_vector)
    C = inflow_vector - outflow_vector
    C_avg = 0
    for i in range(n_months):
        C_avg += (n_months - i - 1) / n_months * C[i]
    return (end_value - initial_value - C_net) / (initial_value + C_avg)


time_weighted_return = \
```

```
    compute_time_weighted_return(initial_value, value_vector, inflow_vector,
                                 outflow_vector, n_months)
simple_dietz_return = \
    compute_simple_dietz_return(initial_value, value_vector[-1],
                                inflow_vector, outflow_vector)
modified_dietz_return = \
    compute_modified_dietz_return(initial_value, value_vector[-1], inflow_vector,
                                  outflow_vector, n_months)
print("time weighted return: " + str(time_weighted_return))
print("simple dietz return: " + str(simple_dietz_return))
print("modified dietz return: " + str(modified_dietz_return))


# 7.2
def compute_future_value(interest_rate, initial_value, inflow_vector,
                         outflow_vector, n_months=12):
    cash_flow = inflow_vector - outflow_vector
    future_value = (1 + interest_rate) * initial_value
    future_value = float(future_value)
    for i in range(n_months):
        future_value += cash_flow[i] * (1 + interest_rate) ** (1 - (i + 1) / n_months)
        # print(future_value)
    return future_value


sol = sp.optimize.root(lambda r:
                       compute_future_value(r, initial_value, inflow_vector,
                                            outflow_vector)
                       - value_vector[-1],
                       x0=0)
irr = float(sol.x)
print('IRR: ' + str(irr))
print("check future value: "
      + str(compute_future_value(irr, initial_value, inflow_vector, outflow_vector)))

# 7.3
import statsmodels.api as sm

modified_monthly_return = \
    compute_modified_monthly_return(initial_value, value_vector, inflow_vector,
                                    outflow_vector, n_months)
df['monthly_return'] = modified_monthly_return
print(df)

results = sm.OLS(df.monthly_return, sm.add_constant(df.rm)).fit()
# print(results.summary())
rsquared = results.rsquared
rsquared_adj = results.rsquared_adj
alpha = results.params[0]
beta = results.params[1]
omega = np.std(results.resid)
mp = np.mean(df.monthly_return)
mm = np.mean(df.rm)
```

```
std_m = np.std(df.rm)
std_p = np.std(df.monthly_return)
rF = 0
Sharpe_ratio = (mp - rF) / std_p
Treynor_ratio = (mp - rF) / beta
IR = (mp - beta * mm) / omega
msquared = mp * std_m / std_p - mm

print("alpha: " + str(alpha))
print("beta: " + str(beta))
print("R squared: " + str(rsquared))
print("adjusted R squared: " + str(rsquared_adj))
print("Sharpe Ratio: " + str(Sharpe_ratio))
print("Treynor Ratio: " + str(Treynor_ratio))
print("IR: " + str(IR))
print("M squared: " + str(msquared))
```

Problem 8

If I understand it correctly

Based on the following table:

Bond	Maturity	Coupon	Price
A	12 Months	0.0%	$997
B	12 Months	1.0%	$1,000
C	12 Months	1.2%	$1,050
D	24 Months	0.0%	$992
E	24 Months	1.3%	$1,100
F	24 Months	1.4%	$1,150
G	36 Months	0.0%	$990
H	36 Months	1.3%	$1,200
I	36 Months	1.4%	$1,280

The coupon payments are made semianually, the cash flow is listed below:

```
cash flow matrix:
[[    0.     10.     12.      0.     13.     14.      0.     13.     14.]
 [ 1000.   1010.   1012.      0.     13.     14.      0.     13.     14.]
 [    0.      0.      0.      0.     13.     14.      0.     13.     14.]
 [    0.      0.      0.   1000.   1013.   1014.      0.     13.     14.]
 [    0.      0.      0.      0.      0.      0.      0.     13.     14.]
 [    0.      0.      0.      0.      0.      0.   1000.   1013.   1014.]]
```

There are several bond that whose IRR is negative while others are tiny positive. Take bond C as an example:

$$bondprice \times (1+r)^2 = coupon1 \times (1+r) + coupon2 + facevalue$$

by substitution,

$$r = -0.0125 = -1.25\%$$

It's so wired that the bond internal rate is negative, I heard it might only happens in Japan.

Question 8.1

Use linear programming to find the least-cost portfolio of bonds that will meet the company's obligations. Show your code. State the portfolio as the number of each type of bond to be bought.

There are 2 matching strategies:

1. Matching cash flow:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & n^T p \\ \text{subject to} \quad & Cn \geq l \\ & n \geq 0 \end{aligned}$$

```
Positive directional derivative for linesearch    (Exit mode 8)
            Current function value: 1833.521707659605
            Iterations: 20
            Function evaluations: 269
            Gradient evaluations: 16
############################################################
scheme 1: match with cash flow
n vector: [ -3.08939536e-07  -3.12776161e-07  -3.78343021e-07   4.99667097e-03
  -4.44034927e-07  -5.09705053e-07  -2.99602015e-07  -5.75236945e-07
   1.42856882e+00]
number of each bond to be bought: [      0       0       0    4996       0       0       0       0 1428568]
portfolio cash flow: [   19.99993543    19.99893537    19.99994309    24.99566032    19.999956
  1448.56790116]
portfolio cost: 1.8335e+03 million dollars
portfolio profit: 1.5536e+03 million dollars
portfolio net earning: -2.7996e+02 million dollars
```

The number of bond to be bought:

Bond	A	B	C	D	E	F	G	H	I
#bond	0	0	0	4996	0	0	0	0	1428568

2. Matching cash carry-forward:

$$\begin{aligned} \underset{h}{\text{maximize}} \quad & \theta^T \pi \\ \text{subject to} \quad & \Xi\theta = l \\ & \theta \geq 0 \end{aligned}$$

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 822.4045691110894
            Iterations: 9
            Function evaluations: 153
            Gradient evaluations: 9
############################################################
scheme 2: match with cash carry-forword
n vector: [ -4.92664466e-12   2.51882178e-11   1.51821862e-02  -1.79694203e-12
   2.16163627e-11   7.01272412e-01  -1.32542359e-12   3.26154413e-11
  -3.15178439e-11]
number of each bond to be bought: [      0       0  15182       0       0 701272       0       0       0]
portfolio cash flow: [  1.00000000e+01   2.51821862e+01   9.81781377e+00   7.11090226e+02
  -1.72490769e-11  -2.45075209e-10]
portfolio cost: 8.2240e+02 million dollars
portfolio profit: 7.5609e+02 million dollars
portfolio net earning: -6.6314e+01 million dollars
```

The number of bond to be bought:

Bond	A	B	C	D	E	F	G	H	I
#bond	0	0	15182	0	0	701272	0	0	0

Question 8.2

What is the cost of the portfolio?

$$p_{port} = n^T p$$

strategy	cash flow	cash carry-forward
cost(million dollars)	1834	822
profit(million dollars)	1554	756
net earning(million dollars)	-280	-66

Analysis: the cash carry-forward strategy requires much less initial cost.
It's worth mentioning that a low cost of portfolio doesn't means profitable especially in this wired market.
We can see both the optimum strategies lose money.

Question 8.3

What is the duration of each of the bonds? (State any assumptions that you need to make).

$$D = \frac{1}{P}\sum_{k=1}^{n}\frac{k}{m}\cdot\frac{C_k}{(1+\frac{\lambda}{m})^k}$$

Since the internal interest rate in these bonds can be either positive or negative. It's reasonable to assume discounted factor (annual interest rate) equals to 0.
By substitution,

bond	maturity	coupon	price	duration
0	12	0.000	997	0.501505
1	12	0.010	1000	0.505000
2	12	0.012	1050	0.481905
3	24	0.000	992	1.512097
4	24	0.013	1100	1.399091
5	24	0.014	1150	1.340870
6	36	0.000	990	2.525253
7	36	0.013	1200	2.164583
8	36	0.014	1280	2.035156

Question 8.4

What is the duration of the portfolio?

There are 2 schemes to have duration of the portfolio
Scheme 1: treat portfolio as a bond, and use the bond duration formula:

$$D = \frac{1}{P}\sum_{k=1}^{n}\frac{k}{m}\cdot\frac{C_k}{(1+\frac{\lambda}{m})^k}$$

Scheme 2: portfolio duration equals to the weighted mean of each bond's duration.

$$D_P = \sum_{i=1}^{n}\frac{n_i p_i}{V_P}D_i$$

Using these two schemes and substitute into two portfolio, matching cash flow portfolio and matching cash carry-forward portfolio,

```
scheme 1
portfolio1's duration: 2.03374334244
portfolio2's duration: 1.32421959475
scheme 2
portfolio1's duration: 2.03374334244
portfolio2's duration: 1.32421959475
```

Normally, theses 2 schemes get the same result.

Question 8 Code

Listing 5: question 8 source code

```python
import numpy as np
import pandas as pd
import scipy as sp
import scipy.optimize

liabilities = np.array([10, 15, 20, 25, 20, 15])
df = pd.DataFrame()
df['maturity'] = [12, 12, 12, 24, 24, 24, 36, 36, 36]
df['coupon'] = np.dot([0, 1, 1.2, 0, 1.3, 1.4, 0, 1.3, 1.4], 0.01)
price_vector = np.array([997, 1000, 1050, 992, 1100, 1150, 990, 1200, 1280])
df['price'] = price_vector
df.index.name = 'bond'
print("##########################################################")
print(df)
print("##########################################################")
n_bonds = len(df.index)
n_coupon_dates = len(liabilities)


def compute_cash_flow_matrix(df, period=6, length=36):
    if (length % period == 0):
        cash_flow_matrix = np.zeros((length // period, len(df.index)))
        for i in range(len(df.index)):
```

```
            data = df.ix[i, :]
            n_coupon_date = int(data.maturity // period)
            cash_flow_matrix[:n_coupon_date, i] = data.coupon * 1000
            cash_flow_matrix[n_coupon_date - 1, i] += 1000
        return cash_flow_matrix

    else:
        print("check your period")
        return


def compute_portfolio_cash_flow(cash_flow_matrix, n_vector):
    return np.dot(cash_flow_matrix, n_vector)


cash_flow_matrix = compute_cash_flow_matrix(df)
print("############################################################")
print("cash flow matrix:")
print(cash_flow_matrix)
print("############################################################")
###########################################################################
# scheme 1
cons_1 = ({'type': 'ineq',
           'fun': lambda n: np.dot(cash_flow_matrix, n) - liabilities,
           'jac': lambda n: cash_flow_matrix},
          {'type': 'ineq',
           'fun': lambda n: n - 1e-6,
           'jac': lambda n: np.eye(n_bonds)})
res_1 = sp.optimize.minimize(lambda n: np.dot(price_vector, n),
                             x0=np.ones(n_bonds) / n_bonds,
                             constraints=cons_1,
                             options={'disp': True})
n_vector = res_1.x
print("############################################################")
print("scheme 1: match with cash flow")
print("n vector: " + str(n_vector))
print("number of each bond to be bought: " + str((n_vector * 1000000).astype(int)))
print("portfolio cash flow: " + str(np.dot(cash_flow_matrix, n_vector)))
print("portfolio cost: " + "{:.4e} million dollars".format(
    np.dot(n_vector, price_vector)))
print("portfolio profit: " + "{:.4e} million dollars".format(
    np.sum(np.dot(cash_flow_matrix, n_vector))))
print("portfolio net earning: " + "{:.4e} million dollars".format(
    np.sum(np.dot(cash_flow_matrix, n_vector)) - np.dot(n_vector, price_vector)))
print("############################################################")


###########################################################################
# scheme 2
def compute_gross_reinvestment_rates_matrix(n_dates, interest_rates_vector=None):
    if (interest_rates_vector):
        if (len(interest_rates_vector) == n_dates):
            return np.diag(interest_rates_vector)
```

```
        else:
            print('check n_dates and len(interest_rates_vector) should be equal')
            return
    else:
        return np.eye(n_dates)


R_matrix = compute_gross_reinvestment_rates_matrix(n_dates=n_coupon_dates)

J_matrix = np.zeros((n_coupon_dates, n_coupon_dates))
for i in range(n_coupon_dates - 1):
    J_matrix[i + 1, i] = 1

Y_matrix = np.dot(J_matrix, R_matrix) - np.eye(n_coupon_dates)
pi_vector = np.hstack((price_vector, np.zeros(n_coupon_dates)))
init_theta_vector = np.ones(n_bonds + n_coupon_dates) / n_bonds
constraint_matrix = np.hstack((cash_flow_matrix, Y_matrix))
# print("####################################")
# print(constraint_matrix)
# print(pi_vector)
# print(init_theta_vector)
# print(liabilities)

cons_2 = ({
              'type': 'eq',
              'fun': lambda theta:
              np.dot(constraint_matrix, theta) - liabilities,
              'jac': lambda n: constraint_matrix
          },
          {
              'type': 'ineq',
              'fun': lambda theta: theta,
              'jac': lambda theta: np.eye(n_bonds + n_coupon_dates)
          })
res_2 = sp.optimize.minimize(lambda theta: np.dot(theta, pi_vector),
                             x0=init_theta_vector,
                             constraints=cons_2,
                             options={'disp': True}
                             )
theta_vector = res_2.x
# print("theta:"+str(theta_vector))
# print("init theta:"+str(init_theta_vector))
n_vector2 = theta_vector[:n_bonds]
print("###########################################################")
print("scheme 2: match with cash carry-forword")
print("n vector: " + str(n_vector2))
print("number of each bond to be bought: "
      + str((n_vector2 * 1000000).astype(int)))
print("portfolio cash flow: " + str(np.dot(cash_flow_matrix, n_vector2)))
print("portfolio cost: " + "{:.4e} million dollars".format(
    np.dot(n_vector2, price_vector)))
print("portfolio profit: " + "{:.4e} million dollars".format(
    np.sum(np.dot(cash_flow_matrix, n_vector2))))
```

```
print("portfolio net earning: " + "{:.4e} million dollars".format(
    np.sum(np.dot(cash_flow_matrix, n_vector2))
    - np.dot(n_vector2, price_vector)))
print("##########################################################")


#############################################################################
# 8.3
def compute_bond_duration(bond_price, cash_flow_vector, n_coupons_per_year,
                          annual_interest_rate=0):
    bond_duration = 0
    for k in range(len(cash_flow_vector)):
        bond_duration += k * cash_flow_vector[k] \
                         / (1 + annual_interest_rate / n_coupons_per_year) ** k
    bond_duration /= n_coupons_per_year * bond_price
    return bond_duration


def compute_bond_portfolio_duration(n_vector, price_vector, duration_vector,
                                    portfolio_price):
    bond_portfolio_duration = \
        np.sum(n_vector * price_vector * duration_vector) / portfolio_price
    return bond_portfolio_duration


annual_interest_rate = 0
df['duration'] = np.zeros(n_bonds)
bond_duration_vector = []
for i in range(n_bonds):
    cash_flow_vector = cash_flow_matrix[:, i]
    bond_price = df.price[i]
    duration = compute_bond_duration(bond_price, cash_flow_vector,
                                     2, annual_interest_rate)
    bond_duration_vector.append(duration)

df.duration = bond_duration_vector
print(df)

#############################################################################
# 8.4
portfolio_price1 = np.dot(n_vector, price_vector)
portfolio_price2 = np.dot(n_vector2, price_vector)
portfolio_cash_flow1 = np.dot(cash_flow_matrix, n_vector)
portfolio_cash_flow2 = np.dot(cash_flow_matrix, n_vector2)

# scheme 1: using portfolio cash flow
print("scheme 1")
portfolio_duration1 = \
    compute_bond_duration(portfolio_price1, portfolio_cash_flow1,
                          2, annual_interest_rate)
portfolio_duration2 = \
    compute_bond_duration(portfolio_price2, portfolio_cash_flow2,
                          2, annual_interest_rate)
```



```python
print("portfolio1's duration: " + str(portfolio_duration1))
print("portfolio2's duration: " + str(portfolio_duration2))

# scheme 2: using each bonds cash flows
print("scheme 2")
portfolio_duration11 = \
    compute_bond_portfolio_duration(n_vector, price_vector,
                                    bond_duration_vector,
                                    portfolio_price1)
portfolio_duration22 = \
    compute_bond_portfolio_duration(n_vector2, price_vector,
                                    bond_duration_vector,
                                    portfolio_price2)
print("portfolio1's duration: " + str(portfolio_duration11))
print("portfolio2's duration: " + str(portfolio_duration22))
```