**投资学案例分析作业(附Python代码实现)**

**金融工程 杨宸宇 2016301550186**

**说明：**

**本人使用mac电脑的Excel求解最优化问题时，发现Excel的Solver插件无法使用，实在无法用本机的Excel实现资产配置设计，所以采用Python语言进行实现(附录从第五页开始)**

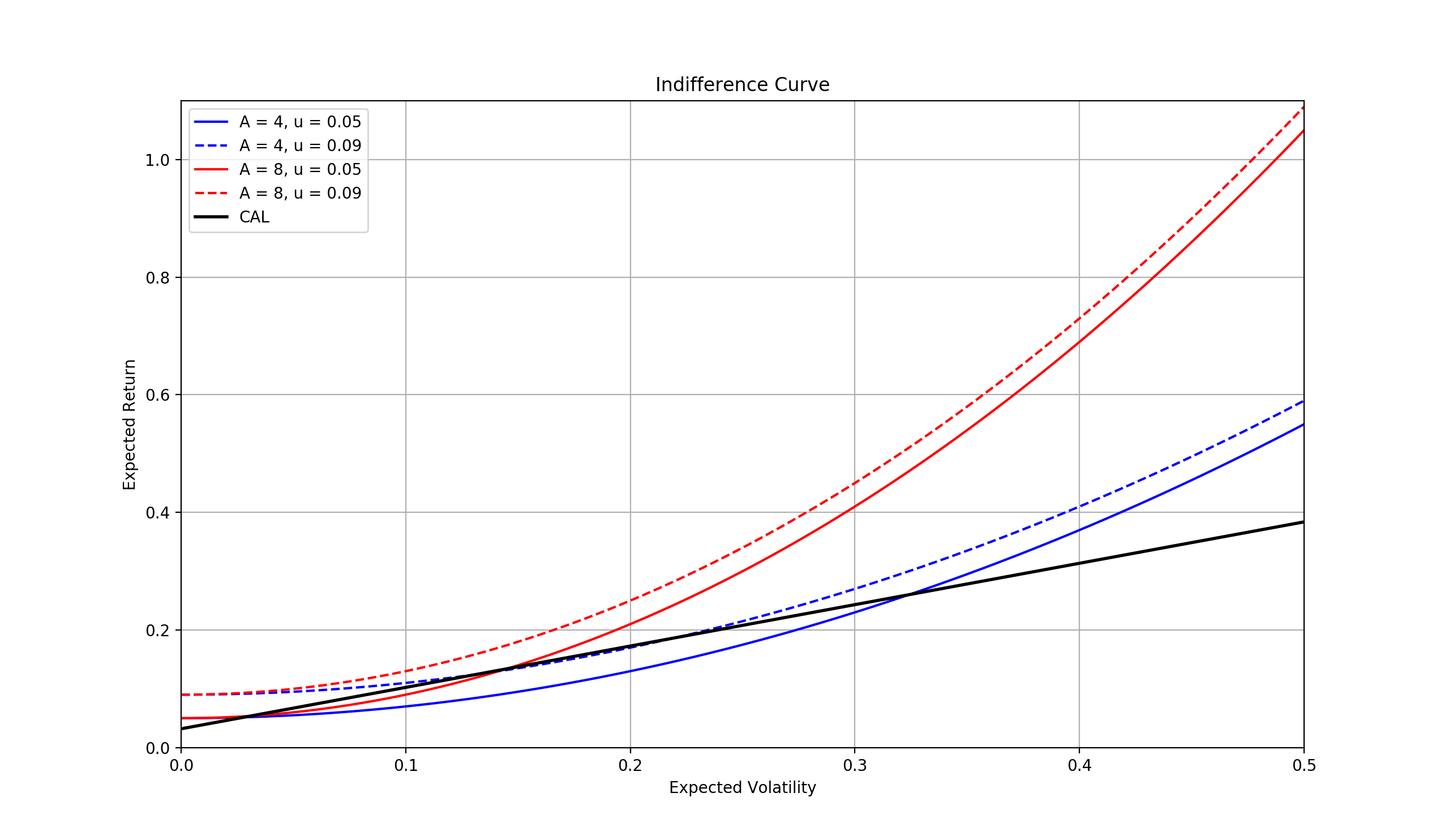
1.

当前的资产配置并不是最优配置(图和表格见第二题)，针对不同的风险偏好的医院，我选取了4种情况对风险资产配置进行举例：

首先运用方法：

求出了在A = 8时，风险资产配比最优为73.23%， 在A = 16时， 风险资产配比最优为36.63%

接着选择了A = 4和8，u = 0.05和0.09的情况绘制了CAL和无差异曲线，如图所示：

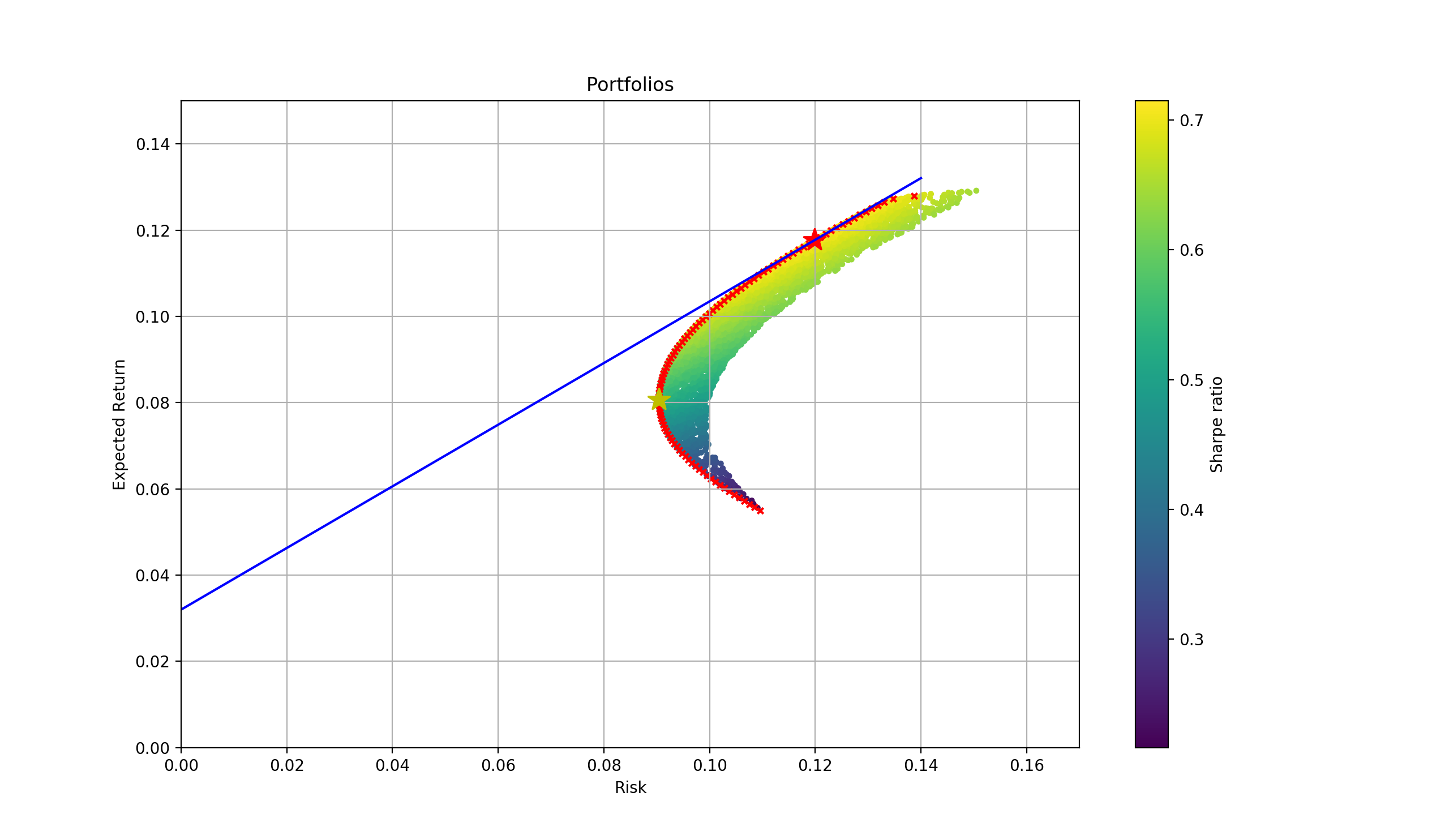


2.

本题首先的图像首先我用蒙特卡洛方法模拟了10000次随机取样，将每次资产的预期风险与预期收益的定画在了图上，并通过颜色深度区分Sharpe的大小，

接着通过凸优化求出了资产配置的有效边界(图上红色叉)，最大Sharpe的点(图上红色星位置)，最小方差的点(图上黄色星位置)，本题的最优组合分配比例及图像如下所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Portfolio | Expected Ret. | Stdev | US Equity | Foreign Equity | Bonds |
| Max\_Sharpe | 11.77% | 11.99% | 40.64% | 47.06% | 12.30% |
| Min\_Variance | 8.07% | 9.04% | 4.18% | 33.58% | 62.24% |



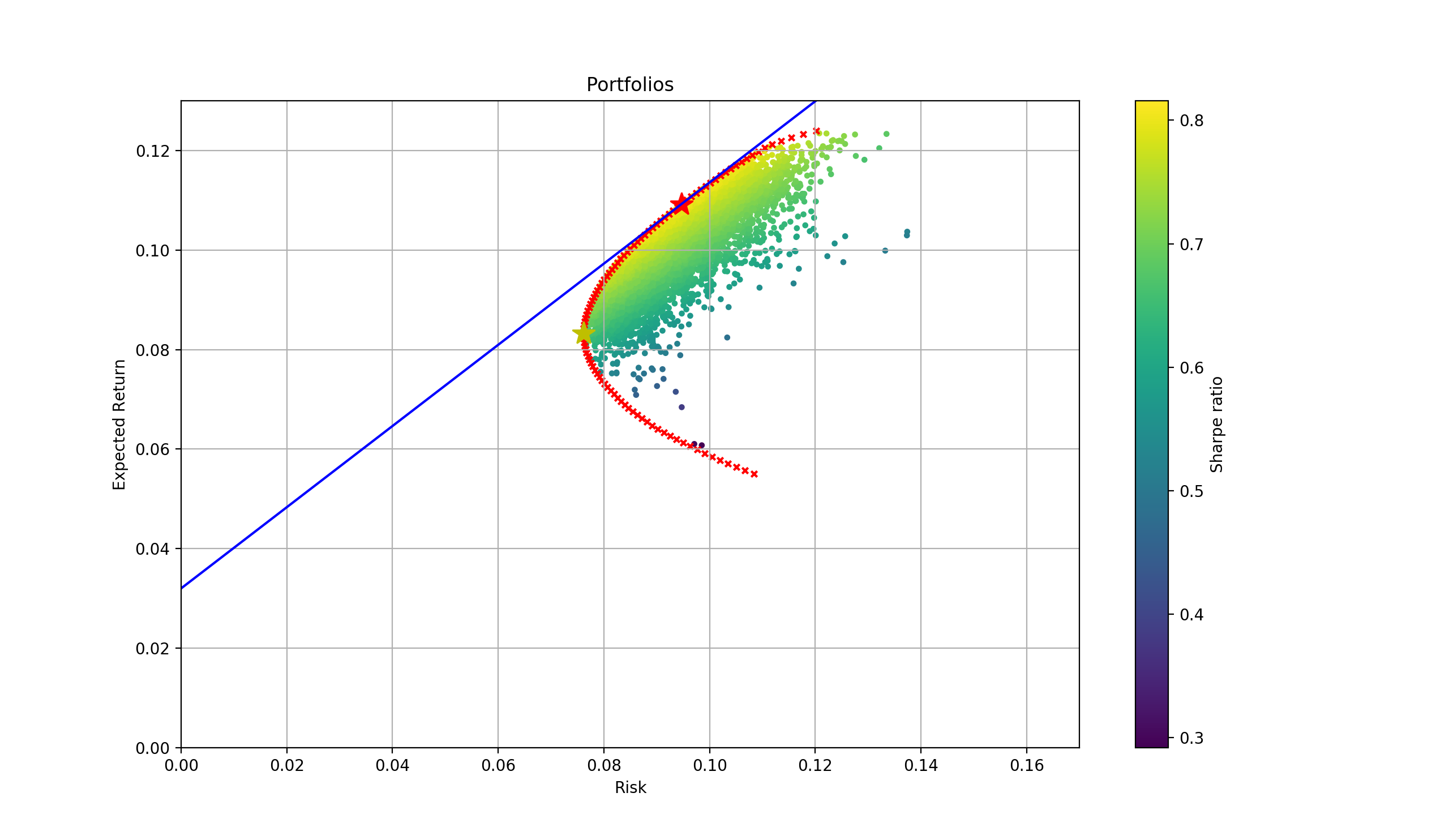
通过观察本图的最优Sharpe处的Portfolio中各个资产比例，我们可以明确的发现LTP的最优配置情况和报告中的配置情况不相符合，由此可见报告中并不是最优资产组合。

3.

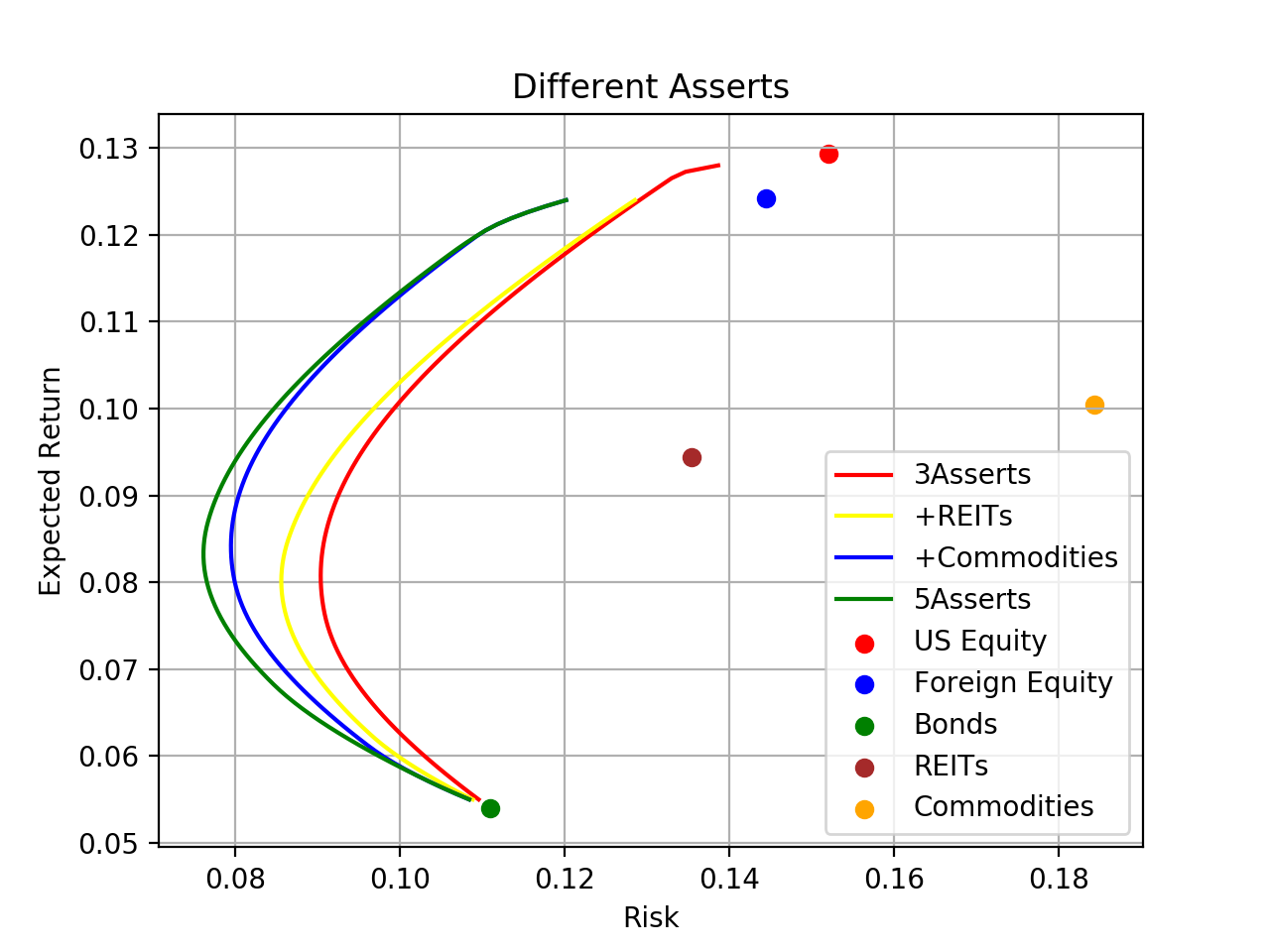
本题首先的图像首先我用蒙特卡洛方法模拟了10000次随机取样，将每次资产的预期风险与预期收益的定画在了图上，并通过颜色深度区分Sharpe的大小。

接着通过凸优化求出了资产配置的有效边界(图上红色叉)，最大Sharpe的点(图上红色星位置)，最小方差的点(图上黄色星位置)，本题的最优组合分配比例及图像如下所示：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Portfolio | Expected Ret. | Stdev | US Equity | Foreign Equity | Bonds | REITs | Commodities |
| Max\_Sharpe | 10.92% | 9.46% | 24.37% | 30.84% | 10.82% | 0.09887556 | 24.09% |
| Min\_Variance | 8.33% | 7.61% | 6.16E-18 | 18.71% | 44.18% | 18.13% | 18.97% |



之后，将原始三种资产组合，添加REITs的组合，添加commodities的组合以及五种资产组合的risk-return曲线进行对比，画在同一幅图中，如图所示：

****

我们可以发现整体上新的资产配置优于原始的三种资产的配置组合，其中Bonds能很好的降低整体风险，其他的资产具有相对较好的预期收益，如US Equity，也有风险相对较高的Commodities，综合考虑还是5种资产的配置组合效果最佳。

**附录：代码**

(Python)

# 投资学案例分析作业

# 作者：杨宸宇

# 学号：2016301550186

# 调用所需的包

import os

import pandas as pd

import numpy as np

from sympy import Symbol

from sympy.solvers import solve

from scipy.optimize import minimize

import matplotlib.pyplot as plt

# 第一题-----------------------------------------------

# 导入准备好的excel数据, 单独赋值

os.chdir(path = r'/Users/mac/Desktop')

data = pd.read\_excel('Investment.xlsx')

data.set\_index(keys=data.iloc[:,0], inplace = True)

# 转换为array方便后面运算

expected\_return = np.array(data.iloc[0:3, 1])

standard\_variance = np.array(data.iloc[0:3, 2])

correlation = np.array(data.iloc[0:3, 3:6])

# 计算协方差矩阵

corvariance = standard\_variance \* correlation \* standard\_variance\

        .reshape(3, 1)

# 由题目知

weight = np.array([0.55, 0.3, 0.15])

STP\_expected\_return = 0.032

# 计算LTP的预期收益率和波动性

LTP\_expected\_return = np.sum(weight \* expected\_return)

LTP\_expected\_volatility = np.sqrt(np.dot(weight.T, \

    np.dot(corvariance, weight)))

# 最优组合配比

def opt(bias):

    return (LTP\_expected\_return - STP\_expected\_return) \

                / (bias \* np.square(LTP\_expected\_volatility))

# bias = 16

y1 = opt(16)

# bias = 8

y2 = opt(8)

# 画无差异曲线取 u = 0.05 和 u = 0.09, A = 4(bias) 和 A = 8(bias)

# A = 4, u = 0.05

A = 4

u = 0.05

standard\_variances = np.linspace(0, 0.5, 100)

expected\_returns = []

for std in standard\_variances:

    x = Symbol('x')

    expected\_return = solve(x - 0.5 \* A \* np.square(std) - u, x)[0]

    expected\_returns.append(expected\_return)

standard\_variances = np.array(standard\_variances)

expected\_returns = np.array(expected\_returns)

plt.plot(standard\_variances, expected\_returns, \

            label = 'A = 4, u = 0.05', c = 'blue')

# A = 4, u = 0.09

A = 4

u = 0.09

standard\_variances = np.linspace(0, 0.5, 100)

expected\_returns = []

for std in standard\_variances:

    x = Symbol('x')

    expected\_return = solve(x - 0.5 \* A \* np.square(std) - u, x)[0]

    expected\_returns.append(expected\_return)

standard\_variances = np.array(standard\_variances)

expected\_returns = np.array(expected\_returns)

plt.plot(standard\_variances, expected\_returns, \

            label = 'A = 4, u = 0.09', c = 'blue', linestyle = '--')

# A = 8, u = 0.05

A = 8

u = 0.05

standard\_variances = np.linspace(0, 0.5, 100)

expected\_returns = []

for std in standard\_variances:

    x = Symbol('x')

    expected\_return = solve(x - 0.5 \* A \* np.square(std) - u, x)[0]

    expected\_returns.append(expected\_return)

standard\_variances = np.array(standard\_variances)

expected\_returns = np.array(expected\_returns)

plt.plot(standard\_variances, expected\_returns, \

            label = 'A = 8, u = 0.05', c = 'red')

# A = 8, u = 0.09

A = 8

u = 0.09

standard\_variances = np.linspace(0, 0.5, 100)

expected\_returns = []

for std in standard\_variances:

    x = Symbol('x')

    expected\_return = solve(x - 0.5 \* A \* np.square(std) - u, x)[0]

    expected\_returns.append(expected\_return)

standard\_variances = np.array(standard\_variances)

expected\_returns = np.array(expected\_returns)

plt.plot(standard\_variances, expected\_returns, \

            label = 'A = 8, u = 0.09', c = 'red', linestyle = '--')

# CAL

k = (LTP\_expected\_return - STP\_expected\_return) / LTP\_expected\_volatility

b = 0.032

y = 0.5 \* k + b

plt.plot([0, 0.5], [0.032, y], label = 'CAL', linewidth = 2,c = 'black')

plt.axis([0, 0.5, 0, 1.1])

plt.grid(True)

plt.xlabel('Expected Volatility')

plt.ylabel('Expected Return')

plt.title('Indifference Curve')

plt.legend()

plt.show()

# 第二题-----------------------------------------------

# 导入准备好的excel数据, 单独赋值

os.chdir(path = r'/Users/mac/Desktop')

data = pd.read\_excel('Investment.xlsx')

data.set\_index(keys=data.iloc[:,0], inplace = True)

# 转换为array方便后面运算

expect\_return = np.array(data.iloc[0:3, 1])

standard\_variance = np.array(data.iloc[0:3, 2])

correlation = np.array(data.iloc[0:3, 3:6])

# 计算协方差矩阵

corvariance = standard\_variance \* correlation \* standard\_variance\

        .reshape(3, 1)

# 采用蒙特卡洛方法来来绘制资产组合的分布图

expect\_returns = []

risks = []

for i in range(10000):

    weights = np.random.random(3)

    # 随机生成资产权重

    weights /= np.sum(weights)

    por\_return = np.sum(weights \* expect\_return)

    por\_risk = np.sqrt(np.dot(weights.T , np.dot(corvariance , weights)))

    expect\_returns.append(por\_return)

    risks.append(por\_risk)

# matplotlib.pyplot倾向于np.array格式，因此进行转换

expect\_returns = np.array(expect\_returns)

risks = np.array(risks)

# 无风险利率定为3.2%(文章中2005年STP的收益率)

risk\_free\_rate = 0.032

plt.scatter(risks, expect\_returns, \

    c = (expect\_returns - risk\_free\_rate) / risks, marker = 'o', s = 8)

plt.grid(True)

plt.xlabel('Risk')

plt.ylabel('Expected Return')

plt.title('Portfolios')

plt.colorbar(label = 'Sharpe ratio')

# 找出Sharpe最大的点，利用Scipy.optimize.minimize进行凸优化

max\_Sharpe = lambda x: - ((np.sum(x \* expect\_return) - risk\_free\_rate) /\

    np.sqrt(np.dot(x.T, np.dot(corvariance, x))))

cons = {'type':'eq', 'fun':lambda x: np.sum(x) - 1}

bnds = tuple((0,1) for x in range(3))

x0 = np.array([1/3, 1/3, 1/3])

# 将优化结果储存到result中

result = minimize(max\_Sharpe, x0, method = 'SLSQP', bounds = bnds, \

    constraints = cons)

weight = result['x']

expected\_return = np.sum(weight \* expect\_return)

risk = np.sqrt(np.dot(weight.T , np.dot(corvariance , weight)))

# A点为Sharpe值最大的资产组合在图中的坐标

A = (risk, expected\_return)

# 画出Sharpe最大的点

plt.plot(risk, expected\_return, 'r\*', markersize = 15.0)

# 为了美观确定坐标轴的刻度范围

plt.axis([0, 0.17, 0, 0.15])

# 找出Risk最低的点

min\_risk = lambda x: np.sqrt(np.dot(x.T, np.dot(corvariance, x)))

cons = {'type':'eq', 'fun':lambda x: np.sum(x) - 1}

bnds = tuple((0,1) for x in range(3))

x0 = np.array([1/3, 1/3, 1/3])

result = minimize(min\_risk, x0, method = 'SLSQP', bounds = bnds, \

    constraints = cons)

weight = result['x']

expected\_return = np.sum(weight \* expect\_return)

risk = np.sqrt(np.dot(weight.T , np.dot(corvariance , weight)))

# 画出Risk最低的点

plt.plot(risk, expected\_return, 'y\*', markersize = 15.0)

# 寻找并画出有效边界

expected\_returns = np.linspace(0.055 ,0.128, 100)

risks = []

for i in expected\_returns:

    cons = ({'type':'eq', 'fun':lambda x: np.sum(x) - 1},\

        {'type':'eq', 'fun':lambda x: np.sum(x \* expect\_return) - i})

    result = minimize(min\_risk, x0, method = 'SLSQP', \

        constraints = cons, bounds = bnds)

    risks.append(result['fun'])

plt.scatter(risks, expected\_returns, c = 'red', marker = 'x', s = 14)

# 画出资产配置线，理论上应该凸优化求出最大k，但因为之前求了Sharpe最大组合，因此选择直接画

k = (A[1] - 0.032) / A[0]

b = 0.032

x = 0.14

y = k \* x + b

plt.plot([0, 0.14], [0.032, y], c = 'blue')

# 显示图像

plt.show()

# 第三题-----------------------------------------------

# 导入准备好的excel数据,单独赋值,文件位置在本机的desktop处

os.chdir(path = r'/Users/mac/Desktop')

data = pd.read\_excel('Investment.xlsx')

data.set\_index(keys=data.iloc[:,0], inplace = True)

# 转换为array方便后面运算

expect\_return = np.array(data.iloc[:, 1])

standard\_variance = np.array(data.iloc[:, 2])

correlation = np.array(data.iloc[:, 3:])

# 计算协方差矩阵

corvariance = standard\_variance \* correlation \* standard\_variance\

        .reshape(5, 1)

# 采用蒙特卡洛方法来来绘制资产组合的分布图

expect\_returns = []

risks = []

for i in range(10000):

    weights = np.random.random(5)

    weights /= np.sum(weights)

    por\_return = np.sum(weights \* expect\_return)

    por\_risk = np.sqrt(np.dot(weights.T , np.dot(corvariance , weights)))

    expect\_returns.append(por\_return)

    risks.append(por\_risk)

expect\_returns = np.array(expect\_returns)

risks = np.array(risks)

# 无风险利率定为3.2%(文章中2005年STP的收益率)

risk\_free\_rate = 0.032

plt.scatter(risks, expect\_returns, \

    c = (expect\_returns - risk\_free\_rate) / risks, marker = 'o', s = 8)

plt.grid(True)

plt.xlabel('Risk')

plt.ylabel('Expected Return')

plt.title('Portfolios')

plt.colorbar(label = 'Sharpe ratio')

# 找出Sharpe最大的点

max\_Sharpe = lambda x: - ((np.sum(x \* expect\_return) - risk\_free\_rate) /\

    np.sqrt(np.dot(x.T, np.dot(corvariance, x))))

cons = {'type':'eq', 'fun':lambda x: np.sum(x) - 1}

bnds = tuple((0,1) for x in range(5))

x0 = np.array([0.2, 0.2, 0.2, 0.2, 0.2])

result = minimize(max\_Sharpe, x0, method = 'SLSQP', bounds = bnds, \

    constraints = cons)

weight = result['x']

expected\_return = np.sum(weight \* expect\_return)

risk = np.sqrt(np.dot(weight.T , np.dot(corvariance , weight)))

A = (risk, expected\_return)

# 画出Sharpe最大的点

plt.plot(risk, expected\_return, 'r\*', markersize = 15.0)

plt.axis([0, 0.17, 0, 0.13])

# 找出Risk最低的点

min\_risk = lambda x: np.sqrt(np.dot(x.T, np.dot(corvariance, x)))

cons = {'type':'eq', 'fun':lambda x: np.sum(x) - 1}

bnds = tuple((0,1) for x in range(5))

x0 = np.array([0.2, 0.2, 0.2, 0.2, 0.2])

result = minimize(min\_risk, x0, method = 'SLSQP', bounds = bnds, \

    constraints = cons)

weight = result['x']

expected\_return = np.sum(weight \* expect\_return)

risk = np.sqrt(np.dot(weight.T , np.dot(corvariance , weight)))

# 画出Risk最低的点

plt.plot(risk, expected\_return, 'y\*', markersize = 15.0)

# 找出并画出有效边界

expected\_returns = np.linspace(0.055 ,0.124, 100)

risks = []

for i in expected\_returns:

    cons = ({'type':'eq', 'fun':lambda x: np.sum(x) - 1},\

        {'type':'eq', 'fun':lambda x: np.sum(x \* expect\_return) - i})

    result = minimize(min\_risk, x0, method = 'SLSQP', \

        constraints = cons, bounds = bnds)

    risks.append(result['fun'])

plt.scatter(risks, expected\_returns, c = 'red', marker = 'x', s = 14)

# 画出资产配置线

k = (A[1] - 0.032) / A[0]

b = 0.032

x = 0.12

y = k \* x + b

plt.plot([0, 0.12], [0.032, y], c = 'blue')

# 显示图像

plt.show()