ECON 577 Programming Assignment 1

Stanley Hong

September 18, 2023

Problem: Part 1. You have been provided a dataset called ETF Time Series Index Returns.csv. This file contains total return factors for 5 exchange traded funds (ETFs):

- IVV iShares Core SP 500 Index ETF
- GOVT iShares US Treasury Bond ETF
- IAU iShares Gold Trust ETF
- HYG iShares iBoxx High Yield ETF
- EEM iShares MSCI Emerging Market Equity ETF

The dataset contains total return factors, which are indexes that reflect both the prices as well as any dividends paid and account for corporate actions such as stock splits. Converting total return factors into returns means that your returns will reflect both the price changes and dividends paid. If you used raw prices, you would not capture the dividends. Read this dataset using the following code snippet:

```
df = pd.read_csv('ETF Time Series Index
    Returns.csv',index_col=0,skiprows=6,names=['IVV','GOVT','IAU','HYG','EEM'],nr ows=119)
```

(a) Convert the total return factors into monthly total returns using the `pct_change` function.

(b) Create an index for each time series starting at 1 so that all time-series are on the same scale. Using the `cum_prod` function will be helpful in doing this.

(c) Plot your indexes over time using `plt.plot`. What can you say about the relative performance of each ETF?

(d) Compute the covariance matrix and correlation matrix of returns using the `.cov()` and `.corr()` functions. Look at the correlation matrix. What can you say about how the different asset classes are historically correlated? Do you notice something interesting about GOVT and IVV?

(e) Compute the annualized standard deviations of each ETF using the covariance matrix. You can do this by using `np.sqrt` and `np.diagonal`. Remember that to annualize a standard deviation you need to multiply by $\sqrt{12}$. What can you say about the relative volatilities of each ETF? Which has been the highest risk? The lowest risk?

Sol. As this problem is not graded, I simply present my code without explanation.

```
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

###PART 1#####################################################################

df = pd.read_csv('ETF Time Series Index Returns.csv',
            index_col=0,skiprows=6,names=['IVV','GOVT','IAU','HYG','EEM'],nrows=119)
# print(df)

monthly_returns = df.pct_change()
growth_factors = 1 + monthly_returns
indexed_series = growth_factors.cumprod().fillna(1)
print(indexed_series)
for column in indexed_series.columns:
   plt.plot(indexed_series.index, indexed_series[column], label=column)
plt.show()

cov_matrix = monthly_returns.cov()
print(cov_matrix)
corr_matrix = monthly_returns.corr()
print(corr_matrix)

variances = np.diagonal(cov_matrix)
monthly_std_dev = np.sqrt(variances)
annualized_std_dev = monthly_std_dev * np.sqrt(12)
print(annualized_std_dev)
```

Problem: Part 2a. Compute the global minimum variance portfolio. Use (2.50) in Campbell to compute the weights of the GMVP. Show your results.

(1) Check the your weights sum to one.

(2) What asset class is the greatest weights in the GMVP. Is this intuitive?

(3) Why doesnt the GMVP hold only the lowest-risk asset class?

(4) Use 2.55 to calculate the variance and standard deviation of the GMVP. Show your results.

Sol. Consider the following code.

```
1  ###2.a###
2
3  cov_matrix = cov_matrix * 12
4  print(cov_matrix)
5
6  evalues = np.linalg.eig(cov_matrix)
7  print(evalues) # All eigenvalues are positive, implying psd
8
9  inv_cov_matrix = np.linalg.inv(cov_matrix)
10 ones_vector = np.ones(shape=(inv_cov_matrix.shape[0], 1))
11 numerator = np.dot(inv_cov_matrix, ones_vector)
12 denominator = np.dot(ones_vector.T, numerator)
13 weights = numerator / denominator
14
15 print(weights)
16 print('weight sum', np.sum(weights))
17 print('variance', 1/denominator)
18 print('sd', np.sqrt(1/denominator))
```

The weights (line 15) are as follows:

```
[[0.0468329] [0.79274642] [-0.06333758] [0.23877981] [-0.01502155]]
```

The sum of weights (line 16) is indeed unity.

Out of all classes, the second class, GOVT, has the greatest weights. Considering it has the least return, it is likely some sort of government-issued bond (hence the name of GOVT), and therefore it makes sense for the least-variance portfolio to include it. However, it doesn't only hold the lowest-risk asset class as the variance of GOVT is nonzero. Therefore it can be hedged with another risky asset.

The variance and standard deviation are calculated in lines 17 and 18. The results are as follows:

```
variance [[0.00079702]] sd [[0.02823146]]
```

Problem: Part 2b. Compute a minimum variance portfolio.

(a) Set your targeted expected return to 0.05.

(b) Calculate A, B, C, D, λ_1, λ_2 as defined in (2.44-2.46). Show youre results.

(c) Calculate the MVP using (2.43). Show your result. Check that your weights sum to one.

(d) Using 2.47 calculate the variance and standard deviation of the MVP. Show your result.

(e) Characterize the differences between this portfolio and the GMVP.

Sol. Consider the following code.

```
1  ###2.b###
2
3  annualized_mean_returns_list = [0.06, 0.016, 0.03, 0.032, 0.071]
4  annualized_mean_returns = np.array(annualized_mean_returns_list).reshape(-1, 1)
5  expected_return = 0.05
6
7  A = np.dot(np.dot(annualized_mean_returns.T, inv_cov_matrix), annualized_mean_returns)
8  B = np.dot(np.dot(annualized_mean_returns.T, inv_cov_matrix), ones_vector)
9  C = denominator
10 D = A * C - np.square(B)
11 print('A', A)
12 print('B', B)
13 print('C', C)
14 print('D', D)
15
16 lambda1 = (C * expected_return - B) / D
17 lambda2 = (A - B * expected_return) / D
18 print('lambda1', lambda1)
19 print('lambda2', lambda2)
20
21 weights2 = lambda1 * np.dot(inv_cov_matrix, annualized_mean_returns) + lambda2 *
       np.dot(inv_cov_matrix, ones_vector)
22 print(weights2)
23 print('weight sum', np.sum(weights2))
24 print('variance', (A - 2 * B * expected_return + C * np.square(expected_return)) / D)
25 print('sd', np.sqrt((A - 2 * B * expected_return + C * np.square(expected_return)) / D))
```

The alphabets are as follows.

```
A [[0.64324644]] B [[25.30467133]] C [[1254.68087696]] D [[166.74261428]] lambda1
[[0.22447395]] lambda2 [[-0.00373022]]
```

The MVP weights (line 22) are as follows:

```
[[0.52311915] [0.64782759] [0.02843804] [-0.55258431] [0.35319952]]
```

The weights do not sum to exact unity, but it is likely due to numerical approximation errors as the difference is in the degree of 10^{-15}. As for variance and standard deviation (lines 24-25), the results are as follows.

```
variance [[0.00749347]] sd [[0.08656486]]
```

We see that although MVP has a higher expectation, it also comes with a much higher variance, a trade-off.

Problem: Part 2c.

(a) Compute the "mutual fund" in (2.57), which when combined with the GMVP produces the same MVP in part b. Show your result. Make sure that the weights produced by 2.57 are identical to the MVP in part b. If not, you did something wrong.

(b) What can you conclude about any portfolio on the efficient frontier from this result?

Sol. Consider the following code.

```
###2.c###

mutualfundnum = np.dot(inv_cov_matrix, annualized_mean_returns)
mutualfunddenom = np.dot(np.dot(ones_vector.T, inv_cov_matrix), annualized_mean_returns)
mutualfund = mutualfundnum / mutualfunddenom
print(mutualfund)
weights_new = lambda1 * B * mutualfund + lambda2 * C * weights
print(weights_new)
```

The mutual fund takes weights in the following:

```
[[ 0.13068258] [0.76723362] [-0.04718058] [0.09946101] [0.04980337]]
```

And by combining with the GMVP, the weights produced by 2.57 is indeed identical to the MVP. Specifically, line 8 prints out

```
[[0.52311915] [0.64782759] [0.02843804] [-0.55258431] [0.35319952]]
```

From the mutual fund theorem and the above example, we could conclude that all portfolios on the efficient frontier could be represented by a linear combination between the GMVP and the single mutual fund portfolio.

Problem: Part 2d. Compute the maximum Sharpe ratio portfolio.

(a) Assume a risk-free rate of 0.005. Hence, the targeted excess return is 0.045.

(b) Calculate the weights on the MSRP using (2.61). Show your result.

(c) How does this portfolio differ from the MVP in part b? Note the fact that the weights are greater than one in the MSRP. What is the interpretation of this?

(d) Use 2.64 to compute the variance and standard deviation of the MSRP. Show your result.

(e) Show that the total expected return of the MSRP is identical to that of the MVP in part b.

(f) Compare the standard deviations of the MSRP and the MVP. What can you say about the efficiency of the MSRP relative to the MVP?

Sol. Consider the following code.

```
###2.d###

riskfree_return = 0.005
excess_return = annualized_mean_returns - riskfree_return * ones_vector
E = np.dot(np.dot(excess_return.T, inv_cov_matrix), excess_return)
```

```
6  lambda1_sharpe = (expected_return - riskfree_return) / E
7  sharpe_weights = lambda1_sharpe * np.dot(inv_cov_matrix, excess_return)
8  print(sharpe_weights)
9
10 variance_sharpe = np.square(lambda1_sharpe) * E
11 sd_sharpe = np.sqrt(variance_sharpe)
12 print("variance: ", variance_sharpe)
13 print("sd:", sd_sharpe)
14
15 expected_return_MVP = np.dot(weights2.T, annualized_mean_returns)
16 expected_return_MSRP = np.dot(sharpe_weights.T, annualized_mean_returns)
17 print("Expected Return for MVP:", expected_return_MVP)
18 print("Expected Return for MSRP:", expected_return_MSRP)
```

The MSRP weights (line 8) is as follows:

`[[ 0.32162999] [1.54153977] [-0.08502712] [0.1087585] [0.14458502]]`

The portfolio puts great emphasis on the GOVT asset. Specifically it puts a > 1 weight on the GOVT, which means that the assets are leveraged, as the investor would borrow money to invest in assets for the highest Sharpe ratio. The variance and standard deviation (lines 12-13) are as follows:

`variance [[0.00480351]] sd [[0.06930736]]`

The expected returns (lines 17-18) are as follows:

`Expected Return for MVP: [[0.05]] Expected Return for MSRP: [[0.05515743]]`

We see that the expected return of MSRP is actually greater, but that is because we considered the additional investment. In fact, the *exces return* of the MSRP portfolio is `0.045`, aligning with expectation. The standard deviation of MSRP is actually lower than MVP. This implies that the MSRP portfolio is more efficient relative to the MVP portfolio.

Problem: Extra Credit. Generate the mean-variance efficient frontier which shows the minimum standard deviation for each level of expected return. Youll do this using a for loop to loop across targeted expected returns and then employ (2.43) in your text to generate the optimal portfolio. To this properly, youll need to define functions for A,B,C,D,Lambda1, and Lambda2. Below is an example of a function for A:

```
def fncA(r,V):
    A = r.transpose() @ V @ r
   return A
```

Do the same thing for the others and then reference these functions inside your for loop. Also create a function for the portfolio variance using (2.47). This is a more elegant way of programming rather than writing each function inside the for loop. Now you can simply call a pre-defined function. Set your range of expected returns between 2.1% and 7% in increments of 0.1%. The following line will generate the vector of expected returns:

```
er_vect = np.arange(.021,max(exp_ret),0.001)
```

Loop across `er_vect` generating the optimal portfolio, expected return and portfolio standard deviation. Store the expected return and portfolio standard deviation in an array and plot them in a scatterplot.

Sol. Consider the following code.

```
def calc_A(annualized_mean_returns, inv_cov_matrix):
    return np.dot(np.dot(annualized_mean_returns.T, inv_cov_matrix), annualized_mean_returns)

def calc_B(annualized_mean_returns, inv_cov_matrix, ones_vector):
    return np.dot(np.dot(annualized_mean_returns.T, inv_cov_matrix), ones_vector)

def calc_C(inv_cov_matrix, ones_vector):
    return np.dot(ones_vector.T, np.dot(inv_cov_matrix, ones_vector))

def calc_D(A, B, C):
    return A * C - B**2

def calc_Lambda1(expected_return, B, C, D):
    return (C * expected_return - B) / D

def calc_Lambda2(A, B, expected_return, D):
    return (A - B * expected_return) / D

def portfolio_variance(A, B, expected_return, C, D):
    return (A - 2 * B * expected_return + C * expected_return**2) / D


er_vect = np.arange(0.021, 0.071, 0.001)

expected_returns = []
std_devs = []

for expected_return in er_vect:
    A = calc_A(annualized_mean_returns, inv_cov_matrix)
```

```
    B = calc_B(annualized_mean_returns, inv_cov_matrix, ones_vector)
    C = calc_C(inv_cov_matrix, ones_vector)
    D = calc_D(A, B, C)
    Lambda1 = calc_Lambda1(expected_return, B, C, D)
    Lambda2 = calc_Lambda2(A, B, expected_return, D)

    weights = Lambda1 * np.dot(inv_cov_matrix, annualized_mean_returns) + Lambda2 *
        np.dot(inv_cov_matrix, ones_vector)
    var = portfolio_variance(A, B, expected_return, C, D)
    expected_returns.append(expected_return)
    std_devs.append(np.sqrt(var))

plt.scatter(std_devs, expected_returns, c='blue')
plt.xlabel('Standard Deviation')
plt.ylabel('Expected Return')
plt.title('Efficient Frontier')
plt.grid(True)
plt.show()
```

The graph is as follows, aligning with expectation.