Lecture Note 2: Optimal Portfolios and Matrices

Espen Sirnes

2024-09-18

Table of contents

This lecture explores the strategic behavior of an investor in the stock market, particularly under the assumption of risk aversion, as discussed in the previous note on utility theory. Risk lovers generally prefer the most risky assets, while risk-neutral investors opt for assets with the highest returns.

In contrast, a risk-averse investor seeks to maximize returns without disproportionately increasing volatility, typically measured as variance.

1 Optimal Portfolio with One Asset

Although it may seem unusual, we begin by considering a portfolio comprising a single asset, setting the stage for more complex scenarios involving multiple assets and matrix algebra.

Assume the return on an asset next year, x, follows a normal distribution $N(mu, \sigma^2)$, and let r represent the risk-free interest rate. If investors must finance their entire investment through borrowing, the initial investment a will accrue to x next year, offset by interest expenses ar. The net value or wealth in the next period, W_1, is then given by:

$$W_1 = a(x - r)$$

Considering an investor with a Constant Absolute Risk Aversion (CARA) utility function, the utility of W_1 is:

$$u(W_1) = -e^{-\pi W_1}$$

where π is the CARA coefficient. Given that W_1 is a linear transformation of a normally distributed variable, it too follows a normal distribution. Consequently, $u(W_1)$ is log-normally distributed. This leads to the expected utility U as:

$$U = \mathbb{E}[u(W_1)] = -e^{-\pi \mathbb{E}[W_1] - \frac{\pi}{2} \text{var}(W_1)}$$

Key conclusions include: 1. Increased uncertainty regarding future wealth ($\text{var}(W_1)$) diminishes utility. 2. Higher risk aversion (π) amplifies the adverse impact of uncertainty.

Calculating the expected wealth and its variance yields:

$$\mathbb{E}[W_1] = a \cdot (mu - r)$$

$$\text{var}(W_1) = a^2\sigma^2$$

To maximize expected utility, the investor solves:

$$U = -e^{-\pi(a\cdot(mu-r)-\frac{\pi}{2}a^2\sigma^2)}$$

w.r.t. a. Simplifying the utility function, we find that maximizing U involves maximizing:

$$\max_a Z = a \cdot (mu - r) - \frac{\pi}{2}a^2\sigma^2$$

Taking the derivative with respect to a, setting it to zero, and solving the first-order condition gives the optimal investment amount:

$$a = \frac{(mu - r)}{\pi\sigma^2}$$

Problem: Show that the second-order condition for a maximum is satisfied.

From this, we conclude: 1. Higher risk aversion leads to lesser investment. 2. Greater expected returns encourage more investment. 3. Increased risk (σ^2) discourages investment.

In the next lecture, we will extend these principles to portfolios with multiple assets using matrix algebra.

2 Matrices

To calculate optimal portfolios for any number of assets, a basic understanding of matrix algebra is essential. Matrix algebra simplifies the resolution of several equations simultaneously, a process that becomes increasingly complex with the addition of variables. Using matrix functions in software like Excel and various statistical packages allows us to solve systems of equations efficiently without manually computing each one.

Matrices not only streamline the computation but also simplify notation, making the formulation of equations for optimal portfolios more manageable.

A matrix is a structured array of numbers arranged in rows and columns, essentially a set of vectors. Here's an example of a vector:

```
import numpy as np
np.random.randint(0,100,3)
```

```
array([78, 20, 57])
```

Combining several vectors side-by-side forms a matrix:

```
np.random.randint(0,100,(2,3))
```

```
array([[73, 70, 57],
       [14, 76, 78]])
```

This format is sometimes denoted as $\mathrm{X}_{N \times K}$ to indicate the number of rows (N) and columns (K).

3 Algebra with Matrices

Matrix algebra operates under similar principles to ordinary algebra—allowing addition, subtraction, multiplication, and division (through inversion)—but it also requires adherence to specific rules.

3.1 Matrix Multiplication

The core operation in matrix algebra is matrix multiplication, which combines elements from the rows of the first matrix with the columns of the second. For example, multiplying a 2×3 matrix by a 3×2 matrix yields:

```
X = np.random.randint(0,100,(2,3))
Y = np.random.randint(0,100,(3,2))
result = np.dot(X, Y)
result
```

```
array([[ 2206,  1482],
       [ 9546, 11190]])
```

Matrix multiplication requires the number of columns in the first matrix to match the number of rows in the second.

3.2 Adding and Subtracting Matrices

Adding or subtracting matrices is straightforward; simply add or subtract corresponding elements:

```
import numpy as np

X = np.random.randint(0,100,(2,2))
Y = np.random.randint(0,100,(2,2))

# Addition of matrices
result_add = X + Y
result_add
```

```
array([[ 59,  70],
       [143,  89]])
```

3.3 Dividing with a Matrix

While direct division isn't defined in matrix operations, we can achieve a similar result by multiplying by the inverse of a matrix. The inverse of a matrix X, denoted X^{-1}, satisfies:

$$XX^{-1} = I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

where I is the identity matrix. Multiplying any matrix by I results in the original matrix, akin to multiplying any number by 1.

To solve the system of equations $XA = B$, we can multiply both sides by X^{-1} to isolate A:

$$A = X^{-1}B$$

In practice, while the concept is straightforward, the actual calculation of a matrix inverse can become complex for larger matrices and is typically handled by computers.

```
# Example of matrix inversion and solving the system
B = np.random.randint(0,100,(3,1))
X = np.random.randint(0,100,(3,3))
# Calculating inverse of X
X_inv = np.linalg.inv(X)

# Solving for A
A = np.dot(X_inv, B)
A
```

```
array([[-0.0505029 ],
       [ 0.343984  ],
       [ 0.21725814]])
```

3.4 Solving Equations with Matrix Algebra

The foundation we've established for matrix algebra now allows us to efficiently solve systems of equations. Consider solving the following pair of simultaneous equations:

$$x_{11}a_1 + x_{12}a_2 = b_1 x_{21}a_1 + x_{22}a_2 = b_2$$

Here, we know the values of x and b but need to find the values of a. These equations can be succinctly expressed using matrix notation:

$$\mathrm{Xa} = \mathrm{b}$$

where a and b are column vectors:

```
a =  np.random.randint(0,100,(2,1))
b =  np.random.randint(0,100,(2,1))

# Define matrix X
X =  np.random.randint(0,100,(2,2))
```

To solve for a, we use the inverse of X, provided it exists:

$$\mathrm{a} = \mathrm{X}^{-1}\mathrm{b}$$

The inverse of X is calculated as:

$$\mathrm{X}^{-1} = \begin{pmatrix} \frac{x_{22}}{x_{11}x_{22}-x_{12}x_{21}} & -\frac{x_{12}}{x_{11}x_{22}-x_{12}x_{21}} \\ -\frac{x_{21}}{x_{11}x_{22}-x_{12}x_{21}} & \frac{x_{11}}{x_{11}x_{22}-x_{12}x_{21}} \end{pmatrix}$$

```
# Manually computing the inverse of X
X = np.random.randint(0,100,(2,2))
det = X[0,0]*X[1,1] - X[0,1]*X[1,0]
X_inv_manual = [
    [X[1,1]/det, -X[1,0]/det],
     [ -X[0,1]/det, X[0,0]/det]
]

X_inv_manual
```

```
[[-0.015080603224128965, 0.020020800832033328],
 [0.01612064482579303, -0.004160166406656267]]
```

This yields the solution:

```
# Solving for a using the inverse of X
a_solution = np.dot(X_inv_manual, b)
a_solution
```

```
array([[0.67056682],
       [0.88663547]])
```

3.5 Practical Example

Let's consider a practical example:

$$2a_1 + 5a_2 = 7$$
$$3a_1 + a_2 = -2$$

Here, X and b are:

```
# Practical example
X_example = np.array([[2, 5], [3, 1]])
b_example = np.array([[7], [-2]])
```

```
# Solving for a using np.linalg.inv
a_example_solution = np.dot(np.linalg.inv(X_example), b_example)
a_example_solution
```

```
array([[-1.30769231],
       [ 1.92307692]])
```

Applying the inverse calculation:

$$\mathrm{a} = \mathrm{X}^{-1}\mathrm{b} = \begin{pmatrix} -\frac{1}{13} & \frac{5}{13} \\ \frac{3}{13} & -\frac{2}{13} \end{pmatrix} \begin{pmatrix} 7 \\ -2 \end{pmatrix} = \begin{pmatrix} -\frac{17}{13} \\ \frac{25}{13} \end{pmatrix}$$

Problem: Verify that these calculations are correct.

This example illustrates how matrix algebra simplifies solving systems of equations, especially when using software tools that can handle large systems effortlessly.

3.6 Transposing

Transposing a matrix involves swapping its rows and columns. For example, a 2×3 matrix:

$$\mathrm{X}_{2\times 3} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$$

transposes to:

$$\mathrm{X}'_{2\times 3} = \begin{pmatrix} x_{11} & x_{21} \\ x_{12} & x_{22} \\ x_{13} & x_{23} \end{pmatrix}$$

where $'$ denotes the transposed matrix. For a column vector a, transposing and then multiplying by itself, $\mathrm{a}'\mathrm{a}$, calculates the sum of squares of its components.

```
# Example of matrix transposition
X_2x3 = np.random.randint(0,100,(2,3))
X_transposed = X_2x3.T
X_transposed
```

```
array([[70, 31],
       [34, 66],
       [23, 15]])
```

Transposition is often used to conform to the requirements of matrix multiplication, where the number of columns in the first matrix must match the number of rows in the second. If this is not the case, one might transpose the first matrix to facilitate multiplication.

4 Calculus and matrices

Deriving matrices follows similar principles to deriving polynomials. For instance:

$$\frac{d\left(a^2\sigma^2\right)}{da} = 2a\sigma^2$$

applies to scalar variables, and for a matrix and a column vector a, we have:

$$\frac{d\left(\mathrm{a}'\ \mathrm{a}\right)}{d\mathrm{a}'} = 2\ \mathrm{a}$$

assuming is symmetric.

```
# Derivation with matrix and vector
a = np.random.randint(0,100,(2,1))
Sigma = np.random.randint(0,100,(2,2))

# Derivative of a' Σ a with respect to a
derivative = 2 * np.dot(Sigma, a)
derivative
```

```
array([[3386],
       [9772]])
```

For a scalar equivalent in a matrix form:

$$\mathrm{a}'\ \mathrm{a} = \sum_{j=1}^{N} a_j \left(\sum_{i=1}^{N} a_i \sigma_{ij} \right)$$

transforms into a vector of derivatives of $\mathrm{a}'\,\mathrm{a}$ concerning each element a_i, simplifying to:

$$\frac{d\,(\mathrm{a}'\,\mathrm{a})}{d\mathrm{a}'} = 2\,\mathrm{a}$$

5 Optimal portfolios with more than one asset

We remember from above that the optimal portfolio with only one asset is

$$a = \frac{(mu - r)}{\pi\sigma^2}$$

From this we concluded that:

1. The more risk-averse the person is, the less they should invest.
2. The larger the expected return of the asset, the more should be invested.
3. The greater the risk associated with the asset, represented by σ^2, the less should be invested.

Now, let us consider the optimal investments if we have more than one asset.

5.1 Optimal Portfolios with Any Number of Assets

Let us now assume that the investor in the previous section has a portfolio of K assets, not just one. Their wealth next period, assuming the entire amount is borrowed, is then expressed in matrix notation as:

$$W_1 = \mathrm{a}'\mathrm{x} - 1r)$$

where a represents the portfolio weights, x represents the returns, and 1 is a column vector of ones, such that $1r$ is a column vector of the risk-free interest rate r. Recall from earlier that the investor aims to maximize the difference between expected return and variance:

$$\max_{\mathrm{a}} Z = \mathbb{E}W_1 - \pi\frac{1}{2}\,\mathrm{var}(W_1)$$

x now is a column vector of many normally distributed variables with different variances and expectations. We denote the expected returns by μ_i for asset i, and the associated vector of these returns by . The expected return on the portfolio then becomes:

$$\mathbb{E}W_1 = \mathrm{a}'\mathbb{E}\mathrm{x} = \mathrm{a}\ -1r)$$

Thus, $\mathbb{E}W_1 = \sum_{i=1}^{K} a_i\mu_i$.

For variance:

```
import numpy as np

# Example matrices for portfolio variance
a = np.random.randint(0,100,(2,1)) # Portfolio weights
mu = np.random.randint(0,100,(2,1))  # Expected returns
r = 0.05  # Risk-free rate

Sigma = np.random.randint(0,100,(2,2))  # Covariance matrix

# Calculating portfolio variance
variance_W1 = np.dot(np.dot(a.T, Sigma), a)
variance_W1
```

```
array([[611308]])
```

Where σ_{ij} is the covariance between i and j, and σ_i^2 is the variance of asset i. This is the covariance matrix, denoted by the capital sigma, .

When a vector is normally distributed we write it as $\mathrm{x} \sim K\ ,\ )$.

We have now found expressions for $\mathbb{E}W_1$ and $\mathrm{var}(W_1)$ in matrix notation. We can thus write:

$$\max_{\mathrm{a}} Z = \mathrm{a}(\ -1r) - \pi\frac{1}{2}\mathrm{a}'\ \mathrm{a}$$

Taking the derivative with respect to a' yields the K first order conditions:

$$\frac{dZ}{d\mathrm{a}} = (\ -1r) - \pi\ \mathrm{a} = 0$$

In optimum, it is necessary that:

$$\mathrm{a} = \frac{1}{\pi} \quad - 1r)$$

By premultiplying with the inverse of , we obtain the optimal portfolio:

$$\mathrm{a} = \frac{1}{\pi}^{-1} \quad - 1r)$$

```
# Example of calculating the optimal portfolio
inv_Sigma = np.linalg.inv(Sigma)
optimal_a =  np.dot(inv_Sigma, (mu - r))
optimal_a
```

```
array([[0.56872064],
       [1.19325985]])
```

Note that this formula looks very similar to the formula for an optimal portfolio with only one asset:

$$a = \frac{(mu - r)}{\pi\sigma^2}$$

In general, we may draw the same conclusions as in the case of one asset:

1. The more risk-averse the person is (large π), the less they should invest.
2. The larger the expected return the asset has, the more should be invested.
3. The more risk is associated with the asset, the less should be invested.

5.2 An Example

Let us take an example with two assets, where the variance and expected return over the risk-free interest rate are given by:

```
# Example covariance matrix and expected returns over risk-free rate
Sigma_example = np.random.randint(0,100,(2,2))
mu_minus_r = np.random.randint(0,100,(2,1))-r

# Optimal portfolio for two assets
```

```
optimal_portfolio =  np.dot(np.linalg.inv(Sigma_example), mu_minus_r
optimal_portfolio
```

```
array([[2.7315261 ],
       [0.32570281]])
```

The optimal portfolio is then:

$$\mathrm{a} = \frac{1}{\pi}^{-1} - 1r)$$

This is the portfolio a CARA-investor with a CARA coefficient π would choose. With this type of utility function, the invested amount is independent of wealth.

An important result within portfolio theory is that a "mean-variance" optimal portfolio has the same composition regardless of the amount invested, and regardless of the utility function, as long as only expected return and variance matter. To express the portfolio more generally, it is useful to normalize it so that the weights a sum up to one. This may be done as follows:

$$\tilde{\mathrm{a}} = \frac{\mathrm{a}}{1'\mathrm{a}} = \begin{pmatrix} \frac{(r_1-r_f)\sigma_2^2-(r_2-r_f)\sigma_{12}}{(r_1-r_f)\sigma_2^2+(r_2-r_f)\sigma_1^2-(r_1+r_2-2r_f)\sigma_{12}} \\ \frac{(r_2-r_f)\sigma_1^2-(r_1-r_f)\sigma_{12}}{(r_1-r_f)\sigma_2^2+(r_2-r_f)\sigma_1^2-(r_1+r_2-2r_f)\sigma_{12}} \end{pmatrix}$$

6 Portfolio Front

An example of this portfolio, with given parameters, is shown in Figure ??.

```
#Examaple here
```

Along the curved line are all the combinations of portfolio weights with the resulting expected return along the y-axis and standard deviation along the x-axis.

The investor can therefore not choose any combination of risk and return, but must choose a combination that is on the portfolio front. However, one can choose to invest everything in the bank at zero risk with a 5% interest rate. If we combine this with the point on the front that gives the highest return for the least amount of variance, we obtain the straight line. This line

provides the maximum trade-off between return and variance for different levels of investment.

Points on the straight line to the left of the tangent point represent an investor who puts some of his wealth in the bank. Points to the right of the tangent point represent an investor who borrows to finance assets.