Generating Financial Portfolio Data

Toren Wallengren

July 16, 2024

1 Introduction

We consider the problem of generating data to simulate financial portfolios. The challenge is to create a data set that is entirely artificial, yet retains the essential qualities of a real financial portfolio. Given the known price data of various assets over time, our goal is to construct a portfolio of holdings that satisfies a list of constraints. Some constraints may include:

- Minimizing risk
- Adhering to a target allocation
- Ensuring the portfolio appears realistic (i.e., no wildly fluctuating positions)

In this paper, we will frame our constraints as the minimum of an objective function and apply gradient descent to find optimal solutions.

2 Definitions

Standardized language is essential for clear communication. We will review the necessary mathematical terminology and introduce vocabulary specific to the problem of creating optimal portfolios.

2.1 Math Terms

We need to understand key concepts from linear algebra, multivariate calculus, and optimization.

2.1.1 Linear Algebra

Linear algebra deals with vectors, matrices, and operations on them. The following concepts are essential:

Vectors A vector is a quantity with both magnitude and direction. Vectors will be denoted by lowercase bold letters with an arrow over the top, e.g., $\vec{\boldsymbol{v}}$. We assume all vectors are column vectors unless otherwise specified. For example:

$$\vec{\boldsymbol{v}} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$$

Here, $\vec{\boldsymbol{v}}$ is an n-dimensional column vector.

Matrices A matrix is a rectangular array of numbers arranged in rows and columns, denoted by uppercase, non-bold letters, e.g., A. Matrices represent linear transformations and systems of linear equations. For example:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Here, A is an $m \times n$ matrix.

Matrix Multiplication Matrix multiplication takes a pair of matrices and produces another matrix. If A is an $m \times n$ matrix and B is an $n \times p$ matrix, their product $C = AB$ is an $m \times p$ matrix where each element c_{ij} is given by:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

For example, if A is a 2×3 matrix and B is a 3×2 matrix:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{pmatrix}$$

then the product $C = AB$ is a 2×2 matrix:

$$C = \begin{pmatrix} 58 & 64 \\ 139 & 154 \end{pmatrix}$$

Matrix Transpose The transpose of a matrix A, denoted A^T, is obtained by swapping its rows and columns. If A is an $m \times n$ matrix, then A^T is an $n \times m$ matrix with elements $a^T_{ij} = a_{ji}$. For example:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \quad A^T = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

Norms The norm of a vector measures its length. For a vector $\vec{\boldsymbol{v}}$, the Euclidean norm is defined as:

$$\|\vec{\boldsymbol{v}}\| = \sqrt{\vec{\boldsymbol{v}}^T \vec{\boldsymbol{v}}}$$

where $\vec{\boldsymbol{v}}^T$ denotes the transpose of $\vec{\boldsymbol{v}}$ and $\vec{\boldsymbol{v}}^T \vec{\boldsymbol{v}}$ is the dot product of $\vec{\boldsymbol{v}}$ with itself. For example, if:

$$\vec{\boldsymbol{v}} = \begin{pmatrix} 3 \\ 4 \end{pmatrix}$$

then the norm is:

$$\|\vec{\boldsymbol{v}}\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = 5$$

Rules of Transpose The transpose operation has several important properties:

- $(A^T)^T = A$
- $(A + B)^T = A^T + B^T$
- $(cA)^T = cA^T$ for any scalar c
- $(AB)^T = B^T A^T$

For example, if:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

then:

$$(A + B)^T = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}^T = \begin{pmatrix} 6 & 10 \\ 8 & 12 \end{pmatrix}$$

and:

$$A^T + B^T = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 7 \\ 6 & 8 \end{pmatrix} = \begin{pmatrix} 6 & 10 \\ 8 & 12 \end{pmatrix}$$

Understanding these linear algebra concepts is crucial for following the optimization techniques and strategies discussed in this paper.

2.1.2 Multivariate Calculus

Multivariate calculus extends single-variable calculus concepts to functions of multiple variables. The following concepts are essential:

Partial Derivatives A partial derivative of a function of several variables is its derivative with respect to one variable, with the others held constant. If $f(x_1, x_2, \ldots, x_n)$ is a function of n variables, the partial derivative with respect to x_i is denoted by $\frac{\partial f}{\partial x_i}$. For example, if:

$$f(x, y) = x^2 y + 3xy^2$$

then the partial derivatives are:

$$\frac{\partial f}{\partial x} = 2xy + 3y^2, \quad \frac{\partial f}{\partial y} = x^2 + 6xy$$

Functions from $\mathbb{R}^n \to \mathbb{R}$ A function $f : \mathbb{R}^n \to \mathbb{R}$ maps an n-dimensional vector to a real number. These functions often represent surfaces or scalar fields. For example, the function $f : \mathbb{R}^2 \to \mathbb{R}$ given by:

$$f(x, y) = x^2 + y^2$$

maps each point (x, y) in $\mathbb{R}^2$ to a real number representing the sum of the squares of the coordinates.

Gradient of a Function from $\mathbb{R}^n \to \mathbb{R}$ The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$, denoted by ∇f, is a vector containing all the partial derivatives of f. It points in the direction of the steepest ascent. For a function $f(x_1, x_2, \ldots, x_n)$, the gradient is given by:

$$\begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

For example, if:

$$f(x, y, z) = x^2 + 2y^2 + 3z^2$$

then the gradient is:

$$\begin{pmatrix} 2x \\ 4y \\ 6z \end{pmatrix}$$

Understanding these multivariate calculus concepts is crucial for following the optimization techniques and strategies discussed in this paper.

2.1.3 Optimization

Optimization is about finding the best solution to a problem within a given set of constraints. Here, we focus on finding the local minimum of functions from $\mathbb{R}^n \to \mathbb{R}$ using gradient descent.

Gradient Descent Gradient descent is an iterative optimization algorithm for finding the local minimum of a differentiable function. Starting with an initial guess, the algorithm updates it iteratively by moving in the direction opposite to the gradient, as this is the direction of steepest descent.

Given a function $f : \mathbb{R}^n \to \mathbb{R}$, the gradient descent update rule is:

$$\vec{\boldsymbol{x}}_{k+1} = \vec{\boldsymbol{x}}_k - \alpha \nabla f(\vec{\boldsymbol{x}}_k)$$

where:

- $\vec{\boldsymbol{x}}_k$ is the current point in $\mathbb{R}^n$,
- $\vec{\boldsymbol{x}}_{k+1}$ is the updated point,
- α is the learning rate, a positive scalar determining the step size,
- $\nabla f(\vec{\boldsymbol{x}}_k)$ is the gradient of f at $\vec{\boldsymbol{x}}_k$.

Example Consider the function $f(x, y) = x^2 + y^2$. The gradient is:

$$\nabla f = \begin{pmatrix} 2x \\ 2y \end{pmatrix}$$

Starting with an initial point $(x_0, y_0) = (1, 1)$ and a learning rate $\alpha = 0.1$, we perform the following iterations:

- Iteration 1:
$$\vec{\boldsymbol{x}}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - 0.1 \begin{pmatrix} 2 \cdot 1 \\ 2 \cdot 1 \end{pmatrix} = \begin{pmatrix} 0.8 \\ 0.8 \end{pmatrix}$$

- Iteration 2:
$$\vec{\boldsymbol{x}}_2 = \begin{pmatrix} 0.8 \\ 0.8 \end{pmatrix} - 0.1 \begin{pmatrix} 2 \cdot 0.8 \\ 2 \cdot 0.8 \end{pmatrix} = \begin{pmatrix} 0.64 \\ 0.64 \end{pmatrix}$$

This process is repeated until convergence, i.e., when the updates become negligibly small. In this case, gradient descent converges to the point:

$$\vec{\boldsymbol{x}}_{min} = \begin{pmatrix} 0.0 \\ 0.0 \end{pmatrix}$$

Convergence Criteria In practice, the gradient descent algorithm stops when one of the following criteria is met:

- The magnitude of the gradient $\|\nabla f(\vec{\boldsymbol{x}}_k)\|$ falls below a predefined threshold.
- The change in function value $|f(\vec{\boldsymbol{x}}_{k+1}) - f(\vec{\boldsymbol{x}}_k)|$ is less than a predefined threshold.
- A maximum number of iterations is reached.

Gradient descent is a powerful and widely used method for optimization, particularly in high-dimensional spaces and complex functions.

Understanding these optimization concepts is crucial for following the techniques and strategies discussed in this paper.

2.1.4 Putting It All Together

Consider a vector $\vec{\boldsymbol{v}}$ of length n and a matrix A of size $m \times n$. The product $A\vec{\boldsymbol{v}}$ is another vector of length m. We are interested in minimizing the norm of this resulting vector, $\|A\vec{\boldsymbol{v}}\|$.

Norm of $A\vec{\boldsymbol{v}}$ The norm of $A\vec{\boldsymbol{v}}$ is defined as:

$$\|A\vec{\boldsymbol{v}}\| = \sqrt{(A\vec{\boldsymbol{v}})^T(A\vec{\boldsymbol{v}})}$$

Using the properties of matrix transposition, this can be rewritten as:

$$\|A\vec{\boldsymbol{v}}\| = \sqrt{\vec{\boldsymbol{v}}^T A^T A\vec{\boldsymbol{v}}}$$

Gradient of the Norm To minimize the norm $\|A\vec{\boldsymbol{v}}\|$, we need to compute its gradient with respect to $\vec{\boldsymbol{v}}$. Let $f(\vec{\boldsymbol{v}}) = \vec{\boldsymbol{v}}^T A^T A\vec{\boldsymbol{v}}$. Then the norm can be expressed as:

$$\|A\vec{\boldsymbol{v}}\| = \sqrt{f(\vec{\boldsymbol{v}})}$$

The gradient of $f(\vec{\boldsymbol{v}})$ is given by:

$$\nabla f(\vec{\boldsymbol{v}}) = 2A^T A\vec{\boldsymbol{v}}$$

Using the chain rule, the gradient of $\|A\vec{\boldsymbol{v}}\|$ is:

$$\nabla\|A\vec{\boldsymbol{v}}\| = \frac{1}{2}\left(\vec{\boldsymbol{v}}^T A^T A\vec{\boldsymbol{v}}\right)^{-\frac{1}{2}} \cdot 2A^T A\vec{\boldsymbol{v}} = \frac{A^T A\vec{\boldsymbol{v}}}{\|A\vec{\boldsymbol{v}}\|}$$

Applying Gradient Descent To minimize the norm $\|A\vec{\boldsymbol{v}}\|$, we apply the gradient descent algorithm. Starting with an initial guess $\vec{\boldsymbol{v}}_0$, the update rule for gradient descent is:

$$\vec{\boldsymbol{v}}_{k+1} = \vec{\boldsymbol{v}}_k - \alpha\nabla\|A\vec{\boldsymbol{v}}_k\|$$

where α is the learning rate. Substituting the gradient we computed:

$$\vec{\boldsymbol{v}}_{k+1} = \vec{\boldsymbol{v}}_k - \alpha\frac{A^T A\vec{\boldsymbol{v}}_k}{\|A\vec{\boldsymbol{v}}_k\|}$$

Example Consider a simple example with:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad \vec{\boldsymbol{v}} = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

Then:

$$A\vec{\boldsymbol{v}} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} v_1 + 2v_2 \\ 3v_1 + 4v_2 \end{pmatrix}$$

The norm is:

$$\|A\vec{\boldsymbol{v}}\| = \sqrt{(v_1 + 2v_2)^2 + (3v_1 + 4v_2)^2}$$

The gradient is:

$$\nabla\|A\vec{\boldsymbol{v}}\| = \frac{A^T A\vec{\boldsymbol{v}}}{\|A\vec{\boldsymbol{v}}\|} = \frac{\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}}{\|A\vec{\boldsymbol{v}}\|} = \frac{\begin{pmatrix} 10 & 14 \\ 14 & 20 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}}{\|A\vec{\boldsymbol{v}}\|}$$

Applying the gradient descent update rule:

$$\vec{\boldsymbol{v}}_{k+1} = \vec{\boldsymbol{v}}_k - \alpha\frac{\begin{pmatrix} 10v_1 + 14v_2 \\ 14v_1 + 20v_2 \end{pmatrix}}{\|A\vec{\boldsymbol{v}}_k\|}$$

This iterative process continues until convergence, i.e., until the updates to $\vec{\boldsymbol{v}}$ become negligibly small. By applying gradient descent, we can effectively minimize the norm of the vector $A\vec{\boldsymbol{v}}$, leading to an optimal solution for the given problem.

Understanding these steps allows us to use gradient descent to minimize the norm of $A\vec{\boldsymbol{v}}$, which will be utilized for constructing optimal financial portfolios.

2.2 Domain-Specific Terms

In the context of financial portfolios, we need to define several domain-specific terms to describe the data and operations involved in constructing and optimizing portfolios.

Single Asset Consider a single asset, such as a stock. Suppose we have the end-of-day price for this stock every single day for T days. We can describe the units owned of that asset at the end of each day using a T-dimensional "unit vector" $\vec{\boldsymbol{u}}$. For example:

$$\vec{\boldsymbol{u}} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_T \end{pmatrix}$$

Price Matrix The prices of the asset can be written into a diagonal "price matrix" P. The diagonal elements of P represent the price of the asset at each end-of-day. For example:

$$P = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & p_T \end{pmatrix}$$

Here, p_i is the price of the asset at the end of day i.

Value Vector The value of the asset at the end of each day, represented by the vector $\vec{\boldsymbol{v}}$, is obtained by multiplying the price matrix P with the unit vector $\vec{\boldsymbol{u}}$:

$$\vec{\boldsymbol{v}} = P\vec{\boldsymbol{u}}$$

Portfolio of N Assets Now, suppose we want to create a portfolio of N assets over T days. Each asset has a corresponding T-dimensional unit vector $\vec{\boldsymbol{u}}_i$ for $i = 1, 2, \ldots, N$. The "portfolio unit vector" $\vec{\boldsymbol{u}}_p$ is a column vector where the first value is 1, and the subsequent values are the concatenation of the unit vectors of the individual assets. Formally, if we have unit vectors $\vec{\boldsymbol{u}}_1, \vec{\boldsymbol{u}}_2, \ldots, \vec{\boldsymbol{u}}_N$, then the portfolio unit vector is:

$$\vec{\boldsymbol{u}}_p = \begin{pmatrix} 1 \\ \vec{\boldsymbol{u}}_1 \\ \vec{\boldsymbol{u}}_2 \\ \vdots \\ \vec{\boldsymbol{u}}_N \end{pmatrix}$$

This vector has dimension $N \times T + 1$.

Portfolio Price Matrix The "portfolio price matrix" P_p is an $(N \times T + 1) \times (N \times T + 1)$ matrix. The first row is $[1, 0, 0, \ldots, 0]$, ensuring that the first element of the portfolio value vector remains 1. The subsequent diagonal blocks correspond to the price matrices of the individual assets. Formally:

$$P_p = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & P_1 & 0 & \cdots & 0 \\ 0 & 0 & P_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & P_N \end{pmatrix}$$

where P_i is the price matrix for the i-th asset.

Portfolio Value Vector The "portfolio value vector" $\vec{\boldsymbol{v}}_p$ is obtained by multiplying the portfolio price matrix P_p with the portfolio unit vector $\vec{\boldsymbol{u}}_p$. It has dimension $N \times T + 1$ and its first element is 1, preserving the structure of the portfolio unit vector. Formally:

$$\vec{\boldsymbol{v}}_p = P_p \vec{\boldsymbol{u}}_p$$

This ensures that the value of the portfolio is computed correctly based on the individual asset prices and units owned.

Example For clarity, consider a simple example with $N = 2$ assets and $T = 3$ days. Let the unit vectors be:

$$\vec{\boldsymbol{u}}_1 = \begin{pmatrix} u_{11} \\ u_{12} \\ u_{13} \end{pmatrix}, \quad \vec{\boldsymbol{u}}_2 = \begin{pmatrix} u_{21} \\ u_{22} \\ u_{23} \end{pmatrix}$$

The portfolio unit vector $\vec{\boldsymbol{u}}_p$ is:

$$\vec{\boldsymbol{p}} = \begin{pmatrix} 1 \\ u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \end{pmatrix}$$

The portfolio price matrix P_p is:

$$P_p = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & p_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & p_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & p_{13} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{21} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & p_{22} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & p_{23} \end{pmatrix}$$

The portfolio value vector $\vec{\boldsymbol{v}}_p$ is:

$$\vec{\boldsymbol{v}}_p = P_p \vec{\boldsymbol{u}}_p = \begin{pmatrix} 1 \\ p_{11}u_{11} \\ p_{12}u_{12} \\ p_{13}u_{13} \\ p_{21}u_{21} \\ p_{22}u_{22} \\ p_{23}u_{23} \end{pmatrix}$$

2.2.1 Significance of the Initial 1 in Vectors and Matrices

The inclusion of the initial 1 in the unit and value vectors, and the [1, 0, 0, ... 0] row in the price matrix, serves two important purposes:

Encoding Constants in Matrix Multiplication In financial modeling, it is often necessary to include constants in operations involving unit vectors. By having a 1 at the beginning of the unit vectors, we can encode constants into matrix multiplication. This allows us to define transformations that incorporate constants directly. For example, suppose we want to add a constant c to each element of the portfolio unit vector $\vec{\boldsymbol{u}}_p$. By defining an appropriate matrix that includes the constant c, we can perform this operation using matrix multiplication.

Consider a portfolio unit vector $\vec{\boldsymbol{u}}_p$ with an initial 1:

$$\vec{\boldsymbol{u}}_p = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \vdots \\ u_T \end{pmatrix}$$

We can define a transformation matrix C that adds a constant c to each element of $\vec{\boldsymbol{u}}_p$:

$$C = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ c & 1 & 0 & \cdots & 0 \\ c & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c & 0 & 0 & \cdots & 1 \end{pmatrix}$$

Multiplying C by $\vec{\boldsymbol{u}}_p$ gives:

$$C\vec{\boldsymbol{u}}_p = \begin{pmatrix} 1 \\ c + u_1 \\ c + u_2 \\ \vdots \\ c + u_T \end{pmatrix}$$

Ensuring Non-Zero Norms in Gradient Descent When using gradient descent to minimize the norm of a vector, we often divide by the norm of the vector. Including a constant 1 in every vector ensures that we never divide by something smaller than 1, thus preventing division by zero or very small numbers, which can lead to numerical instability.

Recalling the expression for the gradient of the norm:

$$\nabla \|A\vec{\boldsymbol{v}}\| = \frac{A^T A\vec{\boldsymbol{v}}}{\|A\vec{\boldsymbol{v}}\|}$$

By having a constant 1 in the vector, we guarantee that $\|A\vec{\boldsymbol{v}}\| \geq 1$. This ensures the denominator is always at least 1, avoiding division by zero or near-zero values. This stabilization is crucial for the robustness and convergence of the gradient descent algorithm.

3 Constructing Portfolios

In this section, we will explore the process of constructing financial portfolios that meet predefined constraints using known price data. We will begin by outlining the abstract idea and then illustrate it with specific examples to demonstrate its practicality and effectiveness.

3.1 Problem Statement

Given a portfolio price matrix P_p and a set of constraints, our objective is to generate a portfolio unit vector $\vec{\boldsymbol{u}}_p$ that satisfies these constraints and is optimal in some sense. This involves translating the constraints into mathematical expressions and using optimization techniques to find the desired portfolio.

3.1.1 Key Insight

Constraints can often be formulated as operators acting on portfolio unit vectors. If these constraint operators are linear, they can be represented as matrices. This allows us to leverage the linear algebra techniques discussed earlier to find optimal unit vectors.

3.2 Portfolio Constraints

Constraints are essential in defining the characteristics of the portfolio. They ensure the portfolio adheres to certain criteria, such as risk minimization, target allocation, and realism in asset positions. Here, we will describe a few constraint operators and discuss how these will help us generate ideal portfolios.

3.2.1 Target Allocation Operator

In financial portfolios, it is common to specify a target allocation. If we have a portfolio of assets, a target allocation will categorize those assets into various classes, and then specify how much of each class the portfolio should be composed of.

Example Suppose we have a portfolio consisting of 2 stock positions and 2 bond positions. We could specify a target allocation by saying "our portfolio should be half equity and half fixed income by value". This means that, if our portfolio is worth \$1000, then \$500 should be in equity (stocks) and \$500 should be fixed income (bonds).

Note: a target allocation does not specify values. It describes the concentration of positions as a percentage of the total value of the portfolio.

Constructing an Operator Let us continue with the example above. Suppose we have a portfolio consisting of 2 stock positions and 2 bond positions over T days. Denote the value vectors of our stock positions as

$$\vec{\boldsymbol{s}}_1 = \begin{pmatrix} s_{11} \\ s_{12} \\ \vdots \\ s_{1T} \end{pmatrix}, \vec{\boldsymbol{s}}_2 = \begin{pmatrix} s_{21} \\ s_{22} \\ \vdots \\ s_{2T} \end{pmatrix}$$

and the value vectors for our bond positions

$$\vec{\boldsymbol{b}}_1 = \begin{pmatrix} b_{11} \\ b_{12} \\ \vdots \\ b_{1T} \end{pmatrix}, \vec{\boldsymbol{b}}_2 = \begin{pmatrix} b_{21} \\ b_{22} \\ \vdots \\ b_{2T} \end{pmatrix}$$

Observation Consider the expression

$$((s_{11}+s_{21})-(b_{11}+b_{21}))^2+((s_{12}+s_{22})-(b_{12}+b_{22}))^2+\cdots+((s_{1T}+s_{2T})-(b_{1T}+b_{2T}))^2$$

This can be rewritten as

$$(v_{e1} - v_{f1})^2 + (v_{e2} - v_{f2})^2 + \cdots + (v_{eT} - v_{fT})^2$$

where v_{en} is the total value of equity positions on day n, and v_{fn} is the total value of fixed income positions on day n. If we have a portfolio that meets a 50/50 target allocation between these two asset classes, this expression is minimized (it is zero in this case, and positive in all other cases). So the question is:

Does there exist some linear operator O_{TA} such that

$$||O_{TA}\vec{\boldsymbol{u}}|| = \sqrt{1 + (v_{e1} - v_{f1})^2 + (v_{e2} - v_{f2})^2 + \cdots + (v_{eT} - v_{fT})^2}$$

In Matrix Form We will describe the matrix of the Target Allocation Operator for this example and leave it to the reader to prove it out. For this example, we have our portfolio unit vector:

$$\vec{u} = \begin{pmatrix} 1 \\ \vec{s}_1 \\ \vec{s}_2 \\ \vec{b}_1 \\ \vec{b}_2 \end{pmatrix}$$

and our target allocation operator

$$O_{TA} = \begin{pmatrix} 1 & \vec{0} & \vec{0} & \vec{0} & \vec{0} \\ \vec{0} & P_{s1} & P_{s2} & -P_{b1} & -P_{b2} \end{pmatrix}$$

where the P_n's are the price matrices for each asset.

General Case of 2 Asset Classes For the case of an arbitrary portfolio with 2 asset classes, it is hopefully clear to see how to start generalizing the Target Allocation Operator from this example. We will state the generalization as follows:

Suppose we have a portfolio consisting of two asset classes C_1 and C_2, and we want a portfolio such that x percent of the total portfolio value is in C_1 and $1 - x$ percent is in C_2. Then our target allocation operator is

$$O_{TA} = \begin{pmatrix} 1 & \vec{0} & \vec{0} \\ \vec{0} & \frac{x}{1-x}C_1 & -C_2 \end{pmatrix}$$

where

$$C_1 = \begin{pmatrix} P_{11} & P_{12} & ... & P_{1N} \end{pmatrix}$$

and

$$C_2 = \begin{pmatrix} P_{21} & P_{22} & ... & P_{1M} \end{pmatrix}$$

are the matrices consisting of all of the underlying price matrices in each asset class.

Further Generalization At this time, it is not entirely clear to me how to generalize this operator to a higher number of asset classes, or to allow for a target allocation that changes over time. Both cases should be possible, but more investigation is required here.

Discussion To review, the name of the game is to write a constraint as a linear operator and then use gradient descent to find an optimal unit vector. In this case, we were able to translate a target allocation constraint into a linear operator in matrix form, which means the machinery we developed earlier applies and this becomes a 'plug n chug' exercise. However, a portfolio that only takes into account the target allocation constraint will probably not be sufficient.

It will work, but the real power of this gradient descent method is that we can perform optimization on multiple constraints at once.

3.2.2 Target Value Operator

When constructing a portfolio, we may want the overall portfolio value to take on specific values or follow specific trends. For example, we may want to be able to construct a portfolio whose overall value stays as close to \$100,000 as possible. This operator allows us to construct such portfolios and more.

Matrix Form Suppose we have a portfolio consisting of N assets over T days. Suppose we also have a target value vector

$$\vec{\boldsymbol{v}}_t = \begin{pmatrix} v_{t0} \\ v_{t1} \\ \vdots \\ v_{tT} \end{pmatrix}$$

Then our target value operator O_{TV} is written as

$$O_{TV} = \begin{pmatrix} 1 & \vec{\mathbf{0}} & \vec{\mathbf{0}} & \cdots & \vec{\mathbf{0}} \\ -\vec{\boldsymbol{v}}_t & P_1 & P_2 & \cdots & P_N \end{pmatrix}$$

where the P_n matrices are the price matrices for all of the underlying assets.

Norm To make sense of this operator, consider $||O_{TV}\vec{\boldsymbol{u}}_p||$. Writing this out, we get an expression of the form

$$||O_{TV}\vec{\boldsymbol{u}}_p|| = \sqrt{1 + (v_{p1} - v_{t1})^2 + (v_{p2} - v_{t2})^2 + \cdots + (v_{pN} - v_{tN})^2}$$

where the v_{pn}'s are the total value of the portfolio on day n. This expression evaluates to 1 when the total portfolio value is equal to the target value every single day, and is a number greater than 1 in all other cases. So the optimal portfolio is the minimum of this function, and we can utilize the gradient descent method to find the appropriate unit vectors.

Discussion Hopefully it is becoming more clear the flexibility we have with the gradient descent method. At this point, we have the machinery necessary to generate portfolios that meet a value constraint and an allocation constraint. These two constraints alone should already be sufficient to generate fairly realistic portfolio data.

Generalization It is possible to generalize the Target Value Operator to something that also considers a kind of allocation constraint. Suppose we have a portfolio with M asset classes, and we specify M target value vectors (one for each asset class). Then we can write a generalized target value operator as

$$O_{TV} = \begin{pmatrix} 1 & \vec{\mathbf{0}} & \vec{\mathbf{0}} & \cdots & \vec{\mathbf{0}} \\ -\vec{\boldsymbol{v}}_{t1} & C_1 & \vec{\mathbf{0}} & \cdots & \vec{\mathbf{0}} \\ -\vec{\boldsymbol{v}}_{t2} & \vec{\mathbf{0}} & C_2 & \cdots & \vec{\mathbf{0}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\vec{\boldsymbol{v}}_{tM} & \vec{\mathbf{0}} & \vec{\mathbf{0}} & \cdots & C_M \end{pmatrix}$$

where

$$C_m = \begin{pmatrix} P_{m1} & P_{m2} & \ldots & P_{mN} \end{pmatrix}$$

is the collection of price matrices corresponding to asset class m.