Avoiding the Downside: A Practical Review of the Critical Line Algorithm for Mean-Semivariance Portfolio Optimization

Harry Markowitz[1] David Starer[2] Harvey Fram[3] Sander Gerber[4]

June 21, 2019

1. Harry Markowitz is President at Harry Markowitz Company, 1010 Turquoise Street. San Diego, CA 92109.

2. David Starer is Director of Research at Constantia Capital, 17 Hendrickson Rd., Suite 100, Lawrenceville, NJ 08648.

3. Harvey Fram is Founder and Portfolio Manager at Constantia Capital, 17 Hendrickson Rd., Suite 100, Lawrenceville, NJ 08648.

4. Sander Gerber is Managing Partner, Chief Executive Officer, and Chief Investment Officer at Hudson Bay Capital Management, 777 Third Avenue, 30th Floor, New York, NY 10017.

Abstract

Optimizing a portfolio to reduce exposure to downside risk can be difficult, and usually involves third or higher order statistical moments of the portfolio's return distribution. Mean-semivariance optimization simplifies this problem by using only the first two moments of the distribution and by penalizing returns below a predetermined reference. Although this penalty introduces a nonlinearity, mean-semivariance optimization can be performed easily and efficiently using the critical line algorithm provided that the covariance matrix is estimated from an historical record of asset returns. In practice, this proviso is not restrictive. This chapter reviews the theory of the critical line algorithm and presents sample computer code for applying the algorithm to mean-variance and mean-semivariance portfolio optimization. It also reviews a method for finding the efficient mean-semivariance portfolio for any given feasible desired expected portfolio return.

1 Introduction

Mean-Variance portfolio optimization continues to dominate asset management. In the majority of cases, it is the best method because, under widely applicable conditions, it maximizes the investor's utility [Levy and Markowitz, 1979], it is very easy to implement, and it is intuitively appealing.

Variance, however, is a symmetric measure; meaning that mean-variance optimization disfavors large positive returns as much as it disfavors large negative returns. Generally though, investors favor large positive returns, or positively skewed portfolio return distributions. Therefore, some would argue that mean-variance optimization should be augmented with methods that penalize only negative returns; i.e., with downside portfolio optimization. For this purpose, semivariance is a particularly useful measure of downside risk because it considers returns as risky only if they are below some reference return.

Despite its intuitive appeal, mean-semivariance optimization remains shrouded in mystery. This is primarily because of the difficulties raised by the apparent nonlinearity involved in the computation of semivariance. For example, Estrada [2008] states, "semivariance optimal portfolios cannot be determined without resorting to obscure numerical algorithms." In a similar vein, Cumova and Nawrocki [2011] state that mean-semivariance optimization "requires a laborious iterative process because the cosemivariance matrix is endogenous and a closed form solution does not exist," and Cumova and Nawrocki [2014] says in part that mean-semivariance optimization "has traditionally been challenged by academic researchers because of the computational complexity of the asymmetric [semicovariance] matrix."

In reality, the mean-semivariance optimization problem was solved more than sixty years ago in Markowitz [1959], which showed that if an estimate of the covariance matrix is computed using time series of security returns, the great computational benefit of the Critical Line Algorithm (CLA) could be exploited to construct mean-semivariance efficient frontiers. In addition to being extremely fast, the CLA is simple to understand.

With the appropriate definition of new non-negative variables, Markowitz, Todd, Xu, and Yamane [1992, 1993] showed how the mean-semivariance problem could be formulated such that a standard quadratic (i.e., mean-variance optimizer) could be used for the mean-semivariance problem as well.

In this chapter, we provide a review of the CLA for mean-variance and mean-semivariance optimization, as well as the method of Markowitz, Todd, Xu, and Yamane [1992, 1993] (the MTXY method) for mean-semivariance optimization. The primary reference for practical computation of the CLA itself is Chapter 13 of Markowitz and Todd [2000], which provides a Visual Basic implementation of the algorithm. Similar descriptions, such as those in Niedermayer and Niedermayer [2009] and Bailey and López de Prado [2013] have provided didactic and open-source versions in FORTRAN and Python. The contributions of this chapter are that we provide simple example code for the CLA applied to the mean-*semivariance* problem. Previously, only a brief description of the method has been available in Markowitz [1959] with speed improvements and slightly more complicated extensions in Markowitz, Todd, Xu, and Yamane [1992, 1993].

Our objective is to provide a step-by-step tutorial of the basic CLA, with the theory of each step clearly explained and tied to a specific block of computer code. To emphasize the flexibility of the CLA, we first review the CLA for mean-variance optimization with reference to a simple Python realization. We then build on the basic CLA and provide R code for mean-semivariance optimization. Finally, we show how to solve the mean-semivariance optimization problem with modern convex optimization using the CVXR library [Boyd and Vandenberghe, 2004]. We emphasize, however, that the computer code presented here is not intended for production purposes. It is merely illustrative.

One of the many benefits of the CLA is that it is amenable to very rapid computation

using an efficient update of a certain matrix inverse when the components of that matrix change. The details of this update, based on the Sherman-Morrison-Woodbury formula (see, for example, Hager [1989] and references therein) are provided in Chapter 13 of Markowitz [1959]. While very important for the purpose of obtaining fast code, the details of this matrix inverse update tend to hide the simplicity of the CLA. Therefore, in this chapter, we simply re-invert that matrix whenever needed. In practice, with modern computer hardware and software, and for typical institutional portfolios, the degradation in performance is not prohibitive. We also omit details of the initialization of the algorithm for general linear equality and inequality constraints. This merely requires standard linear programming, and again the details are available in Markowitz and Todd [2000].

2 The Critical Line Algorithm

The objective in portfolio optimization is to find the fraction x_i of investor wealth to be assigned to each asset i of n assets so as to maximize investor's utility subject to all constraints that the investor faces. In this section, we use the mean-variance approximation to utility [Levy and Markowitz, 1979]. Let μ_{P} be the expected value of the portfolio's return, let σ_{P}^2 be the variance of the portfolio's return, and let $\lambda_{\text{E}} \geq 0$ be a parameter under the investor's control that adjusts the trade-off between risk (variance) and return.[1] The mean-variance approximation to utility is

$$\mathcal{U} = \lambda_{\text{E}}\mu_{\text{P}} - \tfrac{1}{2}\sigma_{\text{P}}^2. \tag{1}$$

The efficient frontier is the locus of all points in $(\sigma_{\text{P}}, \mu_{\text{P}})$-space corresponding to feasible portfolios that minimize σ_{P} for given μ_{P}, or maximize μ_{P} for given σ_{P}. Each point on the efficient frontier is the optimal pair $(\sigma_{\text{P}}, \mu_{\text{P}})$ for a given choice of λ_{E} The Critical Line Algorithm (CLA) [Markowitz, 1987] is a computationally efficient method for finding the entire efficient frontier.

To understand the CLA, let x_i be the fraction of the investor's wealth (or weight) invested in security i, let μ_i be the expected return of security i, and let σ_{ij} or $\sigma_{i,j}$ be the covariance between the returns of securities i and j. Then the expected return and the variance of the portfolio's return are

$$\mu_{\text{P}} = \sum_{i=1}^{n} \mu_i x_i, \quad \sigma_{\text{P}}^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i \sigma_{ij} x_j.$$

Defining the mean vector $\boldsymbol{\mu}$, the portfolio weight vector $\boldsymbol{x}$, and the covariance matrix $\boldsymbol{C}$ as

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \boldsymbol{C} = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \dots & \sigma_{1,n} \\ \sigma_{2,1} & \sigma_{2,2} & \dots & \sigma_{2,n} \\ \vdots & & & \vdots \\ \sigma_{n,1} & \sigma_{n,2} & \dots & \sigma_{n,n} \end{bmatrix},$$

we can also write the portfolio mean and variance as

$$\mu_{\text{P}} = \boldsymbol{\mu}^\top \boldsymbol{x}, \quad \sigma_{\text{P}}^2 = \boldsymbol{x}^\top \boldsymbol{C} \boldsymbol{x}.$$

[1] We can also interpret λ_{E} as a measure of risk tolerance.

The mean-variance approximation to utility in equation (1) is then

$$\mathcal{U} = \lambda_{\text{E}} \boldsymbol{\mu}^\top \boldsymbol{x} - \tfrac{1}{2} \boldsymbol{x}^\top \boldsymbol{C} \boldsymbol{x}. \tag{2}$$

Our objective, therefore, is to find the value of the weight vector $\boldsymbol{x}$ that maximizes (or equivalently that minimizes the negative of) the utility expressed in Equation (2).

The weights x_i are subject to constraints. By definition, since x_i is the fraction of the investor's wealth invested in security i, these weights must sum to 100%. This is the budget constraint, and it is written as

$$\sum_{i=1}^{n} x_i = 1.$$

Using more compact vector notation, with $\boldsymbol{\iota}$ being defined as a vector of ones whose dimension is determined from context, the budget constraint can be written as $\boldsymbol{\iota}^\top \boldsymbol{x} = 1$. The budget constraint is an example of one equation that can be included in a more general set of m linear equality constraints

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{3}$$

for given $m \times n$ constraint matrix $\boldsymbol{A}$ and $m \times 1$ constraint vector $\boldsymbol{b}$.

For the moment, consider the Lagrangian for minimizing the negative of the utility in Equation (2) subject only to the linear equality constraints. That is,

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda} | \lambda_{\text{E}}) = \underbrace{\tfrac{1}{2} \boldsymbol{x}^\top \boldsymbol{C} \boldsymbol{x} - \lambda_{\text{E}} \boldsymbol{\mu}^\top \boldsymbol{x}}_{\text{Objective}} - \boldsymbol{\lambda}^\top \underbrace{(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})}_{\text{Constraint}}. \tag{4}$$

Here, $\boldsymbol{\lambda}$ is the $m \times 1$ Lagrange multiplier vector on the linear equality constraints.

In this case, the vector derivatives of the Lagrangian with respect to both $\boldsymbol{x}$ and $\boldsymbol{\lambda}$ must be equal to zero vectors. Differentiating the Lagrangian, we obtain

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \boldsymbol{C}\boldsymbol{x} - \boldsymbol{\mu}\lambda_{\text{E}} + \boldsymbol{A}^\top \boldsymbol{\lambda}, \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\lambda}} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}. \tag{6}$$

Setting these equations to zero to find the turning points of the Lagrangian, we get

$$\begin{bmatrix} \boldsymbol{C} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu} \\ \mathbf{0} \end{bmatrix} \lambda_{\text{E}} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{b} \end{bmatrix}, \tag{7}$$

where $\mathbf{O}$ is an appropriately dimensioned matrix of zeros, and $\mathbf{0}$ is an appropriately dimensioned vector of zeros.

Besides the linear equality constraints, the weights may also be subject to sets of linear inequality constraints in the form $\boldsymbol{F}\boldsymbol{x} \leq \boldsymbol{g}$. Inequality constraints can be converted into equality constraints with the addition of slack or surplus variables together with non-negativity constraints. Here, we will assume that such conversions have already been made, and consider only bounds on the minimum and maximum values of x_i. Specifically, define the $n \times 1$ minimum (down) vector $\boldsymbol{d}$ with elements d_i such that $x_i \geq d_i$. For example, the

lower bound on a simple long-only portfolio is $d_i = 0$ for all i. Also define the $n \times 1$ maximum (up) vector $\boldsymbol{u}$ with elements u_i such that $x_i \leq u_i$. In vector form, $\boldsymbol{d} \leq \boldsymbol{x} \leq \boldsymbol{u}$.

We now consider the modifications of Equation (7) that are necessary to take account of the bounds. For a particular value of λ_{E} (i.e., for a point on a particular segment of the efficient frontier, as we shall discuss later), element i in the first block in Equation (7), or equivalently element i obtained when the derivative in Equation (5) is set to zero, is

$$\sum_{j=1}^{n} \sigma_{i,j} x_j + \sum_{j=1}^{m} a_{j,i} \lambda_j = \mu_i \lambda_{\text{E}} \text{ for } i \in \texttt{IN} \subseteq \{1, \ldots, n\}, \tag{8}$$

where $a_{j,i}$ is the element in row j and column i of $\boldsymbol{A}$. Importantly, this equation only holds for security i if the derivative $\partial \mathcal{L} / \partial x_i$ is zero for a value of x_i that lies between the upper bound u_i and lower bound d_i. We refer to the set of i for which these conditions hold as the `IN` set. For securities whose derivatives are zero for values outside of these bounds (i.e., for $i \notin \texttt{IN}$), the corresponding elements of Equation (8) do not hold, and must be modified.

First consider all securities i for which $\partial \mathcal{L} / \partial x_i$ would be zero at a value of x_i that is above the upper bound u_i. Such securities must be constrained to their upper bounds. We define `UP` to be the set of i for securities constrained like this. For these securities, since we can no longer set the derivatives equal to zero, we can ignore the previous contents of each equation $i \in \texttt{UP}$ in the top block of equation set (7). Instead, we use each equation i to impose the constraint $x_i = u_i$. Similarly, if $\partial \mathcal{L} / \partial x_i$ would be zero for a value of x_i that is below the lower bound d_i, we must constrain x_i to its upper bound. We define `DN` to be the set of i for securities constrained like this.

To impose these bounds for all $i \in \texttt{UP}$ or $i \in \texttt{DN}$ in the top block row of Equation (7):

S1. Replace row i of $\boldsymbol{C}$ with zeros, except in column i. Place a 1 at the intersection of row i and column i. Let $\bar{\boldsymbol{C}}$ be the resulting matrix after this replacement.

S2. Replace column i of $\boldsymbol{A}$ (i.e., row i of $\boldsymbol{A}^\top$) with zeros. Let $\bar{\boldsymbol{A}}$ be the resulting matrix after this replacement.

S3. Replace element i of $\boldsymbol{\mu}$ with zero. Let $\bar{\boldsymbol{\mu}}$ be the resulting vector after this replacement.

S4. Add a vector $\boldsymbol{k}$ to the right-hand side with entries $k_i = u_i$ if $i \in \texttt{UP}$, or $k_i = d_i$ if $i \in \texttt{DN}$, or $k_i = 0$ if $i \in \texttt{IN}$.

We therefore now have

$$\bar{\boldsymbol{C}} \boldsymbol{x} + \bar{\boldsymbol{A}}^\top \boldsymbol{\lambda} = \bar{\boldsymbol{\mu}} \lambda_E + \boldsymbol{k}. \tag{9}$$

This equation takes care of the first block equation in equation set (7) by ensuring that the derivatives $\partial \mathcal{L} / \partial x_i$ are equal to zero for $i \in \texttt{IN}$, and that the weights x_i are equal to their upper bounds for $i \in \texttt{UP}$ or lower bounds for $i \in \texttt{DN}$.

To maintain symmetry in the matrix in equation set (7), we should replace the $\boldsymbol{A}$ in the second block row with $\bar{\boldsymbol{A}}$. Equation ℓ in the second block row is

$$\sum_{j=1}^{n} a_{\ell,j} x_j = b_\ell \text{ for } \ell \in \{1, \ldots, m\}. \tag{10}$$

Analogous to $a_{i,j}$, let $\bar{a}_{i,j}$ be the element in row i and column j of $\bar{\boldsymbol{A}}$. From step S2 above, we know that if security i is bound with $i \in \texttt{UP}$ or $i \in \texttt{DN}$, we must have $\bar{a}_{i,k} = 0$ for all k. Therefore, to maintain equality in Equation (3), we need to add back these zeroed-out quantities with the appropriate bounds contained in $\boldsymbol{k}$ from step S4. Equation (10) therefore becomes

$$\sum_{j=1}^{n} \bar{a}_{\ell,j} x_j + \sum_{j=1}^{n} a_{\ell,j} k_j = b_\ell \text{ for } \ell \in \{1, \ldots, m\},$$

which is written in matrix form as

$$\bar{\boldsymbol{A}}\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{k}.$$

Combining these two modifications of Equation (7), we find that we can account for securities in all sets (IN, UP, or DN) with the equation

$$\begin{bmatrix} \bar{\boldsymbol{C}} & \bar{\boldsymbol{A}}^\top \\ \bar{\boldsymbol{A}} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{\mu}} \\ \mathbf{0} \end{bmatrix} \lambda_{\text{E}} + \begin{bmatrix} \mathbf{0} \\ \boldsymbol{b} \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \boldsymbol{A}\boldsymbol{k} \end{bmatrix} \tag{11}$$

where $\bar{\boldsymbol{C}}$, $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{\mu}}$, and $\boldsymbol{k}$ are defined to have entries as follows:

$$\bar{\sigma}_{i,j} = \begin{cases} \sigma_{i,j} & \text{for } (i \in \texttt{IN}) \text{ and } (j \in \texttt{IN}), \\ 1 & \text{for } (i \notin \texttt{IN}) \text{ and } (i = j), \\ 0 & \text{otherwise.} \end{cases} \qquad k_i = \begin{cases} u_i & \text{for } i \in \texttt{UP}, \\ d_i & \text{for } i \in \texttt{DN}, \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

$$\bar{a}_{i,j} = \begin{cases} a_{i,j} & \text{for } j \in \texttt{IN}, \\ 0 & \text{otherwise.} \end{cases} \qquad \bar{\mu}_i = \begin{cases} \mu_i & \text{for } i \in \texttt{IN}, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

For brevity, we define

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{A}^\top \\ \boldsymbol{A} & \mathbf{O} \end{bmatrix} \text{ and } \bar{\boldsymbol{M}} = \begin{bmatrix} \bar{\boldsymbol{C}} & \bar{\boldsymbol{A}}^\top \\ \bar{\boldsymbol{A}} & \mathbf{O} \end{bmatrix} \tag{14}$$

and the top block row of $\boldsymbol{M}$ as $\boldsymbol{P}$,

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{A}^\top \end{bmatrix}. \tag{15}$$

The matrix $\bar{\boldsymbol{M}}$ is guaranteed to be non-singular (see Markowitz [1987]). Therefore, Equation (11) can be solved to give

$$\begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \boldsymbol{\alpha} + \boldsymbol{\beta}\lambda_{\text{E}}, \tag{16}$$

where

$$\boldsymbol{\alpha} = \bar{\boldsymbol{M}}^{-1} \begin{bmatrix} \mathbf{0} \\ \boldsymbol{b} - \boldsymbol{A}\boldsymbol{k} \end{bmatrix} \text{ and } \boldsymbol{\beta} = \bar{\boldsymbol{M}}^{-1} \begin{bmatrix} \bar{\boldsymbol{\mu}} \\ \mathbf{0} \end{bmatrix}. \tag{17}$$

Substituting Equation (17) into Equation (5) gives a closed-form expression for the derivative of the Lagrangian with respect to the portfolio weights

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = \boldsymbol{\gamma} + \boldsymbol{\delta}\lambda_{\text{E}} \tag{18}$$

where

$$\boldsymbol{\gamma} = \boldsymbol{P}\boldsymbol{\alpha} \text{ and } \boldsymbol{\delta} = \boldsymbol{P}\boldsymbol{\beta} - \boldsymbol{\mu}. \tag{19}$$

Equations (16) through (19) provide us with the tools to trace out the efficient frontier. Equation (16) describes the straight-line segments along the efficient frontier for given IN, UP, and DN sets. As λ_{E} changes, the portfolio $\boldsymbol{x}$ attains discrete critical points (i.e., corner portfolios) at which the compositions of the IN, UP, and DN sets change. Jacobs, Levy, and Starer [2012] provide an alternative set of equations that use compression of the dimensions of the $\boldsymbol{C}$ matrix to include entries only for securities that are IN. In addition, Jacobs, Levy, and Starer [2012] provide equations for benchmark-sensitive portfolios, beta-constrained portfolios, long-short portfolios, and enhanced active-equity portfolios.

The critical line algorithm is a computationally efficient method tracing out the entire efficient frontier by finding successive critical values. The algorithm starts with λ_{E} effectively at infinity, at which point maximizing utility amounts to maximizing expected portfolio return subject to the required constraints. This is a simple linear programming problem whose solution provides the initial portfolio and the initial contents of the IN, UP, and DN sets. The algorithm then finds the largest value of λ_{E} that is smaller than its current value, and at which the contents of IN, UP, or DN change. This is the next critical value.

At any critical value, one of four events can take place:

L1. An IN security i can move to UP. From Equation (16), $x_i = \alpha_i + \beta_i\lambda_{\text{E}}$. Since x_i is attempting to increase its value as λ_{E} decreases, we must have $\beta_i < 0$. Also, since the security attains its upper bound, its weight must satisfy $x_i = u_i$. Combining these facts, we find that the corner portfolio occurs when $\lambda_{\text{E}} = \lambda_{i,\texttt{IU}}$ with

$$\lambda_{i,\texttt{IU}} = \frac{u_i - \alpha_i}{\beta_i}, \quad \beta_i < 0. \tag{20}$$

L2. An IN security i can move to DN. From Equation (16), $x_i = \alpha_i + \beta_i\lambda_{\text{E}}$. Since x_i is attempting to decrease its value as λ_{E} decreases, we must have $\beta_i > 0$. Also, since the security attains its lower bound, its weight must satisfy $x_i = d_i$. Combining these facts, we find that the corner portfolio occurs when $\lambda_{\text{E}} = \lambda_{i,\texttt{ID}}$ with

$$\lambda_{i,\texttt{ID}} = \frac{d_i - \alpha_i}{\beta_i}, \quad \beta_i > 0. \tag{21}$$

L3. An UP security i can move to IN. From Equation (19), $\partial\mathcal{L}/\partial x_i = \gamma_i + \delta_i\lambda_{\text{E}}$. In UP, this derivative is negative. For the security to move IN, the derivative must increase to zero as λ_{E} decreases. Therefore $\delta_i < 0$ and, since the derivative must attain zero, $\gamma_i + \delta_i\lambda_{\text{E}} = 0$. Combining these facts, we find that the corner portfolio occurs when $\lambda_{\text{E}} = \lambda_{i,\texttt{UI}}$ with

$$\lambda_{i,\texttt{UI}} = -\frac{\gamma_i}{\delta_i}, \quad \delta_i < 0. \tag{22}$$

L4. A DN security i can move to IN. From Equation (19), $\partial\mathcal{L}/\partial x_i = \gamma_i + \delta_i\lambda_{\text{E}}$. In DN, this derivatives is positive. For the security to move IN, the derivative must decrease to zero as λ_{E} decreases. Therefore $\delta_i > 0$ and, since the derivative must attain zero,

$\gamma_i + \delta_i \lambda_{\text{E}} = 0$. Combining these facts, we find that the corner portfolio occurs when $\lambda_{\text{E}} = \lambda_{i,\texttt{DI}}$ with

$$\lambda_{i,\texttt{DI}} = -\frac{\gamma_i}{\delta_i}, \quad \delta_i > 0. \tag{23}$$

The largest of $\lambda_{i,j}$ over all $i \in \{1, \ldots, n\}$ and $j \in \{\texttt{IU}, \texttt{ID}, \texttt{UI}, \texttt{DI}\}$ determines the value of λ_{E} at the next corner, or if the algorithm terminates.

A convenient way to find which security is changing and the direction in which it is changing is to collect the ratios $\lambda_{i,j}$ for all $i \in \{1, \ldots, n\}$ and $j \in \{\texttt{IU}, \texttt{ID}, \texttt{UI}, \texttt{DI}\}$ into the matrix $\boldsymbol{\Lambda}$ as follows:

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_{1,\texttt{IU}} & \lambda_{1,\texttt{ID}} & \lambda_{1,\texttt{UI}} & \lambda_{1,\texttt{DI}} \\ \lambda_{2,\texttt{IU}} & \lambda_{2,\texttt{ID}} & \lambda_{2,\texttt{UI}} & \lambda_{2,\texttt{DI}} \\ \vdots & & & \vdots \\ \lambda_{n,\texttt{IU}} & \lambda_{n,\texttt{ID}} & \lambda_{n,\texttt{UI}} & \lambda_{n,\texttt{DI}} \end{bmatrix}. \tag{24}$$

All entries in $\boldsymbol{\Lambda}$ that are not defined in Equations (20) through (23) are assigned a value of $-\infty$. The column of $\boldsymbol{\Lambda}$ that contains the largest value represents the type of transition made at the next corner. If the first column contains the maximum value, then the transition is for a security moving from `IN` to `UP`, and so forth. The number of the row of $\boldsymbol{\Lambda}$ that contains the largest value represents the index of the security that makes the transition.

We now have all the information necessary to implement the CLA for portfolios with a constraint set that includes lower and upper bounds. In brief, the algorithm proceeds as follows:

- Read data or compute where necessary $\boldsymbol{\mu}$, $\boldsymbol{C}$, $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{d}$, and $\boldsymbol{u}$.
- Use linear programming to find the portfolio $\boldsymbol{x}$ that maximizes expected portfolio return $\boldsymbol{\mu}^\top \boldsymbol{x}$ subject to the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{d} \leq \boldsymbol{x} \leq \boldsymbol{u}$, and initialize the `IN`, `UP`, and `DN` sets.
- Set $\lambda_{\text{E}} = \infty$.
- Repeat while $\lambda_{\text{E}} \geq 0$:
  - With the current `IN`, `UP`, and `DN` sets, calculate $\bar{\boldsymbol{C}}$, $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{M}}$, $\bar{\boldsymbol{\mu}}$, and $\boldsymbol{k}$ using steps S1 through S4.
  - Calculate $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using Equation (17), and $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ using Equation (19).
  - Compute the $\lambda_{i,j}$ ratios using steps L1 through L4 and form the $\boldsymbol{\Lambda}$ matrix.
  - Find the maximum value of $\boldsymbol{\Lambda}$. This is the next value of λ_{E}; the critical value at the next corner portfolio.
  - Find the row that contains the maximum value of $\boldsymbol{\Lambda}$. This is the number of the security whose state will change at the next corner.
  - Find the column that contains the maximum value of $\boldsymbol{\Lambda}$. This is an indicator of the state change that the security will undergo at the next corner portfolio.
  - With the state transition thus defined, compute the next `IN`, `UP`, and `DN` sets.

– If $\lambda_{\text{E}} \geq 0$, compute and store the portfolio $\boldsymbol{x}$ at the next corner using the top block of Equation (16).

We describe the details of the algorithm below.

3 CLA Python Implementation

This section describes a simplified Python implementation of the CLA that maps out the efficient frontier. This simplified version omits the two-stage simplex algorithm for initializing the CLA. In addition, it omits the efficient method of updating inverse matrices, instead relying on the fast and reliable linear algebra module of Python.

Sample Python code is provided in Appendix A. Note, however, that the code is provided for illustrative purposes only and should not be used for production purposes. In particular, it contains no error checking that would be essential for production code. Further, the code is written for readability and correlation with equations presented here. Little attention is paid to the use of proper idiomatic Python.

For reference, the code is written in blocks with comments that include labels in the form `--Axx--`, where `xx` is a two-digit number. The description below refers to those labels.

The first steps necessary for the CLA are to define functions, read data, and initialize parameters. Blocks `--A01--` and `--A02--` define functions that set rows or columns of matrices or vectors to zero. These are used in the conversion of $\boldsymbol{\mu}$ to $\bar{\boldsymbol{\mu}}$ and $\boldsymbol{A}$ to $\bar{\boldsymbol{A}}$ in steps S2 and S3. Block `--A03--` is used to convert $\boldsymbol{C}$ to $\bar{\boldsymbol{C}}$ in step S1.

Block `--A04--` is a function `initport` that initializes the portfolio for λ_{E} effectively infinite. The function computes the maximum return portfolio that satisfies the constraint. In this case, the only constraints are lower bounds `lb` and upper bounds `ub`, and the budget constraint. The function uses the method described by Sharpe in https://web.stanford.edu/~wfsharpe/mia/opt/mia_opt3.htm. A more general initialization procedure is provided in the VBA code in Chapter 13 of Markowitz and Todd [2000], which performs initialization using a two-stage simplex algorithm.

Block `--A05--` reads the data into the returns array `ret`, and drops the first column, which is assumed to contain the date. Block `--A06--` sets the initial parameters. Specifically, it determines the number of securities from the shape of the `ret` matrix, it initializes the lower bounds `lb` and upper bounds `lb` on the portfolio weights, and it initializes the constraint matrix `A` and vector `b`. In this case, the only constraint is the budget constraint, so the number of constraints `m` is one. In this simple code, we are expecting a small number of securities, so we set the lower weight bound to 10% and the upper weight bound to 50%. These, obviously, should be modified to match the user's needs. Block `--A06--` sets a tolerance `tol` for testing whether or not variables should be considered to be zero. Block `--A07--` computes the sample mean `mu` and sample covariance `C` of the return matrix.

The first step of the CLA is performed in Block `--A08--`, which calls function `initport` to find the first portfolio on the efficient frontier. Block `--A09--` uses this first portfolio to create the first state vector `s`. The state vector contains $+1$, -1, and 0 in positions corresponding to securities that are in the `UP`, `DN`, and `IN` sets, respectively. Block `--A10--` sets up the `P` matrix defined in Equation (15) for calculating the derivative in Equation (18).

Finally, in this set of blocks, block `--A11--` initializes storage for the quantities calculated in the CLA.

The CLA loop for stepping from corner portfolio to corner portfolio starts in block `--A12--`. Here, we use the variable `lam` to represent λ_{E}. The loop proceeds as long as λ_{E} is positive, or until it is broken when no further corner portfolios are found.

Inside the CLA loop, block `--A13--` uses the current state vector `s` to create the logical vectors `UP`, `DN`, and `IN` that represent their corresponding sets.

Block `--A14--` creates $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{C}}$, and $\bar{\boldsymbol{M}}$, represented by variables `Abar`, `Cbar`, and `Mbar`. This block first creates the vector `io` containing the indices of the weights that are not `IN`. It then creates `Abar` and `Cbar` from `A` and `C` using `io` and the functions `zerorows` and `bar`.

Block `--A15--` computes array `k` representing the vector $\boldsymbol{k}$ in Equation (12). It then calculates the right-hand side vectors $[\mathbf{0};\ \boldsymbol{b} - \boldsymbol{A}\boldsymbol{k}]$ and $[\bar{\boldsymbol{\mu}};\ \mathbf{0}]$ used in Equation (17).

Block `--A16--` computes the vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\delta}$ as defined in Equations (17) and (19). In this code, we merely call Numpy's function `linalg.solve()` to invert $\bar{M}$ at each corner portfolio. However, it is more efficient to keep a copy of the inverse $\bar{\boldsymbol{M}}^{-1}$ and to update this inverse as the elements of `IN`, `UP`, and `DN` change. Details of efficient updating methods are provided in Markowitz and Todd [2000].

Blocks `--A17--` though `--A24--` comprise a group that finds the maximum of $\lambda_{i,\texttt{IU}}$, $\lambda_{i,\texttt{ID}}$, $\lambda_{i,\texttt{UI}}$, and $\lambda_{i,\texttt{DI}}$ using Equations (20) through (23). To do this, the code places the ratios into the two-dimensional array `L` representing $\boldsymbol{\Lambda}$ in Equation (24).

Specifically, block `--A17--` prepares `L` and sets all of its values to $-\infty$. Blocks `--A18--` through `--A21--` fill `L` with computed values of the ratios $\lambda_{i,\texttt{IU}}$, $\lambda_{i,\texttt{ID}}$, $\lambda_{i,\texttt{UI}}$, and $\lambda_{i,\texttt{DI}}$ for all i for which the ratios are defined. Since Python uses zero-based indexing, the value of $\lambda_{i,j}$ is contained in array element `L[i-1,j-1]`.

If all entries of `L` are negative, no more corners exist. Block `--A22--` checks for this condition, and breaks the loop if it is satisfied.

Using the logic described in the paragraph following Equation (24), block `--A23--` finds which row of `L` contains the maximum value, and thus which security is about to change state. Block `--A24--` finds which column of `L` contains the maximum value, and thus what type of state change is about to occur. Block `--A25--` sets the value of `lam` to the largest value of `L`. Block `--A26--` updates the state vector `s`.

Block `--A27--` computes the portfolio vector `x` at the corner using Equation (16). It then computes the expected return and the variance of the corner portfolio.

Finally, block `--A28--` saves the portfolio vector, the state vector, the variance, the return, and the value of `lam` at the current corner. The loop then repeats with the new value of `lam` and the state vector `s`, or the program terminates if `lam` ≤ 0.

When the program terminates, the workspace contains the matrix `X` whose columns are the portfolio weights at each corner, and the vector `LAM` that contains the values of $\lambda_{\text{E}} =$ `lam` at each corner. The workspace also contains extra information in the form of the matrix `S` whose columns are the state vector at each corner, the vector `R` that contains the expected portfolio return at each corner, and the vector `V` that contains the portfolio's variance at each corner.

Recall from the top block row of Equation (16) that an efficient portfolio is a piece-wise linear function of the corner portfolios that surround it. Therefore, to find the portfolio corresponding to any particular value of λ_{E} that is not a critical value, one must find the two

	Security		
Year	1	2	3
1937	-0.173	-0.318	-0.319
1938	0.098	0.285	0.076
1939	0.200	-0.047	0.381
1940	0.030	0.104	-0.051
1941	-0.183	-0.171	0.087
1942	0.067	-0.039	0.262
1943	0.300	0.149	0.341
1944	0.103	0.260	0.227
1945	0.216	0.419	0.352
1946	-0.046	-0.078	0.153
1947	-0.071	0.169	-0.099
1948	0.056	-0.035	0.038
1949	0.038	0.133	0.273
1950	0.089	0.732	0.091
1951	0.090	0.021	0.054
1952	0.083	0.131	0.109
1953	0.035	0.006	0.210
1954	0.176	0.908	0.112
$\boldsymbol{\mu}$	0.062	0.146	0.128
$\boldsymbol{\sigma}$	0.016	0.091	0.031

Table 1: Returns of Three Securities

critical values $\lambda_{(a)}$ and $\lambda_{(b)}$ immediately above and below λ_{E}, together with the corresponding corner portfolios $\boldsymbol{x}_{(a)}$ and $\boldsymbol{x}_{(b)}$. Then, the desired portfolio is simply

$$\boldsymbol{x}(\lambda_{\text{E}}) = \boldsymbol{x}_{(b)} + \frac{\lambda_{\text{E}} - \lambda_{(b)}}{\lambda_{(a)} - \lambda_{(b)}} \left(\boldsymbol{x}_{(a)} - \boldsymbol{x}_{(b)}\right). \tag{25}$$

3.1 Example

For our basic example, we will use the data provided on page 195 of Markowitz [1959], and repeated here in Table 1 for convenience. The table lists returns for three securities for the 18 years 1937 through 1954, together with the securities' sample mean returns $\boldsymbol{\mu}$ and sample standard deviations $\boldsymbol{\sigma}$.

Saving this table in the tab-separated variable file `example.tsv` and running the Python script from Appendix A, we obtain the critical values and corner portfolios (rounded to four decimal places) in Table 2. The table shows that the maximum return portfolio subject to the constraints is $\boldsymbol{x} = [0.1,\ 0.5,\ 0.4]$. This portfolio assigns the highest allowable weight to security 2 since this security has the highest sample mean return, and the lowest allowable weight to security 1 since this security has the lowest sample mean return. The remaining weight is assigned to security 3.

	Security		
λ_E	1	2	3
∞	0.1000	0.5000	0.4000
1.7567	0.1000	0.5000	0.4000
1.2203	0.1000	0.4000	0.5000
0.3142	0.1000	0.4000	0.5000
0.0973	0.3764	0.1236	0.5000
0.0853	0.4644	0.1000	0.4356
0.0770	0.5000	0.1000	0.4000
0	0.5000	0.1000	0.4000

Table 2: Critical Values and Corner Portfolios

The algorithm finds the first corner portfolio at $\lambda_E = 1.757$, and then at successively lower values until it reaches a minimum at $\lambda_E = 0.077$, which corresponds to the lowest variance portfolio subject to the constraints. The table extrapolates this final portfolio down to $\lambda_E = 0$. At this lowest end of the efficient frontier, the algorithm has now assigned the lowest allowable weight to security 2 and the highest allowable weight to security 1, which in this case happen to have the highest and lowest standard deviations, respectively. Again, the remaining weight is assigned to security 3. Figure 1 shows the reversal of the roles of securities 1 and 2, which plots the weights of the securities as a function of λ_E. Figure 2 shows the corresponding efficient frontier in which portfolios between critical values are obtained using Equation (25).

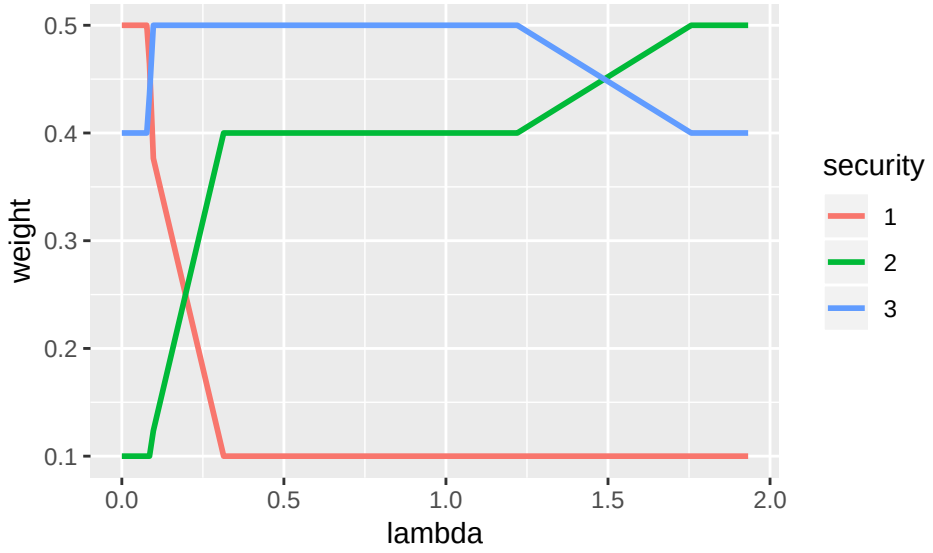

Figure 1: Portfolio Weights as a Function of λ_E

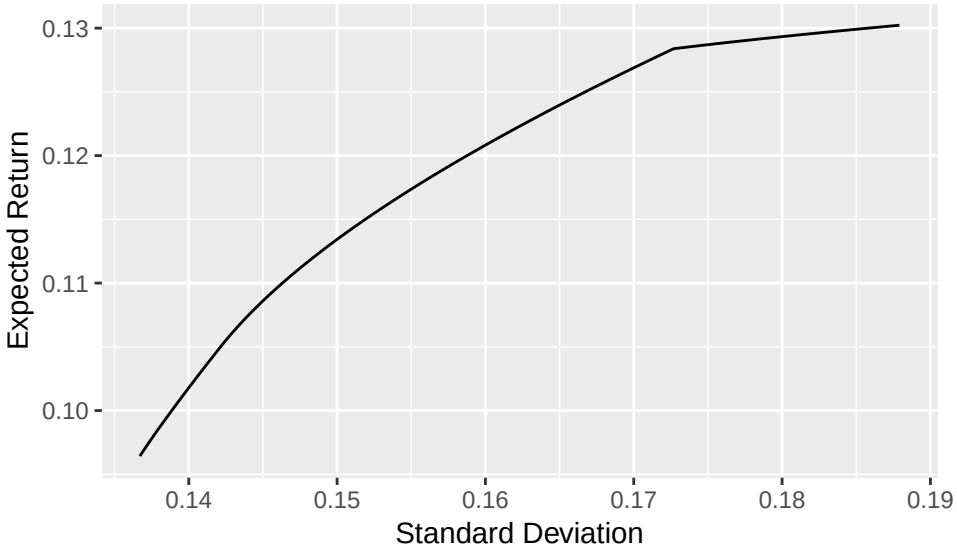

Figure 2: Efficient Frontier

4 The CLA for Semivariance

Risk is associated with the variability of return. Most often, the statistic used to measure variability is variance. Since variance uses the square of the difference from the mean, large positive deviations from the mean will contribute to an increase in perceived risk. However, most investors do not consider large positive returns to be risky. But they do consider large negative returns to be risky. Therefore, a risk measure should ideally be asymmetric and assign more weight to negative returns. The semivariance is a natural measure that extends variance so as to introduce this asymmetry.

For a random variable R with mean μ, the semivariance is defined as the expected squared deviation from the mean when that deviation is negative. That is

$$s^2 = \mathbb{E}\left[\left\{(R-\mu)^-\right\}^2\right],$$

where the notation $(\cdot)^-$ denotes the negative part of the argument in parenthesis. That is, for any x,

$$(x)^- = \begin{cases} x & \text{if } x < 0, \\ 0 & \text{otherwise.} \end{cases}$$

In this chapter, we will use a slightly generalized version of semivariance that we define relative to a reference return rather than relative to the mean. The traditional semivariance is then simply a special case.

With return measurements r_t for $t = 1, \ldots, T$, we define the population semivariance with respect to the reference return ϱ as

$$s_\varrho^2 = \frac{1}{T}\sum_{t=1}^{T}\left\{(r_t - \varrho)^-\right\}^2 \tag{26}$$

Advantageously with semivariance, only returns below the reference return contribute to the measure of risk. Returns above the reference point are given no weight. Two particular choices of reference point interest us. If we choose the reference point to be the mean, then $s_\mu^2 \equiv s^2$ is the same as the standard definition of semivariance. If we choose the reference point to be zero, then s_0^2 is the squared downside deviation; the word downside referring to any negative return.

For a symmetrical return distribution, the variance is twice the semivariance; $\sigma^2 = 2s^2$. For a left-skewed distribution, $\sigma^2 < 2s^2$, and the opposite is true for a right-skewed distribution. Therefore, the ratio $\sigma^2/(2s^2)$ is a useful indirect measure of skewness. For portfolios with the same expected return and variance, optimization based on minimizing s^2 yields the portfolio with the smallest skewness to the left.

As noted in Markowitz [1959, page 193], "Several conflicting considerations influence the choice of [variance] or [semivariance] as the measure of variability in a portfolio analysis. These considerations include cost, convenience, familiarity, and the desirability of the portfolios produced by the analysis. ... Analyses based on [semivariance] tend to produce better portfolios than those based on [variance]. Variance considers extremely high and extremely low returns equally undesirable. An analysis based on [variance] seeks to eliminate both extremes. An analysis based on s, on the other hand, concentrates on reducing losses."

Importantly, with regard to familiarity, Markowitz [1959] states that "[F]amiliarity is a transient thing: use can make [semivariance] as familiar as [variance]." In this section, we hope to aid the process of familiarization.

It is straightforward to extend population variance to population semivariance. For a population of T realizations of the returns of n securities, we know that the population covariance between securities i and j is

$$\sigma_{ij} = \frac{1}{T}\sum_{t=1}^{T}(r_{t,i} - \mu_i)(r_{t,j} - \mu_j) \tag{27}$$

where $r_{t,k}$ is the return of security k over time period t, and μ_k is the average return of security k. If $\boldsymbol{R}$ is the $T \times n$ returns matrix with $r_{t,k}$ in its t-th row and k-th column, with $\boldsymbol{\iota}$ and $\boldsymbol{\mu}$ as previously defined, the elements given in Equation (27) be written in matrix form as

$$\boldsymbol{C} = \frac{1}{T}\left(\boldsymbol{R} - \boldsymbol{\iota}\boldsymbol{\mu}^\top\right)^\top \left(\boldsymbol{R} - \boldsymbol{\iota}\boldsymbol{\mu}^\top\right).$$

Conveniently, we can write the covariance matrix in square root form as $\boldsymbol{C} = \boldsymbol{B}^\top\boldsymbol{B}$ where the square root matrix is

$$\boldsymbol{B} = \frac{1}{\sqrt{T}}\left(\boldsymbol{R} - \boldsymbol{\iota}\boldsymbol{\mu}^\top\right).$$

Note, of course, for $T > n$, that this square root matrix is not unique. That is, many different $\boldsymbol{B}$ matrices can yield the same $\boldsymbol{C}$. However, the $\boldsymbol{B}$ in our case is uniquely determined by our data. Using the square root, the variance of the return of portfolio $\boldsymbol{x}$ is

$$\sigma_{\text{P}}^2 = \boldsymbol{x}^\top\boldsymbol{B}^\top\boldsymbol{B}\boldsymbol{x} = \boldsymbol{y}^\top\boldsymbol{y}$$

where we have implicitly defined the $T \times 1$ vector of artificial variables $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{x}$. More generally, in terms of the reference return $\boldsymbol{\varrho}$, we could have defined

$$\boldsymbol{B}_{\boldsymbol{\varrho}} = \frac{1}{\sqrt{T}}\left(\boldsymbol{R} - \boldsymbol{\iota}\boldsymbol{\varrho}^\top\right) \tag{28}$$

and

$$\boldsymbol{y} = \boldsymbol{B}_{\boldsymbol{\varrho}}\boldsymbol{x}. \tag{29}$$

In this chapter, we omit the subscript from $\boldsymbol{B}$ and infer its value from the context. Usually, we deal with $\boldsymbol{\varrho} = \boldsymbol{0}$.

The artificial variable vector $\boldsymbol{y}$ and its components y_t for $t = 1, \ldots, T$ have important interpretations. In particular, y_t is the return of the portfolio over period t that is in excess of the reference return and normalized by $\sqrt{T}$. Therefore, using the negative parts of $\boldsymbol{y}$, we can write the semivariance simply as

$$\begin{aligned} s^2 &= \sum_{t=1}^{T}((y_t)^-)^2 = \left(\boldsymbol{y}^\top\right)^-\left(\boldsymbol{y}\right)^- \\ &= \left(\boldsymbol{x}^\top\boldsymbol{B}^\top\right)^-\left(\boldsymbol{B}\boldsymbol{x}\right)^-. \end{aligned}$$

In general $\left(\boldsymbol{B}\boldsymbol{x}\right)^{-} \neq (\boldsymbol{B})^{-}(\boldsymbol{x})^{-}$. So, unlike variance, we cannot write semivariance in a simple quadratic form $s^2 = ((\boldsymbol{x})^{-})^{\top}\, \boldsymbol{S}\, ((\boldsymbol{x})^{-})$ for some universally applicable matrix $\boldsymbol{S}$. If this were possible, the problem could be solved with standard quadratic programming. Instead, we will find that there exist regions in which semicovariance matrices are locally applicable. In those regions, the problem *is* quadratic. To understand this, we need to examine the concept of profitability lines.

The excess portfolio return for any period is the inner product of the vector of excess returns for that period and the vector of portfolio weights. A period t is unprofitable if the excess return for all permissible values of $\boldsymbol{x}$ is negative; i.e.,

$$y_t \propto \sum_{i=1}^{n} (r_{t,i} - \varrho_i) x_i = \left(\boldsymbol{r}^t - \boldsymbol{\varrho}^{\top}\right) \boldsymbol{x} < 0, \tag{30}$$

where $\boldsymbol{r}^t$ is the t-th row of matrix $\boldsymbol{R}$. Since the sum of all weights must be one, this inequality defines a half-plane in $n-1$-dimensional portfolio space. The unprofitable region for period t is the set all points $\boldsymbol{x}$ that satisfy the inequality, and the profitability line is the locus of points $\boldsymbol{x}$ that satisfy the equality $(\boldsymbol{r}^t - \boldsymbol{\varrho}^{\top})\, \boldsymbol{x} = 0$.

We therefore have stated a criterion according to which observations should, or should not, be included in the computation of the semivariance.

The returns for period t will be included in the semivariance computation (Equation (26)) if that period is unprofitable.

In other words, if we define $\texttt{IS} \subseteq \{1, 2, \ldots, T\}$ to be the set of time periods that are included in the semicovariance computation in a region of $\boldsymbol{x}$, then period t is a member of `IS` if the t-th element of $\boldsymbol{y}$ is negative. That is,

$$y_t < 0 \longrightarrow t \in \texttt{IS}. \tag{31}$$

Consider a practical example with $\boldsymbol{\varrho} = \boldsymbol{0}$. From Table 1, we see that any long-only portfolio would be unprofitable in 1937 since all securities had negative returns in that year. In contrast, in 1938, any long-only portfolio would be profitable since all securities had positive returns in that year. Therefore the year 1937 should always be included in calculations of the semivariance, but the year 1938 should never be included.

The case is not so trivial for years that have both positive and negative returns. Using Equation (30) in these cases, we have the criterion that we should include period t if

$$r_{t,1}x_1 + r_{t,2}x_2 + r_{t,3}x_3 < 0.$$

Or, equivalently, since $x_3 = 1 - x_1 - x_2$, we should include period t if

$$(r_{t,1} - r_{t,3})x_1 + (r_{t,2} - r_{t,3})x_2 + r_{t,3} < 0. \tag{32}$$

This defines the unprofitable region for year t. For the year, 1947, for example, we have $\boldsymbol{r}^t = [r_{t,1},\ r_{t,2},\ r_{t,3}] = [-0.071,\ +0.169,\ -0.099]$, and from Equation (32), the unprofitable region is

$$0.028x_1 + 0.268x_2 - 0.099 < 0.$$

The corresponding profitability line for 1947 is displayed in Figure 3. The shaded region is the intersection of the unprofitable region, the non-negativity constraints $(x_1, x_2, x_3 \geq 0)$,

and the budget constraint ($x_1 + x_2 + x_3 = 1$). Any portfolios in this region should include 1947 in the calculation of the semivariance.

In a region of any portfolio vector $\boldsymbol{x}$, therefore, there is a set of time periods that should be included in the computation of the semivariance. Let $\mathbb{B}$ be the matrix obtained from $\boldsymbol{B}$ by including only those rows (time periods) that lead to unprofitable portfolios in the region of $\boldsymbol{x}$. In this case, the semivariance can be written in the simple quadratic form

$$s^2 = \left(\boldsymbol{x}^\top \boldsymbol{B}^\top\right)^- \left(\boldsymbol{B}\boldsymbol{x}\right)^- = \boldsymbol{x}^\top \mathbb{B}^\top \mathbb{B}\, \boldsymbol{x} = \boldsymbol{x}^\top \boldsymbol{S}\, \boldsymbol{x}$$

where we have defined the locally applicable semicovariance matrix (LSM) as $\boldsymbol{S} = \mathbb{B}^\top \mathbb{B}$.

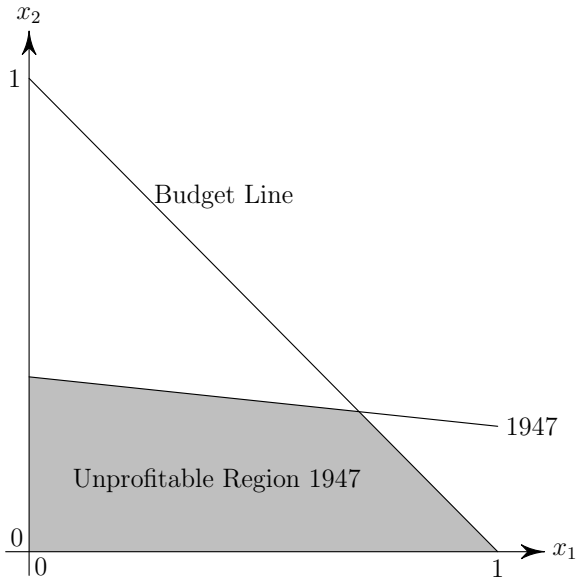

Figure 3: Profitability Line and Unprofitable Region for 1947

Given a set `IS` and a critical line for that set that contains the current portfolio $\boldsymbol{x}$, we can find the value of λ_{E} at which the constituents of `IS` change. We can thereby determine the borders of the current region. A region is locally applicable as long as portfolios within it are unprofitable for all periods.

At the border of a region, moving along the critical line as λ_{E} decreases, the current portfolio may be become profitable for one period that was previously unprofitable (so that the period needs to be included in the semivariance computation), or a period that was unprofitable may become profitable (so that the period must be excluded from the semivariance calculation). Therefore, the borders are defined by lines of zero profitability.

At zero profitability, $\boldsymbol{B}_{\boldsymbol{\varrho}}\boldsymbol{x} = \boldsymbol{0}$. Using Equation (28) for the specific case $\boldsymbol{\varrho} = \boldsymbol{0}$, profitability lines are therefore defined by

$$\boldsymbol{R}\,\boldsymbol{x} = \boldsymbol{0}. \tag{33}$$

But, considering only the top block of Equation (16), we know that $\boldsymbol{x} = \boldsymbol{\alpha} + \boldsymbol{\beta}\lambda_{\text{E}}$. Using this in Equation (33) and defining $\boldsymbol{\xi} = \boldsymbol{R}\boldsymbol{\alpha}$ and $\boldsymbol{\zeta} = \boldsymbol{R}\boldsymbol{\beta}$, the value of λ_{E} at which the critical

line crosses a profitability line for period t is

$$\lambda_{\mathrm{E},t} = -\frac{\xi_t}{\zeta_t} \text{ for } \zeta_t \neq 0 \tag{34}$$

The particular period t entering or leaving the calculation of the LSM is the one corresponding to the maximum value of $\lambda_{\mathrm{E},t}$ that is less than the current value of λ_{E}.

We are now ready to describe the CLA for semivariance optimization. In brief, the algorithm proceeds as follows:

1. Read data or compute where necessary $\boldsymbol{\mu}$, $\boldsymbol{C}$, $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{d}$, and $\boldsymbol{u}$.
2. Using linear programming, find the portfolio $\boldsymbol{x}$ that maximizes expected portfolio return $\boldsymbol{\mu}^\top \boldsymbol{x}$ subject to the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ and $\boldsymbol{d} \leq \boldsymbol{x} \leq \boldsymbol{u}$, and initialize the `IN`, `UP`, and `DN` sets. This is independent of semivariance.
3. For the initial portfolio vector, calculate the locally applicable semicovariance matrix.
4. Set $\lambda_{\mathrm{E}} = \infty$.
5. Repeat while $\lambda_{\mathrm{E}} \geq 0$:
    a. With the current `IN`, `UP`, and `DN` sets, together with the current locally applicable semicovariance matrix, calculate $\bar{\boldsymbol{C}}$, $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{M}}$, and $\boldsymbol{k}$ using steps S1 through S4.
    b. Calculate $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using Equation (17), and $\boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ using Equation (19).
    c. Compute the $\lambda_{i,j}$ ratios using steps L1 through L4 and form the $\boldsymbol{\Lambda}$ matrix.
    d. Find the maximum value of $\boldsymbol{\Lambda}$. Call this λ_{sec} to indicate that it is the λ_{E} at which a security's state changes.
    e. Find the row that contains the maximum value of $\boldsymbol{\Lambda}$. This is the number of the security whose state will change at the next corner.
    f. Find the column that contains the maximum value of $\boldsymbol{\Lambda}$. This is an indicator of the state change that the security will undergo at the next corner portfolio.
    g. Using Equation (34), find the maximum value of $\lambda_{\mathrm{E},t}$ at which the next profitability line will be crossed. Call this λ_{obs} to indicate that is the $\lambda_{\mathrm{E},t}$ at which the returns of the observation in period t will either be added to, or removed from, the calculation of the semivariance.
    h. If $\lambda_{\mathrm{obs}} > \lambda_{\mathrm{sec}}$ then an observation is leaving or entering the semicovariance calculation. Proceed as follows:
        * Set $\lambda_{\mathrm{E}} = \lambda_{\mathrm{obs}}$. This is the critical value at the next corner portfolio.
        * Calculate the portfolio at the next corner using $\boldsymbol{x} = \boldsymbol{\alpha} + \boldsymbol{\beta}\lambda_{\mathrm{E}}$.
        * Using Equation (31), decide what observations should be included and excluded in the next segment of the critical line.
        * Calculate the locally applicable semicovariance matrix and the locally applicable $\boldsymbol{M}$ matrix.

i. Else, a security's state is changing at the next corner. Proceed as follows:
   * Set $\lambda_{\text{E}} = \lambda_{\text{sec}}$. This is the critical value at the next corner portfolio.
   * Calculate the portfolio at the next corner using $\boldsymbol{x} = \boldsymbol{\alpha} + \boldsymbol{\beta}\lambda_{\text{E}}$.
   * Use the information from steps [e.] and [f.] to set the next state vector (and thereby the next `IN`, `UP`, and `DN` sets).

j. Store information about the current corner.

We describe the details of the algorithm below.

4.1 Semivariance CLA R Implementation

We describe the CLA for mean-semivariance optimization with reference to the sample R code given in Appendix B. As in the Python code in Appendix A, the first few blocks (`--B01--` through `--B04--`) define functions for creating the matrices $\bar{\boldsymbol{A}}$, $\bar{\boldsymbol{C}}$, and $\bar{\boldsymbol{M}}$, and vector $\bar{\boldsymbol{\mu}}$, and for initializing the portfolio.

Blocks `--B05--` through `--B36--` define the `dscla` function for downside optimization using the CLA. The function's input parameters are the return matrix $\boldsymbol{R}$ represented by `R`, the constraint matrix $\boldsymbol{A} =$ `A` and vector $\boldsymbol{b} =$ `b`, the lower and upper bound vectors $\boldsymbol{d} =$ `lb` and $\boldsymbol{u} =$ `ub`, the reference return $\boldsymbol{\varrho} =$ `refret`, and a tolerance scalar `tol` that is used to decide if variables are sufficiently small to be considered to be zero. The default values of `refret` and `tol` are zero and 10^{-9}, respectively.

Block `--B05--` finds basic properties of the returns data; the number of observations `no` and securities `ns`, and the securities' average return $\boldsymbol{\mu} =$ `mu`. Block `--06--` creates the initial portfolio vector `x` that has maximum expected return and is constrained, in this example case, only by the budget constraint and the lower and upper bounds.

Block `--B07--` creates an integer vector `state` that represent the current state of the critical line. This vector contains entries of zero corresponding to securities that are `IN`, -1 for securities that are `DN`, and $+1$ for securities that are `UP`.

Block `--B08--` is the first block specific to the CLA for semivariance. It sets up the excess return matrix `Rex` that represents $\boldsymbol{R} - \boldsymbol{\iota}\boldsymbol{\varrho}^{\top}$. It also computes the time series $\boldsymbol{y} =$ `y` of portfolio excess returns, and sets the logical vector `obs.in` that indicates which time periods should be included in the computation of the semicovariance matrix for the current portfolio vector.

In block `--B09--`, we use the indicator vector `obs.in` to create the local return matrix `Rloc` that contains only observations that would lead to unprofitable portfolios for the current region of $\boldsymbol{x}$. With this, we compute the locally applicable semicovariance matrix `Sloc`, and well as the local $\boldsymbol{M}$ matrix `Mloc` and its top block row `Ploc`.

In block `--B10--`, we allocate storage for the results of the CLA. Since we do not know *a priori* how many corners the algorithm will find, we simply allocate the first value, and then extend the result on each successful completion of the CLA loop.

Block `--B11--` starts the CLA proper. In this block, we enter the CLA loop that repeats as long as $\lambda_{\text{E}} > 0$. The variable `lam` represents λ_{E} (which has already been set to $+\infty$). Block `--B12--` converts the integer state vector `state` into logical vectors `UP`, `DN`, and `IN`. These vectors are used later in the code. The integer vector `OT` indicates the securities that are not

IN (i.e., are either UP or DN). Block --B13-- uses OT to form $\bar{\boldsymbol{S}}$, $\bar{\boldsymbol{A}}$, and $\bar{\boldsymbol{M}}$ from $\boldsymbol{S}$, $\boldsymbol{A}$, and $\boldsymbol{M}$ in accordance with steps S1 and S2, and Equation (14) above.

Block --B14-- computes the vector $\boldsymbol{k} =$ k using Equation (12) and the right-hand side vectors for finding $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ using Equation (17). With these, block --B19-- computes $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, and $\boldsymbol{\delta}$ using Equations (17) and (19).

At this stage, the current segment of the efficient frontier is fully defined. The portfolios on the segment satisfy $\boldsymbol{x} = \boldsymbol{\alpha} + \boldsymbol{\beta}\lambda_{\text{E}}$ for the current $\boldsymbol{\alpha} =$ alf.sec and $\boldsymbol{\beta} =$ bet.sec. The algorithm is therefore ready to find the value of λ_{E} at which either the portfolio state vector state changes, or the LSM Sloc changes. That value of λ_{E} is determined from the maximum of $\boldsymbol{\Lambda}$ from Equation (24) and $\lambda_{\text{E},t}$ from Equation (34). The next steps, therefore, are to calculate $\boldsymbol{\Lambda}$ and $\lambda_{\text{E},t}$.

Blocks --B17-- through --B20-- calculate the columns $\boldsymbol{\lambda}_{\texttt{IU}}$, $\boldsymbol{\lambda}_{\texttt{ID}}$, $\boldsymbol{\lambda}_{\texttt{UI}}$, and $\boldsymbol{\lambda}_{\texttt{DI}}$. Then block --B21-- juxtaposes them to form $\boldsymbol{\Lambda}$, represented by variable L.

The row that contains the maximum value of L represents which security is changing state. The index of this security, secchg, is found in block --B22--. The column that contains the maximum value of L represents the direction in which the security's state is changing. The index of this change, dirchg, is found in block --B23--. Block --B24-- stores lam.sec, the maximum value of λ_{E} for a security state change.

While blocks --B17-- through --B24--, described above, focus on what security state changes might occur, blocks --B25-- and --B26-- focus on what changes might affect the computation of the LSM. Specifically, block --B25-- computes $\boldsymbol{\xi} = \boldsymbol{R}\boldsymbol{\alpha}$ (represented by the variable num) and $\boldsymbol{\zeta} = \boldsymbol{R}\boldsymbol{\beta}$ (represented by the variable den). The maximum value of the ratio rat = num / den, where it is defined and less than the previous value of λ_{E}, is the value of λ_{E} at which the profitability region changes and observations in the LSM change. Block --B26-- finds this maximum value and assigns it to the variable lam.obs.

Block --B27-- decides whether the next change is a change of a security's state or a change of the profitability region. If lam.obs $>$ lam.sec, the next change is a profitability region change in which observations in the LSM change. On the other hand, if lam.obs $\leq$ lam.sec, the next change is a security state change. Blocks --B28-- through --B31-- handle profitability region changes, and blocks --B32-- through --B34-- handle security state changes.

For a change in profitability region, blocks --B28-- and --B29-- set the value of λ_{E} and compute the portfolio vector x at the next corner. Block --B30-- creates the vector obs.in that indicates which periods are unprofitable in the region of x using Equation (31). With this vector, it computes the LSM, Sloc. Block --B31-- sets the local matrix $\boldsymbol{M}$ in Equation (14), and its top block row $\boldsymbol{P}$.

Referring back to block --B27--, if the next value of λ_{E} had indicated a change of security state rather than a change pf profitability region, the algorithm would have skipped to block --B32--, which sets the new value of λ_{E} appropriately. Block --B33-- computes the portfolio vector at the next corner. Block --B34-- sets the new state vector state appropriately, recalling that the integer secchg is the number of the security that changes state, dirchg represents the direction of the change with a value of 1 indicating a change from IN to UP, 2 indicating a change from IN to DN, and values of 3 or 4 indicating a change to from UP or DN to IN.

We emerge from the if else statement to block --B35--, where we now have three

	Security		
λ_{E}	1	2	3
∞	0.0000	1.0000	0.0000
0.2898	0.0000	1.0000	0.0000
0.1579	0.0000	0.8902	0.1098
0.1450	0.0000	0.8704	0.1296
0.0665	0.0000	0.6623	0.3377
0.0358	0.0000	0.5205	0.4795
0.0300	0.0000	0.4919	0.5081
0.0284	0.1210	0.3567	0.5223
0.0263	0.2740	0.1858	0.5402
0.0077	0.6706	0.0000	0.3294
0	0.7688	0.0000	0.2312

Table 3: Critical Values and Corner Portfolios for Semivariance Optimization

important quantities pertaining to the current corner: the value of λ_{E} = `lam`, the portfolio vector $\boldsymbol{x}$ = `x`, and the state vector `state`. Block `--B35--` stores relevant quantities together with the current expected return and semivariance.

With the new value of λ_{E} and state vector `state`, we can go back to block `--B11--` to reenter the CLA loop, or exit the program if $\lambda_{\text{E}} \leq 0$.

In Block `--B36--`, the function saves the efficient frontier in the list `ef`, and returns it to the calling program.

To find the efficient frontier, the program loads data in Block `--B37--` and defines the constraints in block `--B38--`. Finally, in block `--B39--`, it calls the downside CLA function `dscla`.

4.2 Example

We again use the data from Table 1, but now wish to find the mean-semivariance efficient frontier rather than the mean-variance efficient frontier. Saving this table in the tab-separated variable file `example.tsv` and running the R script from Appendix B, we obtain the critical values and corner portfolios (rounded to four decimal places) in Table 3. The table shows that the maximum return portfolio subject only to the long-only and budget constraints is $\boldsymbol{x} = [0,\ 1,\ 0]$. This portfolio assigns the highest allowable weight to security 2 since this security has the highest sample mean return.

The algorithm finds the first corner portfolio at $\lambda_{\text{E}} = 0.2898$, and then at successively lower values until it reaches a minimum at λ_{E} close to zero, which corresponds to the lowest variance portfolio subject to the constraints. Figure 4 shows the weights of the securities as a function of λ_{E}, and Figure 5 shows the weights of the securities as a function portfolio return. Figure 6 shows the corresponding efficient frontier.

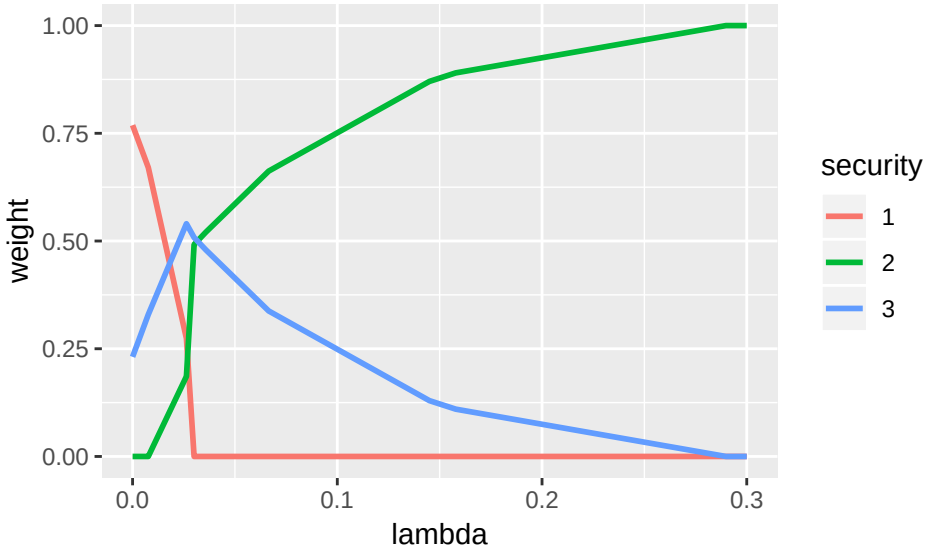

Figure 4: Semivariance Portfolio Weights as a Function of λ_{E}

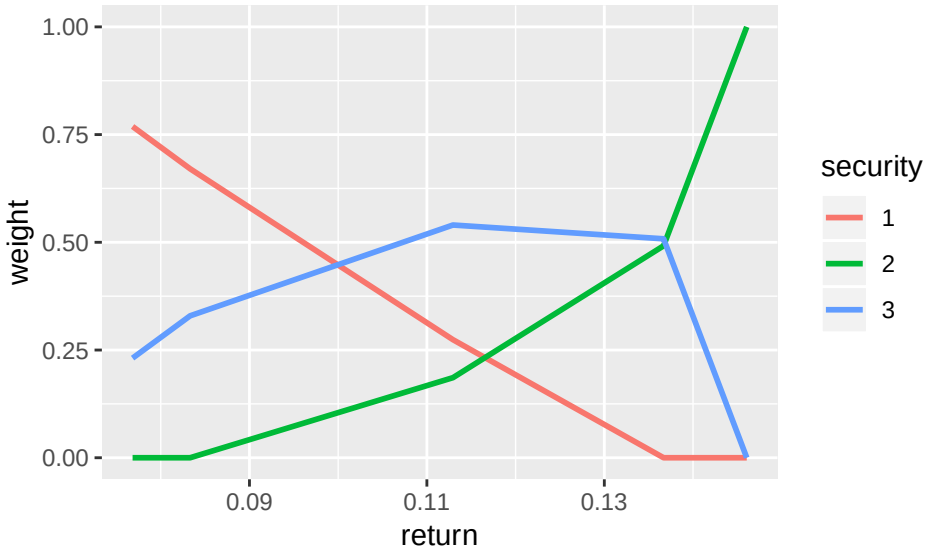

Figure 5: Semivariance Portfolio Weights as a Function of Portfolio Return

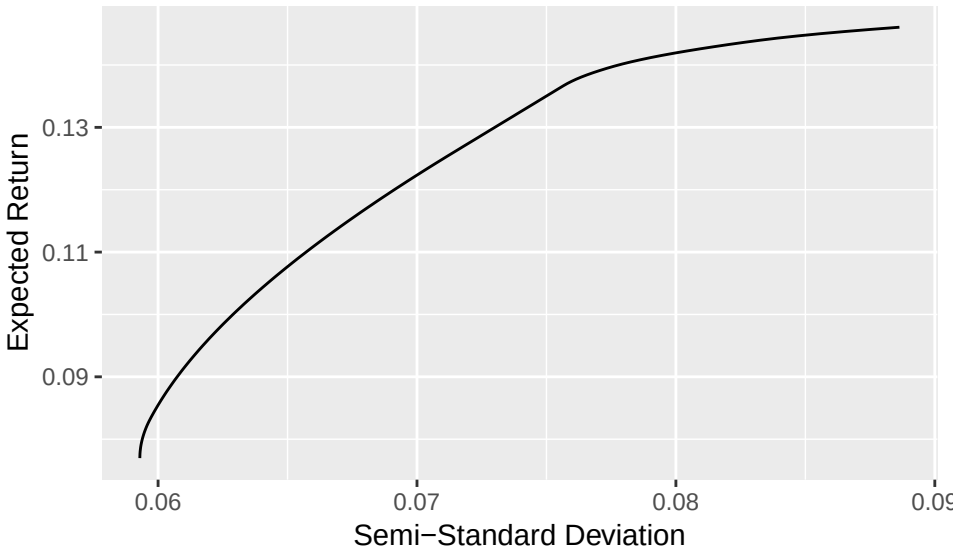

Figure 6: Mean-Semivariance Efficient Frontier

5 The MTXY Method

While the CLA is the most computationally economical way to find the entire efficient frontier, there exist a number of shortcuts that allow one to find individual points on the efficient frontier without mapping it out completely. For standard mean-variance optimization, one can use quadratic programming to find such points for given expected return, or for given variance, or for given λ_{E}.

In mean-semivariance optimization, though, shortcuts are not so readily available because of the locally applicable nature of the semicovariance matrix. However, a simple reformulation described in Markowitz, Todd, Xu, and Yamane [1992, 1993] (the MTXY method) allows one to find individual efficient points if one uses historical samples to form an estimate of the semicovariance matrix.

For a given required expected portfolio return μ_{P}, the mean-semivariance problem is to find $\boldsymbol{x}$ so as to

$$\left.\begin{array}{lrcl} \text{minimize} & \left((\boldsymbol{y})^-\right)^\top \left((\boldsymbol{y})^-\right) & & \\ \text{subject to} & \boldsymbol{\mu}^\top \boldsymbol{x} & = & \mu_{\text{P}}, \\ & \boldsymbol{A}\boldsymbol{x} & = & \boldsymbol{b}, \\ & \boldsymbol{B}_{\varrho}\boldsymbol{x} - \boldsymbol{y} & = & \boldsymbol{0}, \\ & \boldsymbol{x} & \geq & \boldsymbol{0}. \end{array}\right\} \qquad \text{(SV1)}$$

This problem cannot be solved in its current form using quadratic programming because of the presence of the "negative part" operator.

Nevertheless, following Markowitz, Todd, Xu, and Yamane [1992, 1993], we can convert this into a standard form by defining new non-negative variables that separately represent the positive parts and absolute values of the negative parts of $\boldsymbol{y}$. That is, let $\boldsymbol{n} = |(\boldsymbol{y})^-|$ hold the absolute values of the negative parts of $\boldsymbol{y}$, and let $\boldsymbol{p} = (\boldsymbol{y})^+ = \boldsymbol{y} - (\boldsymbol{y})^-$ hold the positive parts. We therefore have that $\boldsymbol{y} = \boldsymbol{p} - \boldsymbol{n}$. Then, using Equation (29), we arrive at the constraint $\boldsymbol{B}_{\varrho}\boldsymbol{x} = \boldsymbol{p} - \boldsymbol{n}$. The objective now simplifies to $\boldsymbol{n}^\top \boldsymbol{n}$. Therefore, the mean-semivariance problem (SV1) is equivalent to finding $\boldsymbol{x}$, $\boldsymbol{p}$, and $\boldsymbol{n}$ so as to

$$\left.\begin{array}{lrcl} \text{minimize} & \boldsymbol{n}^\top \boldsymbol{n} & & \\ \text{subject to} & \boldsymbol{\mu}^\top \boldsymbol{x} & = & \mu_{\text{P}}, \\ & \boldsymbol{A}\boldsymbol{x} & = & \boldsymbol{b}, \\ & \boldsymbol{B}_{\varrho}\boldsymbol{x} - \boldsymbol{p} + \boldsymbol{n} & = & \boldsymbol{0}, \\ & \boldsymbol{x} & \geq & \boldsymbol{0}, \\ & \boldsymbol{p} & \geq & \boldsymbol{0}, \\ & \boldsymbol{n} & \geq & \boldsymbol{0}. \end{array}\right\} \qquad \text{(SV2)}$$

The major disadvantage of this simple approach is that it increases the number of variables from n for $\boldsymbol{x}$ to $n + 2T$ for $\boldsymbol{x}$, $\boldsymbol{p}$, and $\boldsymbol{n}$. Naturally, T can be large, and this increases the computational burden significantly. Despite this, for typical institutional portfolios, and for standard desktop computer hardware, the computational burden is not prohibitive. If minimal computational burden is absolutely essential, one can use the streamlined version described in Markowitz, Todd, Xu, and Yamane [1992, 1993]. Those papers describe how to reduce the dimension of the problem at the expense of slightly more tricky matrix manipulations.

The sample R function `downside.cvx` in Appendix C solves Problem (SV2) using the CVXR package of Fu, Narasimhan, and Boyd [2017], which demonstrates the ease of using disciplined convex optimization [Grant, 2004] in this case. The input parameters to the function are the required return $\mu_{\mathrm{P}} =$ `reqret`, the vector of expected security returns $\boldsymbol{\mu} =$ `mu`, and the matrix of centered, normalized returns $\boldsymbol{B} =$ `B`.

Block `--C01--` loads the required `CVXR` library, and block `--C02--` sets the dimensions of the problem. Block `--C03--` declares the vectors $\boldsymbol{x} =$ `x`, $\boldsymbol{p} =$ `p`, and $\boldsymbol{n} =$ `n` to be decision variables in the optimization problem. Block `--C04--` defines the objective `obj` of the optimization to be minimization of the sum of the squares of the negative parts of $\boldsymbol{y}$.

Block `--C05--` defines the constraints. `C1` through `C3` are the non-negativity constraints on the decision variables. `C4` constrains the portfolio return to be equal to required return, and `C5` is the budget constraint. `C6` implements the constraint $\boldsymbol{B}_{\boldsymbol{\ell}}\boldsymbol{x} = \boldsymbol{p} - \boldsymbol{n}$. Finally, `C1` through `C6` are gathered together in the constraint list `con`.

Block `--C06--` defines the problem `prb` in terms of the objective `obj` and constraints `con`, and block `--C07--` solves the problem, putting the result into list `sol`. Finally, block `--C08--` extracts the values of `x`, `p`, and `n` from `sol` and returns them to the calling function.

6 Conclusion

It is often agreed that, intuitively, mean-semivariance portfolio optimization is more sensible than mean-variance optimization, but it is also often assumed that mean-semivariance optimization is too difficult. Even though the mean-semivariance problem was solved more than sixty years ago, portfolio managers are more familiar and comfortable with mean-variance optimization. Mean-semivariance optimization has, to a large extent, been forgotten. The purpose of this chapter is to serve as a reminder of the simplicity and intuitive appeal of mean-semivariance optimization, and to revive interest in it.

To accomplish this end, we provide a tutorial review of the theory of the critical line algorithm (CLA) as applied to standard mean-variance portfolio optimization. Coupled to the theory, we provide sample Python code to illustrate the ease with which the CLA traces out the entire efficient frontier. The Python code contains extensive comments that links the code closely to the theory.

Following this, we show how one can reduce downside exposure using mean-semivariance optimization. In general, mean-semivariance optimization is difficult to perform. However, the problem can be simplified dramatically if one uses an estimate of the covariance matrix derived from historical asset returns. In this case, one works with a particular square root of the covariance matrix from which the portfolio's semivariance is easily computed. We provide an explanation of this theory, and follow with sample R code that is extensively cross-referenced to the theory.

Finally, we show how to use the historical square root of the covariance matrix to write the mean-semivariance problem in standard quadratic form, and we provide a simple R function to find the optimal mean-semivariance portfolio for any feasible expected portfolio return.

Appendix A: Sample Python Code for Mean-Variance CLA

```
import numpy as np

# --A01--
def zerorows(x, j):
    y = x.copy()
    for k in j:
        y[k] = 0
    return y

# --A02--
def zerocols(x, j):
    y = x.copy()
    for k in j:
        y[:,k] = 0
    return y

# --A03--
def bar(x, j):
    y = x.copy()
    for k in j:
        y[k] = 0
        y[k,k] = 1
    return y

# --A04-- Portfolio Initialization.
def initport(mu, lb, ub):
   x       = lb.copy()
   ii      = np.argsort(-mu, axis=0)
   amtleft = 1 - np.sum(x)
   ix      = 0
   ns      = mu.shape[0]
   while ((amtleft > 0) and (ix < ns)):
      i       = ii[ix]
      chg     = min(ub[i]-lb[i], amtleft)
      x[i]    = x[i] + chg
      amtleft = amtleft - chg
      ix     += 1
   return(x)

# --A05-- Read the raw data.
ret = np.genfromtxt('example.tsv', delimiter="\t")
ret = np.delete(ret, 0, 1)

# --A06-- Set initial parameters.
ns  = ret.shape[1]
lb  = 0.1 * np.ones([ns, 1])
ub  = 0.5 * np.ones([ns, 1])
A   = np.ones([1, ns])
b   = 1.0
m   = 1
```

```
tol = 1E-9

# --A07-- Compute basic statistics of the data.
mu = np.mean(ret, axis=0).reshape([ns, 1])
C  = np.cov(ret.T)

# --A08-- Initialize the portfolio.
x  = initport(mu, lb, ub)

# --A09-- Set the state vector.
up = 1 * (abs(x - ub) < tol)
dn = 1 * (abs(x - lb) < tol)
s  = np.subtract(up, dn)

# --A10-- Set the P matrix.
P = np.concatenate((C, A.T), axis=1)

# --A11-- Initialize storage for quantities
# to be computed in the main loop.
LAM = float('inf')
lam = float('inf')
X   = x.copy()
V   = np.matmul(x.T, np.matmul(C, x))
R   = np.matmul(mu.T, x)
S   = s.copy()

# --A12-- The main CLA loop, which steps
# from corner portfolio to corner portfolio.
while lam > 0:

   # --A13-- Create the UP, DN, and IN
   # sets from the current state vector.
   UP = s > +0.9
   DN = s < -0.9
   IN = np.invert(np.logical_or(UP, DN))

   # --A14-- Create the Abar, Cbar, and Mbar matrices.
   io     = np.where(np.logical_not(IN))[0]
   Abar   = zerocols(A, io)
   Cbar   = bar(C, io)
   toprow = np.concatenate((Cbar, Abar.T), axis=1)
   botrow = np.concatenate((Abar, np.zeros([m, m])), axis=1)
   Mbar   = np.concatenate([toprow, botrow], axis=0)

   # --A15-- Create the right-hand sides for alpha and beta.
   up  = np.multiply(1 * UP, ub)
   dn  = np.multiply(1 * DN, lb)
   k   = np.add(up, dn)
   bot = b - np.matmul(A, k)
   top = zerorows(mu, io)
   rhsa = np.concatenate([k, bot], axis=0)
   rhsb = np.concatenate([top, np.zeros([m, m])], axis=0)

   # --A16-- Compute alpha, beta, gamma, and delta.
```

```
   alpha = np.linalg.solve(Mbar, rhsa)
   beta  = np.linalg.solve(Mbar, rhsb)
   gamma = np.matmul(P, alpha)
   delta = np.matmul(P, beta) - mu

   # --A17-- Prepare the ratio matrix.
   L = -float('inf') * np.ones([ns, 4])

   # --A18-- IN security possibly going UP.
   for i in np.where(IN & (beta[range(ns)] < -tol))[0]:
      L[i,0] = (ub[i] - alpha[i]) / beta[i]

   # --A19-- IN security possibly going DN.
   for i in np.where(IN & (beta[range(ns)] > +tol))[0]:
      L[i,1] = (lb[i] - alpha[i]) / beta[i]

   # --A20-- DN security possibly going IN.
   for i in np.where(UP & (delta < -tol))[0]:
      L[i,2] = -gamma[i] / delta[i]

   # --A21-- UP security possibly going IN.
   for i in np.where(DN & (delta > +tol))[0]:
      L[i,3] = -gamma[i] / delta[i]

   # --A22--If all elements of ratio are negative,
   # we have reached the end of the efficient frontier.
   if np.max(L) < 0:
      lam = -float('inf')
      break

   # --A23-- Find which security is changing state.
   secmax = np.max(L, axis=1)
   secchg = np.argmax(secmax)

   # --A24-- Find in which direction it is changing.
   dirmax = np.max(L, axis=0)
   dirchg = np.argmax(dirmax)

   # --A25-- Set the new value of lambda_E.
   lam = np.max(secmax)

   # --A26-- Set the state vector for the next segment.
   s[secchg] = (+1 if dirchg == 0 else
                -1 if dirchg == 1 else
                 0
   )

   # --A27-- Compute the portfolio at this corner.
   x = alpha[range(ns)]+ lam * beta[range(ns)]
   v = np.matmul(x.T, np.matmul(C, x))
   r = np.matmul(mu.T, x)

   # --A28-- Save the data computed at this corner.
   X = np.concatenate([X, x], axis=1)
```

```
159     S = np.concatenate([S, s], axis=1)
160     V = np.append(V, v)
161     R = np.append(R, r)
162     LAM = np.append(LAM, lam)
```

Appendix B: Sample R Code for Mean-Semivariance CLA

```
#############################################################
##                    Utility functions.                   ##
#############################################################

# --B01-- Function to set rows j of matrix X to zero.
zerorows <- function(X, j) {
   Y = X; for (k in j) Y[k,] <- 0; return(Y)
}

# --B02-- Function to set columns j of matrix X to zero.
zerocols <- function(X, j) {
   Y <- X; for (k in j) Y[,k] <- 0; return(Y)
}

# --B03-- Function to set rows j of matrix X to zero,
# and diagonal elements j with 1.
overbar <- function(X, j) {
   Y <- zerorows(X, j);
   for (k in j) Y[k,k] <- 1; return(Y)
}

# --B04-- Function to initialize a long-only portfolio
# subject only to the budget constraint.
initport <- function(mu, lb, ub) {
   xinf  <- lb; amtleft <- 1 - sum(xinf)
   ii    <- order(-mu); ns <- dim(mu)[1]
   ix    <- 1
   while ((amtleft > 0) & (ix <= ns)) {
      i       <- ii[ix]
      chg     <- min(ub[i]-lb[i], amtleft)
      xinf[i] <- xinf[i] + chg
      amtleft <- amtleft - chg
      ix      <- ix + 1
   }
   return(xinf)
}

#############################################################
##   Critical Line Algorithm for Downside Optimization.    ##
#############################################################
dscla <- function(R, A, b, lb, ub, refret=0, tol=1E-9) {

   # --B05-- Basic properties of returns data.
   no   <- dim(R)[1]
   ns   <- dim(R)[2]
   mu   <- matrix(colMeans(R), ncol=1)

   # --B06-- Initialize portfolio with Sharpe's method.
   x  <- initport(mu, lb, ub)
   x  <- matrix(x, ncol=1)
```

```
51
52    # --B07-- Set the initial state vector.
53    state <- matrix((x == ub) - (x == lb), ncol=1)
54
55    # --B08-- Find which securities should be included
56    # in the initial locally applicable semicovariance matrix.
57    Rex <- R - matrix(refret, no, ns, byrow=T)
58    y   <- Rex %*% x
59    obs.in <- y < -tol
60
61    # --B09-- Compute the locally applicable
62    # semicovariance matrix, Sloc, and set up
63    # the locally applicable P and M matrices.
64    Rloc <- Rex[obs.in,]
65    Sloc <- t(Rloc) %*% Rloc / no
66    Ploc <- cbind(Sloc, t(A))
67    Mloc <- rbind(Ploc, cbind(A, 0))
68
69    # --B10--  Initialize storage for quantities
70    # to be computed in the main loop.
71    portfolios <- x
72    semivars   <- as.numeric(t(x) %*% Sloc %*% x)
73    returns    <- sum(x * mu)
74    lambdas    <- lam <- Inf
75
76    # --B11-- Enter the main CLA loop.
77    while (lam > 0) {
78       lam.prv <- lam
79
80       # --B12-- Set the logical state indicators from the
81       # state vector, and find securities that are OUT.
82       UP <- state > +0.9; DN <- state < -0.9
83       IN <- !(UP | DN);   OT <- which(!IN)
84
85       # --B13-- Create the Abar, Sbar, and Mbar matrices.
86       Sbar <- overbar(Sloc, OT); Abar <- zerocols(A, OT)
87       Mbar <- rbind(cbind(Sbar, t(Abar)), cbind(Abar, 0))
88
89       # --B14-- Create the right-hand sides for alpha and beta.
90       k <- ub * UP + lb * DN
91       rhsa <- rbind(matrix(0, ns, 1), b-A%*%k)
92       rhsb <- rbind(zerorows(mu, OT), 0)
93
94       # --B15-- Compute alpha, beta, gamma, and delta.
95       alf <- solve(Mbar, rhsa)
96       bet <- solve(Mbar, rhsb)
97       gam <- Ploc %*% alf
98       del <- Ploc %*% bet - mu
99
100      # --B16-- Select alpha and beta for real securities
101      # only (i.e., exclude Lagrange multipliers).
102      alf.sec <- matrix(alf[1:ns], ncol=1)
103      bet.sec <- matrix(bet[1:ns], ncol=1)
104
```

```
      # --B17-- Check for IN security possibly going UP.
      LIU <- matrix(-Inf, ns, 1)
      i   <- which(IN & (bet.sec < -tol))
      if (length(i) > 0) LIU[i] <- (ub[i] - alf.sec[i]) / bet.sec[i]

      # --B18-- Check for IN security possibly going DN.
      LID <- matrix(-Inf, ns, 1)
      i   <- which(IN & (bet.sec > +tol))
      if (length(i) > 0) LID[i] <- (lb[i] - alf.sec[i]) / bet.sec[i]

      # --B19-- Check for UP security possibly going IN.
      LUI <- matrix(-Inf, ns, 1)
      i   <- which(UP & (del < -tol))
      if (length(i) > 0) LUI[i] <- -gam[i] / del[i]

      # --B20-- Check for DN security possibly going IN.
      LDI <- matrix(-Inf, ns, 1)
      i   <- which(DN & (del > +tol))
      if (length(i) > 0) LDI[i] <- -gam[i] / del[i]

      # --B21-- Form Lambda matrix for finding maximal lambda_E.
      L <- cbind(LIU, LID, LUI, LDI)

      # --B22-- Find which security might change state.
      secmax <- apply(L, 1, max, na.rm=T)
      secchg <- which.max(secmax)

      # --B23-- Find the direction in which the state might change.
      dirmax <- apply(L, 2, max, na.rm=T)
      dirchg <- which.max(dirmax)

      # --B24-- Set the value of lambda_E for a changing security.
      lam.sec  <- max(secmax)

      # --B25-- Find the lambda_E ratios for possible
      # constituent changes in the semicovariance calculation.
      num <- Rex %*% alf.sec
      den <- Rex %*% bet.sec
      rat <- matrix(-Inf, no, 1)
      i   <- which(abs(den) > +tol)
      if (length(i) > 0) rat[i] <- -num[i] / den[i]

      # --B26-- Find the maximum lambda_E for
      # observations changing in the LSM.
      lam.obs <- -Inf
      i <- which(rat - lam.prv < -tol)
      if(any(i)) lam.obs <- max(rat[i])

      # --B27-- Decide if the constituents of the LSM are
      # changing or if a security's state is changing.
      if (lam.obs > lam.sec) { # crossing a profitability line.

         # --B28-- Set lambda_E.
         lam <- lam.obs
```

```
159
160         # --B29-- Compute the portfolio at profitability line.
161         x <- alf.sec + lam * bet.sec
162
163         # --B30-- Find which securities should be
164         # included in the next LSM, and compute it.
165         obs.in <- Rex %*% x < -tol
166         Rloc   <- Rex[obs.in,]
167         Sloc   <- t(Rloc) %*% Rloc / no
168
169         # --B31-- Form the local P and M matrices.
170         Ploc   <- cbind(Sloc, t(A))
171         Mloc   <- rbind(Ploc, cbind(A, 0))
172
173      } else { # a security's state is changing at a new corner.
174
175         # --B32-- Set lambda_E.
176         lam <- lam.sec
177
178         # --B33-- Compute the portfolio at this corner.
179         x <- alf.sec + lam * bet.sec
180
181         # --B34-- Set the state vector for the next segment.
182         if (dirchg == 1) {        # IN security going UP.
183            state[secchg] <- +1
184         } else if (dirchg == 2) { # IN security going DN.
185            state[secchg] <- -1
186         } else {                  # Security coming IN.
187            state[secchg] <- 0
188         }
189      }
190
191      # --B35-- Save the data computed at this value of lambda_E.
192      portfolios <- cbind(portfolios, x)
193      semivars   <- c(semivars, as.numeric(t(x) %*% Sloc %*% x))
194      returns    <- c(returns, sum(x * mu))
195      lambdas    <- c(lambdas, lam)
196   }
197
198   # --B36-- Return a list representing the efficient frontier.
199   ef <- list(portfolios = portfolios,
200              semivars   = semivars,
201              returns    = returns,
202              lambdas    = lambdas)
203   return(ef)
204 }
205
206 ##############################################################
207 ##                      Main Program.                       ##
208 ##############################################################
209
210 # --B37-- Read the raw returns data.
211 R <- as.matrix(read.table("example.tsv",
212                           row.names = 1,
```

```
                              as.is      = T))

# --B38-- Set constraints.
ns <- dim(R)[2]
A  <- matrix(1, 1, ns)   # Budget constraint matrix.
b  <- 1                  # Budget constraint constant.
lb <- matrix(0,  ns, 1)  # Long-only
ub <- matrix(10, ns, 1)  # No upper bound in practice.

# --B39-- Call the Downside CLA function.
ef <- dscla(R, A, b, lb, ub, refret=0, tol=1E-9)
```

Appendix C: MTXY Method Sample CVXR Code

```
################################################################
## Downside portfolio optimization with CVXR
downside.cvx <- function(reqret, mu, B) {

   # --C01-- Load the CVXR package.
   suppressMessages(library(CVXR))

   # --C02-- Find the dimensions of the problem.
   NT  <- dim(B)[1]
   NS  <- dim(B)[2]

   # --C03-- Declare the variables.
   x   <- Variable(NS)
   p   <- Variable(NT)
   n   <- Variable(NT)

   # --C04-- Define the objective.
   obj <- Minimize(sum(n^2))

   # --C05-- Define the constraints.
   C1  <- x > 0
   C2  <- p > 0
   C3  <- n > 0
   C4  <- sum(x * mu) == reqret
   C5  <- sum(x) == 1
   C6  <- B %*% x - p + n == 0
   con <- list(C1, C2, C3, C4, C5, C6)

   # --C06-- Set up the problem in terms of the
   # objective and constraints.
   prb <- Problem(obj, con)

   # --C07-- Solve the problem.
   sol <- solve(prb)

   # --C08-- Create and return a list of results.
   x.o <- sol$getValue(x)
   p.o <- sol$getValue(p)
   n.o <- sol$getValue(n)
   res <- list(x=x.o, p=p.o, n=n.o)
   return(res)
}
```

References

David H. Bailey and Marcos López de Prado. An Open-Source Implementation of the Critical-Line Algorithm for Portfolio Optimization. *Algorithms*, 6(1):169–196, 2013.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Denisa Cumova and David Nawrocki. A Symmetric LPM Model for Heuristic Mean-Semivariance Analysis. *Journal of Economics and Business*, 63(3):217–236, 2011.

Denisa Cumova and David Nawrocki. Portfolio Optimization in an Upside Potential and Downside Risk Framework. *Journal of Economics and Business*, 71:68–89, 2014.

Javier Estrada. Mean-Semivariance Optimization: A Heuristic Approach. *Journal of Applied Finance*, Spring/Summer:57–72, 2008.

Anqi Fu, Balasubramanian Narasimhan, and Stephen Boyd. CVXR: An R Package for Disciplined Convex Optimization. Technical report, Stanford University, 2017. URL https://web.stanford.edu/~boyd/papers/cvxr_paper.html.

Michael C. Grant. *Disciplined Convex Programming*. PhD thesis, Stanford University, 2004.

William W. Hager. Updating the Inverse of a Matrix. *SIAM Review*, 31(2):221–239, 1989.

Bruce I. Jacobs, Kenneth N. Levy, and David Starer. Practical Optimization of Enhanced Active Equity Portfolios, Chapter 3 in Bernd Scherer and Kenneth Winston (Eds), *The Oxford Handbook of Quantitative Asset Management*, pages 32–49. Oxford University Press, Oxford, UK, 2012.

Haim Levy and Harry M. Markowitz. Approximating Expected Utility by a Function of Mean and Variance. *American Economic Review*, 69(3):308–317, June 1979.

Harry Markowitz, Peter Todd, Ganlin Xu, and Yuji Yamane. Computation of Mean-Semivariance Efficient Sets by the Critical Line Algorithm. *Annals of Operations Research*, 45(1):307–317, 1993.

Harry M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Basil Blackwell, Cambridge, MA, 1959.

Harry M. Markowitz. *Mean-Variance Analysis in Portfolio Choice and Capital Markets*. Basil Blackwell, Cambridge, MA, 1987.

Harry M. Markowitz and G. Peter Todd. *Mean-Variance Analysis in Portfolio Choice and Capital Markets*. Frank J. Fabozzi, New Hope, PA, 2000.

Harry M. Markowitz, Peter Todd, Ganlin Xu, and Yuji Yamane. Fast Computation of Mean-Variance Efficient Sets using Historical Covariances. *Journal of Financial Engineering*, 1 (2):117–132, 1992.

Andras Niedermayer and Daniel Niedermayer. Applying Markowitz's Critical Line Algorithm. In John B. Guerard, editor, *Handbook of Portfolio Construction*, pages 383–400. Springer, 2009.