Leiden University

Mathematics

Master Thesis

Analysis of Near-Optimal Portfolio Regions and Polytope Theory

Author:
W.J.B. van Eeghen

Supervisors:
Dr. O.W. van Gaans
(M.I. Leiden)

Dr. M. van der Schans
(Ortec Finance)

September 6, 2018

Universiteit Leiden

Abstract

Many investors use optimization as a tool to make investment decisions. An investor decides which proportion of his wealth to invest in which asset class, thus composing his investment portfolio. Given all asset classes available to the investor, with optimization the investor tries to find the portfolio with the most favourable risk and return trade-off. Historical data and models are used to estimate asset classes' risks and returns. A problem with this approach is that optimal portfolios are often sensitive to variation in the uncertain risk- and return estimates, as shown in [18] and [23].

An alternative is to compute not only the optimal portfolio, but also the portfolios that are near-optimal, see [16]. This results in a near-optimal region of portfolios that can be shown to be more robust. The method to compute near-optimal regions is however computationally intensive. In this thesis, we reduce the computation time to determine a near-optimal region, without losing significant accuracy.

When a near-optimal region is known, an investor still needs to decide which (near-optimal) portfolio to invest in. Near-optimal region estimates can be objects of high affine dimension, which are difficult to grasp and navigate for practitioners. In this thesis, we use polytope theory and show how this can be used to study the near-optimal region in more detail.

Acknowledgements

This thesis was written to get my master's title for my studies mathematics at Leiden University. My graduation project was conducted during an internship at Ortec Finance, where I have worked for a period of over seven months. I would like to thank both Ortec Finance, and Leiden University, for making this challenging project possible.

My special thanks go to my supervisors Martin van der Schans and Onno van Gaans. Martin guided me throughout my internship and not only helped me to succeed in an interesting research with practical value, but also put great focus making sure the research was sufficiently scientific. Onno many times reviewed my writing attempts helped me patiently, thoroughly, and enthusiastically, with many mathematical proofs. I would like to thank my colleagues at Ortec Finance, who showed their interest in this project and with whom I have enjoyed working together. I also want to thank my father, who critically read this thesis and gave lots of valuable remarks and who, together with my mother, supported me throughout my studies.

Contents

Introduction

Many investors use optimization as a tool to make their investment decisions. In portfolio management, investors can generally choose from a wide variety of different asset classes to invest in. These asset classes have different characteristics. Investors select their investment portfolio by choosing which proportion of their wealth to invest which asset class. Using models and historical data, investors assess asset classes to estimate their expected risks and returns. Then, the investor makes a trade-off between the risk involved with investing in these asset classes and the return that these asset classes yield. This results in a portfolio optimization problem, from which an optimal portfolio is deduced. A problem with this approach is that the optimal portfolio's are often sensitive to small changes in the risk and return estimations, as is shown in [18] and [23]. Therefore, there is a need for robustness in portfolio optimization.

Several approaches to robustness in portfolio optimization are available in the literature. One approach is to reduce sensitivity by reducing variation in the input of historical data. This may be attempted in different ways, including shrinkage [24], robust statistics [34], Black-Litterman inverse optimization [7], and Bayesian optimization [38]. Another approach changes the optimization objective to reduce its sensitivity to varying input. DeMiguel [28] and Brody [8] attempt this through regularization. There are also methods that combine these two approaches. For instance, Michaud [29] resamples the input and averages over the optimization results to construct a resampled frontier.

In 2017, De Graaf and Van der Schans presented the approach of near-optimal portfolio optimization in [15] and [16]. They propose to change the model's objective into finding a region of near-optimal portfolios instead of a single optimal portfolio and present a method to do so. Their method succeeds in constructing a region of portfolios that is more robust than the traditional optimization outcome. This method is, however, computationally very expensive.

When a region of near-optimal portfolios is known, the question arises which (near-optimal) portfolio should be selected by the investor. The near-optimal regions are often complex objects that are not easy to navigate. To effectively use the near-optimal approach in practice, we need to get a better grip on a near-optimal region's structure.

In this thesis we perform further research on the near-optimal region as defined by De Graaf and Van der Schans. We give a description of their work in Section 1.1. We analyze the near-optimal region's structure using polytope theory in Chapter 2, where we formulate an algorithm to triangulate a near-optimal region in Section 2.3. We investigate the near-optimal region's robustness and present two ways to select a specific portfolio out of the near-optimal region in Chapter 3. We improve computation time of the method to construct a near-optimal region by simplifying the existing implementation and by presenting a new method in Chapter 4.

This research was performed in close collaboration with Ortec Finance. Part of Ortec Finance's activities involves advising on investment decisions. The development of the near-optimal approach in [15] and [16] was commissioned by the company. Now that it has been shown that the method is viable, Ortec Finance is interested in whether it can be improved to make it more applicable in practice. At the end of this thesis, in Section 5, we will make a recommendation to Ortec Finance on how to proceed with the near-optimal portfolio method.

1 Near-Optimal Portfolio Optimization

Portfolio optimization is often based on estimations of risk and return. A widely used method to optimize risk and return is *Mean-Variance Optimization*, based on the influential paper [26] by Markowitz. Suppose an investor can compose his investment portfolio of $d \in \mathbb{N}$ different asset classes. We denote an investment portfolio by $w \in \mathbb{R}^d$, where w_i denotes the proportion of the investor's wealth that is invested in asset class i. We hence require $\sum_{i=1}^{d} w_i = 1$ and $w_i \geq 0$ for all $i \in \{1, \ldots, d\}$. In Mean-Variance Optimization, a portfolio is estimated to have an expected return based on its assets' mean returns $\vec{\mu} \in \mathbb{R}^d$, and to have and expected risk, based on its assets' estimated covariance matrix $\Sigma \in [-1, 1]^{d \times d}$, as

$$\begin{aligned} \text{Risk}(w) &= w^\intercal \Sigma w \\ \text{return}(w) &= w^\intercal \vec{\mu}. \end{aligned} \tag{1}$$

There is always a trade-off between minimizing risk and maximizing return. An *efficient frontier* optimizing this trade-off is defined in in Definition 1.1.

Definition 1.1 (Mean-Variance Efficient Frontier)**.** Suppose an investor can invest in $d \in \mathbb{N}$ different asset classes with mean returns $\vec{\mu} \in [0, \infty)^d$ and covariance matrix $\Sigma \in [-1, 1]^{d \times d}$. Let X denote the set of possible portfolios

$$X := \{w \in \mathbb{R}^d : \sum_{i=1}^{d} w_i = 1, w_i \geq 0 \text{ for all } i \in [d]\}.^1$$

The *Mean-Variance Efficient Frontier* is defined as

$$\{(\mu, \sigma^2) \in \mathbb{R}^2 : \mu = w^\intercal \vec{\mu}, \sigma^2 = \min_{w \in X}\{w^\intercal \Sigma w\}, w \in X\}.$$

Alternatively, the mean-variance efficient frontier can be constructed by minimizing

$$w^\intercal \Sigma w - q \cdot w^\intercal \vec{\mu},$$

for $q \in [0, \infty)$. In the latter expression, q can be considered as a parameter for an investor's risk-appetite. Note that the efficient frontier is a curve in risk-return space. An example efficient frontier is displayed in Figure 1. The portfolios whose estimated risk and return lie on the efficient frontier are called *efficient portfolios*.

Definition 1.2 (Mean-Variance Efficient Portfolios)**.** Suppose an investor can invest in $d \in \mathbb{N}$ different asset classes with mean returns $\mu \in [0, \infty)^d$ and covariance matrix $\Sigma \in [-1, 1]^{d \times d}$. Let X denote the set of possible portfolios

$$X := \{w \in \mathbb{R}^d : \sum_{i=1}^{d} w_i = 1, w_i \geq 0 \text{ for all } i \in [d]\}.$$

The set of *Mean-Variance Efficient Portfolios* is defined as

$$\{w \in X : (w^\intercal \vec{\mu}, w^\intercal \Sigma w) \text{ is an element of the Mean-Variance Efficient Frontier}\}.$$

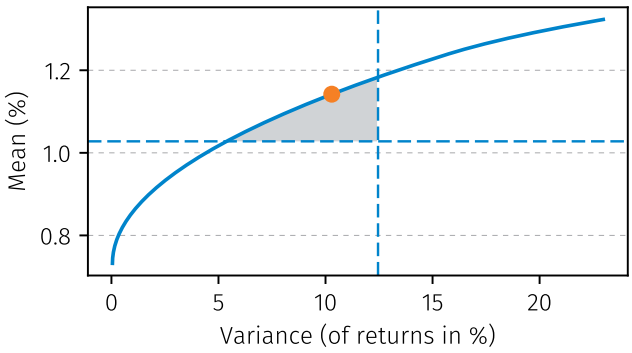
[1] We use the notation $[d] \equiv \{1, \ldots, d\}$.

The traditional way investors use Mean-Variance optimization to select their investment portfolio is by estimation of the efficient frontier and selection of a single efficient portfolio depending on the investor's risk appetite. The problem with this approach is that the estimated set of efficient portfolios is usually sensitive to small changes in the input parameters. This may harm robustness of the optimization outcome. Chopra showed in [9] that portfolios with risk- and return very close to the estimated optimal portfolio can differ greatly in their allocations, as we show in an example in Section 1.2. A lack of robustness in the optimization may thus have great influence on the investment decision proposed by the model. De Graaf and Van der Schans develop a solution to the robustness problem in [15], and [16]. We start by giving an overview of their work.

1.1 Method

In [15] and [16], De Graaf and Van der Schans changed the optimization objective in order to gain more robust results. Instead of looking for a single optimal efficient portfolio, the goal was defined as finding all 'near-optimal' portfolios whose risk and return lie just below the efficient frontier within certain boundaries. Near-optimality can thus be thought of as closeness to the single "optimal" portfolio in terms of estimated risk and return. The principle is illustrated by Figure 1 from [16].

Figure 1: An illustration of mean-variance optimization by De Graaf and Van der Schans in [16]. The near-optimal region in risk-return space is represented by the shaded area. It is defined by the efficient frontier (solid blue line) and the risk- and return-cutoffs (dotted blue lines). The orange dot represents the position of the mean-variance optimal portfolio that is the outcome of traditional optimization. Note that the position of the mean-variance optimal portfolio on the efficient frontier depends on the investor's risk appetite.



De Graaf and Van der Schans argue that the volatility of the available models' estimations justifies the consideration of near-optimal portfolios. They define their near-optimal region based on the traditional Mean-Variance optimization model. They present a method to estimate a near-optimal region. They next show that the near-optimal region estimate, based on an example from Chopra in [9] is significantly more robust than a single optimization outcome.

Next to robustness, an additional advantage of a near-optimal region as optimization result is that the investor can bring extra arguments into selection of his final portfolio. Usually, not all relevant arguments are taken into account in optimization; some arguments are difficult to quantify or are not considered for different reasons. When an overview of near-optimal portfolios is presented as the model's outcome to an investor, he or she may use these additional arguments and his or her expert opinion to decide

which portfolio to invest in. Note that a similar approach is already common practice; optimization models are often used to support decision making, rather than replace it.

We now introduce the setting from [16] in which the near-optimal region is defined and constructed. Suppose an investor manages a portfolio consisting of $d \in \mathbb{N}$ possible asset classes. In traditional optimization models, the investor estimates the performance of a portfolio $w \in \mathbb{R}^d$ using its estimated risk and return. Common measures for a portfolio's risk and return are its variance and mean return respectively, given by (1). The traditional optimization constructs the efficient frontier as in Definition 1.1 and selects a single optimal portfolio w^*, depending on the investors risk appetite, as outcome. De Graaf and Van der Schans however define a near-optimal region OPT as the set of possible portfolios that have expected risk and return within user-defined margins $\delta_\Sigma, \delta_\mu$ of the risk and return of w^* respectively. That is, for portfolio w in the near-optimal region OPT, they require

$$\begin{aligned} w^\intercal \Sigma w &\leq w^{*\intercal} \Sigma w^* + \delta_\Sigma \\ w^\intercal \mu &\geq w^{*\intercal} \mu - \delta_\mu. \end{aligned} \tag{2}$$

The constraints in (2) enforce near-optimality of the portfolios. The set of "possible portfolios" is defined by a system of linear equality constraints and a system of linear inequality constraints,

$$\begin{aligned} Aw &= b \\ Gw &\leq h, \end{aligned} \tag{3}$$

among which are the constraints that all portfolio weights should be non-negative and that together they should sum to one. These constraints may contain further optimization input specified by the investor. For instance, an investor may want to constrain his portfolio's allocation to real estate to a maximum of ten percent because the asset class's poor liquidity. De Graaf and Van der Schans now come to the definition of the near-optimal region OPT, combining the constraints in (2) and (3) in Definition 1.3.

Definition 1.3. Consider a portfolio space of $d \in \mathbb{N}$ asset classes with mean returns $\mu \in \mathbb{R}^d$ and estimated covariance matrix $\Sigma \in [-1, 1]^{d \times d}$. A near-optimal region OPT with respect to Mean-Variance optimization is defined by

$$\begin{aligned} OPT(w^*, \delta_\Sigma, \delta_\mu) := \Big\{ w \in \mathbb{R}^d : w^\intercal \Sigma w \leq w^{*\intercal} \Sigma w^* + \delta_\Sigma, w^\intercal \mu \geq w^{*\intercal} \mu, \\ Aw = b, Gw \leq h \Big\}, \end{aligned}$$

where $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$ $(m < d)$, $G \in \mathbb{R}^{k \times d}$, $h \in \mathbb{R}^k$, $\delta_\Sigma, \delta_\mu > 0$, and $w^* \in \mathbb{R}^d$ is the estimated Mean-Variance-optimal portfolio.

De Graaf and Van der Schans proceed to estimate OPT. They first show that the near-optimal region is convex. They then estimate the convex near-optimal region iteratively starting with $w_0 = w^*$. In iteration i, point w_i is defined to be the point in OPT that lies furthest away from convex hull $P_{i-1} = \text{Conv}(w_0, \ldots, w_{i-1})$. The procedure continues until the Euclidean distance $d(w_{i+1}, P_i)$ is smaller than some predefined $\epsilon > 0$.

De Graaf and Van der Schans show in [16] that their technique works and produces results that are more robust than the results from traditional portfolio optimization. The computation time of their algorithm ranges from several minutes to several hours, depending on the size of the input. In the common case of portfolios consisting of over 20 asset classes the computation time starts at around two hours and up.

We note that there are other ways to define a portfolio's risk and return than specified in the formula's in (1). Different ways of estimating risk and return may lead to different efficient frontier. We extend Definition 1.3 of De Graaf and Van der Schans to general risk and return measures in Definition 1.4.

Definition 1.4. Consider a portfolio space of $d \in \mathbb{N}$ asset classes with mean returns $\mu \in \mathbb{R}^d$ and estimated covariance matrix $\Sigma \in [-1,1]^{d \times d}$. A near-optimal region OPT with respect to general risk- and return constraints as

$$OPT := \{w \in \mathbb{R}^d : Aw = b, Gw \leq h, f_\mu(w) \leq C_\mu, f_\Sigma(w) \leq C_\Sigma\},$$

with $A \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{k \times d}$, $h \in \mathbb{R}^k$, and with $C_\mu, C_\Sigma \in \mathbb{R}$, and $f_\mu, f_\Sigma : \mathbb{R}^d \to \mathbb{R}$ convex.

In Definition 1.4 f_Σ and f_μ denote risk and return measures respectively. We require their convexity to ensure that OPT is a convex region. We will discuss how to construct OPT, including a further analysis of the method by De Graaf and Van der Schans, in Chapter 4.

We observe that De Graaf and Van der Schans use their method to estimate a convex region in $\mathbb{R}^d$ by a convex hull $\text{Conv}(w_0, \ldots, w_n)$. In the setting of geometry and discrete optimization, the convex hull of a finite set of points $w_0, \ldots, w_n \in \mathbb{R}^d$ may also be called a *convex polytope*. In this research, we will look at the near-optimal portfolio region OPT defined by De Graaf and Van der Schans as a convex polytope in a portfolio space in $\mathbb{R}^d$. We will investigate the structure of convex polytopes using a technique called *triangulation* in Chapter 2. First, we present an example of a near-optimal region in Section 1.2.

1.2 Example

We present an example from De Graaf and Van der Schans in [16]. The example defines a simple portfolio space of three assets: Stocks, Bonds, and Treasury Bills. The statistics of these assets are presented in Table 1 and originate from [9] by Chopra.

	Stocks	Bonds	T-Bills
Mean	1.323%	1.027%	0.729%
Stdev.	4.793%	3.984%	0.219%
Correlations:			
Stocks	1.000		
Bonds	0.341	1.000	
T-Bills	-0.081	0.050	1.000
Optimal alloc.	58.1%	22.8%	19.1%

Table 1: Expected risk and return estimations

Table 1 shows statistics of monthly returns from January 1980 to December 1990 as reported by [9]. The mean-variance efficient frontier in Figure 1 is based on the statistics in Table 1. The optimal allocation consists portfolio weights of the orange dot in Figure 1.

De Graaf and Van der Schans used their convex hull computation method to construct a near-optimal region for this example. They also constructed the centroid portfolio, defined in Definition 1.5. Their results are displayed in Figure 2 and Table 2.

Definition 1.5. Let $R = \text{Conv}(w_0, \ldots, w_n)$ with $w_0, \ldots, w_n \in \mathbb{R}^d$ vertices of R. The *centroid of* R is defined as

$$\frac{1}{n+1} \sum_{i=0}^{n} w_i.$$

Consequently, if R is a region of portfolios, the centroid may also be called the *centroid portfolio*.

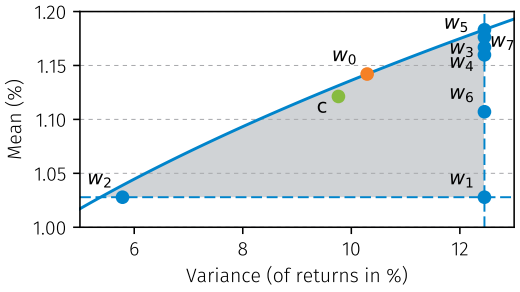


Figure 2: Mean-variance efficient frontier with a near-optimal region.

	Allocation			Mean	Std.
	Stocks	Bonds	T-Bills		
w_0	58.1%	22.8%	19.1%	1.14%	3.21%
w_1	7.7%	85.0%	7.3%	1.03%	3.53%
w_2	50.3%	0.0%	49.7%	1.03%	2.41%
w_3	73.7%	0.0%	26.3%	1.17%	3.53%
w_4	46.1%	52.7%	1.2%	1.16%	3.53%
w_5	62.6%	27.6%	9.8%	1.18%	3.53%
w_6	27.8%	71.4%	0.8%	1.11%	3.53%
w_7	55.0%	40.6%	4.5%	1.18%	3.53%
c	46.2%	39.6%	14.2%	1.12%	3.12%

Table 2: Near optimal portfolios and their centroid.

Figure 2 shows, as in Figure 1, the mean-variance efficient frontier and mean-variance optimal portfolio (orange dot) based on the statistics in Table 1. In addition, the figure shows a near-optimal region (shaded) covered by the near-optimal portfolios (blue dots). The centroid (green dot) is the average of the blue dots. Table 2 lists the allocation weights and statistics of the optimal portfolio w_0, the near-optimal portfolios and centroid portfolio c.

When looking at Table 2, we can clearly see an example of the impact the mean-variance optimization's robustness problem might have. Several portfolios in Table 2 have very similar expected risk and return values, while their allocation can differ greatly. To see how much allocations of portfolios that are close in terms of risk and return may differ in their asset allocations, look for instance at portfolios w_3 and w_4 in Table 2.

Throughout this thesis we will use the portfolio space given by Table 1 to generate test results. We note that a portfolio space consisting of these three assets is highly simplified and is unlikely to appear in portfolio optimization in practice. To be able to compare results for a more realistic portfolio space, we also use an example consisting of 133 asset classes in Chapter 4. We do not show the underlying statistics of this example in this thesis.

2 Triangulation

We want to analyze the structure of near-optimal regions. Near-optimal regions can be estimated by *convex polytopes*. In order to analyze near-optimal regions in detail, we can use their polytope estimations. We introduce polytope theory in this chapter and connect it to near-optimal portfolio theory to get more grip on the structure of near-optimal regions. We will use the theory and algorithms presented in this chapter in later chapters to extend knowledge and applications of near-optimal portfolio regions.

We give an introduction to polytope theory in Section 2.1, summarizing theory from the literature. We focus on an important tool in polytope theory, called *triangulation* in Section 2.2. In Section 2.3, we reformulate an algorithm from the literature to triangulate full-dimensional convex polytopes in $\mathbb{R}^d$ and extend the literature by providing its detailed proof. We furthermore extend the application of the algorithm to general convex polytopes in $\mathbb{R}^d$ in Section 2.4. In Section 2.5 we analyze the algorithm performance and test it in the setting of portfolio optimization.

2.1 Polytope theory

We give an introduction to polytope theory. All results in this section except for Definitions 2.1, 2.2, and Proposition 2.4 are from Schäfer in [37, Section 8.3]. Definition 2.2 is due to Lee and Santos in [14, Chapter 16].

Polytopes have been studied since classical antiquity. For a long time, polytope theory has been limited to two-dimensional polytopes, also called *polygons*, and three-dimensional polytopes, called *polyhedra*. Mathematicians, starting with August Ferdinand Möbius, are known to have started showing interest in polytopes in higher dimensions in the 19th century. In 1967 Grünbaum [17] related convex polytope theory to a more abstract setting of combinatorial properties and discrete mathematics. Since then, polytope theory has found many applications in different areas such as optimization, linear programming, computer graphics, search engines and many more. In particular, polytope theory can be applied in the setting of portfolio optimization.

Unfortunately, the literature does not always agree on the exact definition of polytopes and their features. We limit ourselves to convex polytopes in $\mathbb{R}^d$ for $d \in \mathbb{N}$. This allows us to speak of dimensions of polytopes, defined by their affine span. We introduce the concepts of affine spaces and spans before starting on the basics of polytope theory.

Definition 2.1 (Affine Subspace)**.** We say that A is an *affine subspace* over $\mathbb{R}^d$ if $A \subseteq \mathbb{R}^d$ and there is $v \in \mathbb{R}^d$ such that $B = \{a - v : a \in A\}$ is a linear subspace. The dimension of A is defined as equal to the dimension of B.

Definition 2.2 (Affine Span)**.** The *affine span* of a set $V \subset \mathbb{R}^d$ is the smallest affine subspace containing V. It is denoted $\text{aff}(V)$.

In Section 1.1, we introduced the concept of (convex) polytopes as the convex hull of a finite set of points in $\mathbb{R}^d$. This is usually called the $\mathcal{V}$-representation.

$$P = \text{Conv}(X), \text{ where } X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d. \tag{4}$$

Alternatively, a polytope can be represented as the bounded set of solutions to a finite system of linear inequalities (or as the intersection of a finite number of halfspaces). This is usually called a polytope's $\mathcal{H}$-representation.

$$P = \{x \in \mathbb{R}^d | Gx \leq h\}, \text{ where } G \in \mathbb{R}^{m \times d}, h \in \mathbb{R}^m \text{ and } P \text{ bounded and non-empty.} \tag{5}$$

Note that the boundedness of P ensures that the solution set to the system of linear inequalities can be represented by a finite number of vertices. Unbounded linear systems fall in the more general class of *polyhedra* [2], which we will not elaborate on any further.

The affine dimension $\dim(P)$ of a polytope $P = \mathrm{Conv}(X)$ is defined by the dimension of the affine span $\mathrm{aff}(X)$ of X or, equivalently, the affine span $\mathrm{aff}(P)$ of P. For now, we assume that any polytope $P \subset \mathbb{R}^d$ is full-dimensional, that is $\dim(P) = d$. In Section 2.4, we show how to deal with polytopes that are not full-dimensional ($\dim(P) < d$). To avoid any confusion on the dimension of a polytope, we introduce the notation *d-polytope* for a polytope with an affine span of dimension d.

Theorem 2.3 (Main Theorem of Polytope Theory)
The $\mathcal{V}$-representation in (4) and the $\mathcal{H}$-representation in (5) of polytopes are equivalent.

For a proof, we refer the reader to [40, Section 1.1] by Günther Ziegler. In the following, we use the $\mathcal{H}$-representation of polytopes to define some polytope properties.

Suppose we have a given d-polytope $P = \{x | Gx \leq h\}$ in $\mathbb{R}^d$. An inequality $w^\intercal x \leq t$ is called *valid* for P if it holds for P, i.e. $P \subseteq \{x | w^\intercal x \leq t\}$. A *hyperplane* is given by $\{x | w^\intercal x = t\}$. It is called a *supporting hyperplane* if $w^\intercal x \leq t$ is valid for P and $P \cap \{x | w^\intercal x = t\} \neq \emptyset$. The intersection of a supporting hyperplane with P is called a *face*. Every face of a polytope is by itself again a polytope, but then of a lower dimension. This follows directly from the definition as

$$\{Gx \leq h\} \cap \{w^\intercal x = t\} = \{G'x \leq h'\}, \tag{6}$$

where

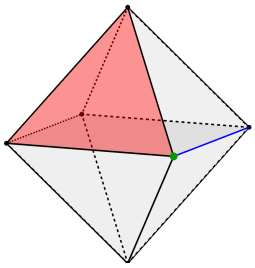
$$G' = \begin{pmatrix} w^\intercal \\ -w^\intercal \\ G \end{pmatrix}, \quad \text{and} \quad h' = \begin{pmatrix} t \\ -t \\ h \end{pmatrix}.$$

Note that a face is inclusion maximal in the supporting hyperplane that defines it.

Some faces have been given a special name, depending on the dimension of their affine span. Faces with affine dimensions zero and one are called *vertices* and *edges* respectively. Note that these names correspond with the well-known definitions of vertices and edges used in geometry. Faces with affine dimension $d - 1$ are called *facets*. The union of all facets of P is exactly the boundary of P as we will show in Proposition 2.4. A (supporting) hyperplane whose intersection with P forms a facet may also be called a *facet defining hyperplane*. The halfspace induced by this hyperplane that contains P is called a *facet defining halfspace*. We denote by h_f^+ the facet defining halfspace for facet f containing P. We will use the notion of facet defining hyperplanes and halfspaces in a triangulation algorithm in Section 2.3.

[2] The name *polyhedron* is used for two different definitions. First, a polyhedron denotes a three-dimensional polytope (as we mentioned in the beginning of this chapter). Second, a polyhedron denotes an intersection of halfspaces in general dimension without the restriction that it be bounded. Moreover, these definitions are not completely undisputed. There are authors that do not require convexity of polyhedra or polytopes. These discrepancies are examples of the lack of a universal system of definitions in polytope theory. The definitions used in this thesis are among the most widely used in modern polytope theory.

(a) 3-Polytope with a facet in red, an edge in blue and a vertex in green



(b) 2-Polytope with a facet defining halfspace

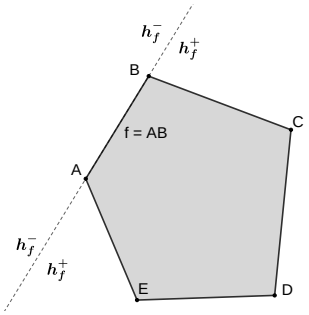


Figure 3: Examples of polytopes

Proposition 2.4
Let $P = \{x \in \mathbb{R}^d : Gx \leq h\}$ be a d-dimensional polytope. Then P is closed in $\mathbb{R}^d$ and the boundary $\partial(P)$ of P is the union of all facets of P

Proof. Let $P = \{x \in \mathbb{R}^d : Gx \leq h\}$ be a d-dimensional polytope. We prove that P is closed in $\mathbb{R}^d$ by showing that its complement $\mathbb{R}^d \backslash P$ is open in $\mathbb{R}^d$. Define $f : \mathbb{R}^d \to \mathbb{R}^m, x \mapsto Gx$. Note that f is a continuous linear mapping. P can now be rewritten as

$$P = \{x \in \mathbb{R}^d | Gx \leq h\} = \{x \in \mathbb{R}^d | \forall i \in [m] : (Gx)_i \leq h_i\}. \tag{7}$$

Now we use this expression to rewrite the complement of P as

$$\mathbb{R}^d \backslash P = \{x \in \mathbb{R}^d | \exists i \in [m] : (Gx)_i > h_i\} = \bigcup_{i \in [m]} f^{-1}\big((h_i, \infty)\big). \tag{8}$$

Since for every $i \in [m]$ (h_i, ∞) is open in $\mathbb{R}$ and since f is continuous, we see that $\mathbb{R}^d \backslash P$ is a union of open sets and is hence open. We conclude that P is closed.

We now show that $\partial(P)$ is the union of all facets of P. Let $\mathcal{F}$ be the union of all facets of P. Note that any inclusion-wise maximal face of P is a facet. This means that any non-trivial face of P is a subset of some facet of P. We write

$$\mathcal{F} = \{x \in \mathbb{R}^d : \exists i \text{ s.t. } (Gx)_i = h_i\}.$$

Let $x \in \mathcal{F}$. Then there is $i \in [m]$ such that $(Gx)_i = h_i$. Now take an open d-ball $B(x, \epsilon)$ with radius $\epsilon > 0$ around x. Since G is a non-trivial linear mapping, there is $y, y' \in B(x, \epsilon)$ such that $(Gy)_i < h_i$ and $(Gy')_i > h_i$. Hence $y \in P$ while $y' \in \mathbb{R} \backslash P$. This means that every neighbourhood of x intersects P as well as $\mathbb{R}^d \backslash P$. Hence $x \in \partial(P)$.

Now let $x \notin \mathcal{F}$. Suppose $x \notin P$. Then $x \notin \partial(P)$, since P is its own closure and the boundary of P lies in the closure of P. Suppose $x \in P \backslash \mathcal{F}$. Then for all $i \in [m]$ we have $(Gx)_i < h_i$. Therefore, for all y in the ball $B(Gx, \epsilon)$ with $\epsilon = \min_i(h_i - (Gx)_i)$, we have $(Gy)_i < h_i$ for all i. Since $x \mapsto Gx$ is continuous, there exists a ball $B(x, \delta)$ whose image under G is entirely contained in ball $B(Gx, \epsilon)$. Hence $B(x, \delta) \subset P$, which implies that x lies in the interior of P. This means that $x \notin \partial(P)$.

We have thus shown that $x \in \mathcal{F} \Rightarrow x \in \partial(P)$ and $x \notin \mathcal{F} \Rightarrow x \notin \partial(P)$. We conclude that $\mathcal{F} = \partial(P)$.

□

2.2 Triangulations

Triangulations are a natural way to decompose polytopes into smaller pieces that are easy to handle. Intuitively, a triangulation partitions a polytope into triangular shapes called *simplices*. As polytopes can be complex objects that are difficult to analyze, triangulations can be a useful tool. They can for instance be used to calculate a polytope's volume. They may furthermore be used to build a binary tree where vertices represent nodes. Triangulations may also be used for mathematical proofs about polytopes.

We now formally define the concept of triangulation of polytopes in Definitions 2.5 and 2.6.

Definition 2.5 (Simplex)**.** Let $d \geq 2$. A d-simplex is a polytope with vertices $v_0, \ldots, v_d$ such that vectors $v_1 - v_0, \ldots, v_d - v_0$ are linearly independent.
Equivalently, a $d-$simplex can be defined as a d-polytope with $d+1$ vertices.

The concept pf a simplex is a generalization of the triangle (2-simplex) and the tetrahedron (3-simplex) to a general dimension d.

Definition 2.6 (Triangulation)**.** Let P be a d-polytope in $\mathbb{R}^d$. Let $v(P)$ denote the vertex set of P. A triangulation of P is a finite collection $\mathcal{T}$ of d-simplices in $\mathbb{R}^d$ such that:

(T1) $P = \bigcup_{T \in \mathcal{T}} T$

(T2) $v(P) = \bigcup_{T \in \mathcal{T}} v(T)$

(T3) $\forall T_1, T_2 \in \mathcal{T}, T_1 \neq T_2 \Longrightarrow {T_1}^{\circ} \bigcap {T_2}^{\circ} = \emptyset$ (where T° denotes the interior of T)

(a) Triangulation of a 2-dimensional polytope (b) Triangulation of a 3-dimensional polytope

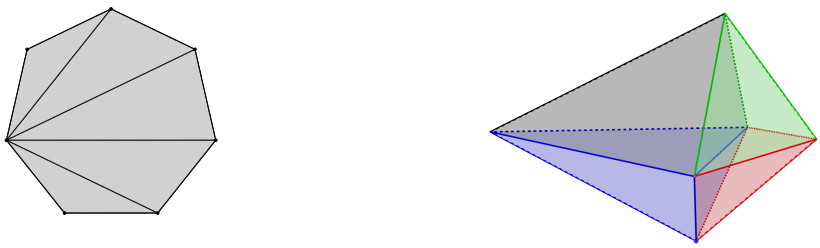


Figure 4: Examples of Triangulations

We note triangulations may also be defined for general convex sets. One may argue that it is also possible to triangulate convex bodies that are not polytopes using an infinite number of simplices. One may even want to triangulate non-convex bodies in $\mathbb{R}^d$. We purposely rule out these options by requiring a finite number of simplices in the triangulation. We make this restriction for the sake of simplicity, as is done in many other researches on polytopes.

A polytope's triangulation is in general not unique. There are usually many ways to triangulate a polytope. Subsequently the literature distinguishes between different classes of triangulations. The best known triangulation class is that of Delaunay triangulations, which have the property that the circum-hypersphere of any simplex does

not contain other vertices of the polytope. For more on Delaunay triangulations and the closely related concept of Voronoi diagrams, we refer the reader to [14, Chapter 27]. For material on the set of all triangulations of a polytope, we refer the reader to [14, Chapter 16].

There are many algorithms on polytope triangulations. Most of them are focused on triangulating 2-polytopes or 3-polytopes. We are interested in triangulation algorithms on d-polytopes. One way to triangulate a d-polytope in $\mathbb{R}^d$ is to lift it to $\mathbb{R}^{d+1}$ and look at the facets that are "visible from below". De Loera, Santos and Rambau show in [25, Section 2.2.1] that lifting a polytope to an extra dimension to get a triangulation is always possible. They use this to prove that any polytope always admits at least one triangulation. Though useful in a theoretical setting, we note that the lifting algorithm is not easily applied in practice. It requires a lot of information on the polytope that we want to triangulate. In particular, one has to know all facets of the lifted polytope.

We are particularly interested in triangulation algorithms that require the coordinates of the polytope vertices only. In that setting, triangulation algorithms are closely related to convex hull computation algorithms. According to Seidel [14, Chapter 27], these algorithms are mainly variations of three methods: divide-and-conquer, graph traversal, and incremental insertion.

There is a software package called 'Qhull' [4], which is based on the convex hull computation algorithm 'Quickhull', described in [3]. Quickhull is a divide-and-conquer algorithm that takes the input's extreme points per coordinate as starting points for the algorithm. One of the features of Qhull is that it can compute a Delaunay triangulation of a set of coordinates. The Qhull manual states that the program can be used for up to eight-dimensional polytopes. Indeed, when testing the algorithm, Qhull's Python implementation crashes when eight or more dimensions are provided as input. In the setting of portfolio spaces, where each asset class represents a dimension, Qhull's eight dimensions are not nearly enough.

Of the main methods used for convex hull computation algorithms, incremental insertion appears to be most appropriate for triangulating polytopes in large dimensions (> 10). In Section 2.3, we present our implementation of an incremental insertion algorithm called the "Beneath-and-Beyond algorithm".

2.3 Beneath-and-Beyond algorithm

In this section, we present a triangulation algorithm and prove its correctness. The algorithm is established and proved to be valid in the literature. The available proofs are however very high-level or rest on a different theoretic approach. In this section, we present a rigorous proof of our own, relying minimally on intuitive arguments.

The algorithm we use to triangulate polyopes is based on Kallay's "Beneath-and-Beyond" method for convex hull computation [33, Section 3.4.2]. The Beneath-and-Beyond algorithm computes the convex hull of a set of input points $x_1, \ldots, x_n \in \mathbb{R}^d$ by computing their triangulation. It is among the most natural algorithms on polytope constructions and is widely treated in the literature, for instance by Grünbaum in [17, Section 5.2] and by Edelsbrunner in [12, Section 8.4]. We use the revision of the Beneath-and-Beyond algorithm by Joswig [21] to formulate our algorithm below. The main differences between the algorithm we present below and Joswig's algorithm are that we compute a triangulation instead of a convex hull and that we make a few assumptions that apply specifically to our situation. In particular, we assume that all vertices of the polytope to be triangulated are known before-hand. Joswig assumes a general point set

in $\mathbb{R}^n$, not knowing which points will lie in the interior of their convex hull and which will be vertices. We also assume that the affine space of the input points is full-dimensional so that no transformations are necessary.

We present the two most important steps of the Beneath-and-Beyond algorithm in Algorithms 1 and 2. The final algorithm is presented in Algorithm 3. For convenience of the reader, we first explain the algorithm intuitively before starting on the foundations for a proof of its validity.

The idea of the algorithm is to start with vertices $\{x_1, \ldots, x_{d+1}\}$ that form a d-simplex. Its triangulation is trivial. We then iteratively add vertices and extend the triangulation until all points are added. After the initialization, every iteration consists of two steps. First, a new point is added to the polytope and its triangulation is extended. In the second step, the boundary of the polytope is updated as the collection of facets and their facet defining halfspaces.

When adding a point x_{k+1} to a polytope $P_k = \text{Conv}(x_1, \ldots, x_k)$, note that there are three separate possibilities. First, x_{k+1} may lie in the interior of P_k. In this case there is no need to update the polytope's triangulation or boundary. Second, x_{k+1} may be such that x_i lies in the interior of P_{k+1} for some $i \in \{1, \ldots, k\}$. Third, x_{k+1} may be such that $x_1, \ldots, x_k$ remain on the boundary of P_{k+1}.

As stated before, we assume that all points $\{x_1, \ldots, x_n\}$ are vertices of $P = \text{Conv}(x_1, \ldots, x_n)$. A vertex by definition does not lie in the interior of a polytope. Hence, for each point to be added, we can exclude the first two possibilities. From now on, we thus assume in our algorithm that x_{k+1} is always such that $x_1, \ldots, x_k$ remain on P_{k+1}'s boundary. This makes updating the triangulation easier, since we do not have to check which of the possibilities we deal with.

How should we extend the triangulation of P_k when adding one point x_{k+1}? First of all, note that we never have to remove a simplex from our current triangulation, but that we only have to add new simplices. We can focus on which new simplices we need to add. This depends on the facet defining hyperplanes that form the boundary $\mathcal{F}_k$ of P_k. The facet defining hyperplanes do not separate P_k's vertices $x_1, \ldots, x_k$. That is, all vertices will lie in the same facet defining halfspaces. However, for every point x_{k+1} in P_k's exterior, there are one or more facet defining hyperplanes that now separate $x_1, \ldots, x_k$ from x_{k+1}. We are looking for the facet defining hyperplanes that do so.

The separating facet defining hyperplanes determine both steps of our algorithm. For each facet defining hyperplane h_f of P_k that separates x_{k+1} from $\{x_1, \ldots x_k\}$, we need to add one or more simplices to our triangulation, namely the simplex or simplices formed by the vertices spanning f and the new point x_{k+1}. Furthermore, the facet f defined by h_f will no longer lie on the boundary of the updated polytope P_{k+1} and should be replaced in the second step.

We now present the two steps that form our triangulation algorithm.

Algorithm 1: EXTEND TRIANGULATION

Data: Triangulation $\mathcal{T}$ of polytope P, collection of facets $\mathcal{F}$ of P and corresponding facet defining halfspaces $\mathcal{H} = \{h_f^+ : f \in \mathcal{F}\}$, vertex x.
Result: Triangulation of $\text{Conv}(P, x)$

$\mathcal{T}' \leftarrow \mathcal{T}$;
foreach *facet* f *of* P **do**
  if $x \notin h_f^+$ **then**
    foreach $(d-1)$*-simplex* $T \in \mathcal{T}_f$ **do**
      Add the d-simplex formed by T and x to $\mathcal{T}'$
    end
  end
end
return $\mathcal{T}'$

Algorithm 2: UPDATE BOUNDARY

Data: Triangulation $\mathcal{T}$ of polytope $P = \text{Conv}(X)$
Result: full set of facet defining halfspaces of P

$\mathcal{F} \leftarrow \emptyset$;
foreach *simplex* $T \in \mathcal{T}$ **do**
  foreach *facet* f *of* T **do**
    if *facet defining hyperplane* h_f *does not separate* X **then**
      $\mathcal{F} \leftarrow \mathcal{F} \cup \{h_f^+\}$
    end
  end
end
return $\mathcal{F}$

(a) $P = \{x_1, \ldots, x_k\}$, $\mathcal{F} = \{f_1, \ldots, f_6\}$ (b) $P = \{x_1, \ldots, x_{k+1}\}$, $\mathcal{F} = \{f_1, f_3, \ldots, f_8\}$

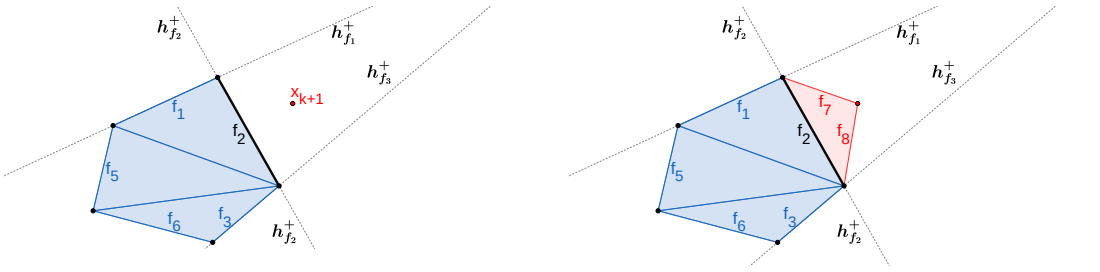


Figure 5: Illustration of the triangulation algorithm in which the steps EXTEND TRIANGULATION and UPDATE BOUNDARY are combined. Vertex x_{k+1} is added in this iteration. Facet f_2 has the only separating facet defining hyperplane. The triangulation is extended with the red triangle. f_2 was removed from the boundary and facets f_7 and f_8 were added.

In the above we use the notion of a triangulation restricted to facet f,

$$\mathcal{T}_f := \{T \cap f : T \in \mathcal{T}\}.$$

$\mathcal{T}_f$ turns out to be a triangulation of f, which we will formalize and prove in Proposition 2.11. The remainder of Section 2.3 is focused on proving the validity of Algorithms 1 and 2 in detail. We start by proving some useful lemmas.

Lemma 2.7
Let $P \in \mathbb{R}^d$ be convex and let $x \in \mathbb{R}^d$. Then

$$\text{Conv}(P, x) = \bigcup_{p \in P} \{\lambda p + (1 - \lambda)x : \lambda \in [0, 1]\}. \tag{9}$$

Proof. We prove inclusion both ways.
Let

$$y \in \bigcup_{p \in P} \{\lambda p + (1 - \lambda)x : \lambda \in [0, 1]\}.$$

Then y is a convex combination of an element in P and x. By definition $y \in \text{Conv}(P, x)$.

Now, let $y \in \text{Conv}(P, x)$. Then we can write y as

$$y = t_0 x + t_1 p_1 + t_2 p_2 + \cdots + t_n p_n,$$

where

$$p_1, \ldots, p_n \in P,\ t_0, \ldots, t_n \geq 0,\ \sum_{i=0}^{n} t_i = 1.$$

We now distinguish three possibilities. First, if $t_0 = 1$, then $y = t_0 x + (1 - t_0)p$ for any $p \in P$. Second, if $t_0 = 0$ then $y \in P$ by convexity of P and $y = t_0 x + (1 - t_0)y$. Lastly, Suppose $t_0 \in (0, 1)$. Define $p = \frac{1}{1-t_0}(t_1 p_1 + \cdots + t_n p_n)$. Note that $\frac{t_1}{1-t_0}, \ldots, \frac{t_n}{1-t_0}, \geq 0$ and $\sum_{i=1}^{n} \frac{t_i}{1-t_0} = 1$. Hence $p \in P$ by convexity of P. Now we observe

$$y = t_0 x + t_1 p_1 + \cdots + t_n p_n = t_0 x + (1 - t_0)p,$$

and thus,

$$y \in \bigcup_{p \in P} \{\lambda p + (1 - \lambda)x : \lambda \in [0, 1]\}.$$

□

Lemma 2.8
Let P be a d-polytope in $\mathbb{R}^d$. Let f be a facet of P with facet defining hyperplane h_f and the facet defining halfspace h_f^+. Let $x \in f^o$. Then for all $a \in h_f^+$, there exists $\delta > 0$ such that for all $\lambda \in [0, \delta]$, we have $x + \lambda(a - x) \in P$.

Intuitively, the claim of Lemma 2.8 is that starting from x, we can walk a little in every direction in h_f^+ without leaving P.

Proof. Let $a \in h_f^+$. We show that we can walk $\delta > 0$ from x in the direction of a without leaving P by breaking up the vector $\delta(a - x)$ into a component in $h_f \subset P$ and a component in P^o. We then use convexity of P to conclude that $x + \delta(a - x)$ lies inside P.

Define

$$h_f - x := \{y - x : y \in h_f\}.$$

Since h_f is a $(d - 1)$-dimensional hyperplane and $x \in h_f$, we know that $h_f - x$ is a $(d - 1)$-dimensional linear subspace of $\mathbb{R}^d$. Let $p \in P^o$. Then $p \notin h_f$ (using Proposition 2.4). Hence $p - x \notin h_f - x$ and thus $\text{Span}(p - x, h_f - x) = \mathbb{R}^d$. This means that there are $\beta \in \mathbb{R}$ and $y \in h_f$ such that

$$a - x = \beta(p - x) + (y - x).$$

Since $a, p \in h_f^+$, we know that $\beta > 0$.
We furthermore know that

$$\forall t \in [0,1] : x + t(p - x) \in P \qquad \text{(by convexity of } P\text{)} \tag{10}$$

and

$$\exists \epsilon > 0 \text{ such that } \forall t \in [0, \epsilon] : x + t(y - x) \in P \qquad \text{(since } x \in f^{\text{o}}\text{)}. \tag{11}$$

We define $\delta = \frac{1}{2}\min(\frac{1}{\beta}, \epsilon)$ and find that for all $t \in [0, \delta]$,

$$x + t(a - x) = x + t(\beta(p - x) + (y - x)) \tag{12}$$

$$= \frac{1}{2}(x + 2\beta t(p - x)) + \frac{1}{2}(x + 2t(y - x)). \tag{13}$$

Since $2\beta t \in [0,1]$ and $2t \in [0, \delta]$ and using (10) and (11) we find that (13) is a convex combination of two elements of P. By convexity of P, we conclude that $x + t(a - x) \in P$. In particular, $x + \delta(a - x) \in P$. Since $x \in P$, the result follows again by convexity of P. □

We introduce the following notation, which we use in Lemmas 2.9 and 2.12 and in Proposition 2.11:

$$l(x, y) := \{\lambda x + (1 - \lambda)y : \lambda \in [0, 1]\} \qquad \text{(for } x, y \in \mathbb{R}^d\text{)}$$

Lemma 2.9
Let $X, Y \subseteq \mathbb{R}^{d-1} \times \{0\}$ be $(d-1)$-polytopes. Let $z \in \mathbb{R}^d$ be non-zero in the d-th coordinate. For $a \in \text{Conv}(X, z)$ with $a \neq z$, let a' be the unique point in X such that $a \in l(a', z)$. Then we have

(i)

$$a \in \text{Conv}(X, z)^{\text{o}} \iff a' \in X^{\text{o}},$$

(ii)

$$X^{\text{o}} \cap Y^{\text{o}} = \emptyset \iff (\text{Conv}(X, z))^{\text{o}} \cap (\text{Conv}(Y, z))^{\text{o}} = \emptyset,$$

where $X^{\text{o}}, Y^{\text{o}}$ are relative interiors in $\mathbb{R}^{d-1}$ and $(\text{Conv}(X, z))^{\text{o}}$, $(\text{Conv}(Y, z))^{\text{o}}$ are relative interiors in $\mathbb{R}^d$.

Proof. Denote $C_X = \text{Conv}(X, z)$ and $C_Y = \text{Conv}(Y, z)$. C_X and C_Y may also be called *cones* in $\mathbb{R}^d$. Using Lemma 2.7 [3],

$$C_X = \bigcup_{x \in X} l(x, z) = \bigcup_{x \in X} \{\lambda x + (1 - \lambda)z : \lambda \in [0, 1]\}$$
$$C_Y = \bigcup_{y \in Y} l(y, z) = \bigcup_{y \in Y} \{\lambda y + (1 - \lambda)z : \lambda \in [0, 1]\}.$$

(i) Without loss of generality, assume $z_d > 0$.
Let $a \in C_X, a' \in X$ be given with $\lambda \in (0, 1)$ such that $a = \lambda a' + (1 - \lambda)z$ and $a \neq z$. Denote $a = (a_1, \ldots, a_d)$.

[3] We do not necessarily need convexity of X and Y. It suffices if X and Y are closed and bounded in $\mathbb{R}^{d-1}$ and hence compact. Cones with compact bases in the Euclidean space are equivalent to a union of line segments.

Suppose $a' \in X^{\circ}$. So there is $\epsilon > 0$ such that

$$\{y \in \mathbb{R}^{d-1} \times \{0\} : \|y - a'\| < \epsilon\} \subseteq X.$$

Define

$$H := \{(x_1, \ldots, x_d) \in \mathbb{R}^d : x_d = a_d\}.$$

H is a hyperplane through a, parallel to $\mathbb{R}^{d-1} \times \{0\}$. Then $C_X \cap H$ is a polytope with the same topology as X. In particular,

$$\{y \in H : \|y - a\| < \lambda \cdot \epsilon\} \subseteq C_X \cap H.$$

We can thus move from a both ways in the first $d-1$ coordinates without leaving C_X.

Let $z' \in X$. We note that $\mathrm{Conv}(C_X \cap H, z)$ as well as $\mathrm{Conv}(C_X \cap H, z')$ are polytopes in $\mathbb{R}^d$ with facet $C_X \cap H$ and facet defining hyperplane H. Using Lemma 2.8, we find that there are $\delta, \delta' > 0$ such that

$$(a_1, \ldots, a_{d-1}, a_d + \delta) \in \mathrm{Conv}(C_X \cap H, z) \subset C_X$$

and

$$(a_1, \ldots, a_{d-1}, a_d - \delta') \in \mathrm{Conv}(C_X \cap H, z') \subset C_X.$$

We thus find that we can also move from a the d-th coordinate both ways without leaving C_X. By convexity of C_X, we conclude that $a \in {C_X}^{\circ}$.

Now suppose $a \in {C_X}^{\circ}$. There is $\epsilon > 0$ such that d-ball $B(a, \epsilon) \in C_X$. Let λ be such that $a = \lambda a' + (1 - \lambda) z$. Since by assumption $a \neq z$, and since ${C_X}^{\circ} \cap X = \emptyset$, we know that $\lambda \in (0, 1)$. Now $(d-1)$-ball

$$B\left(a', \frac{\epsilon}{1-\lambda}\right) := \{x \in \mathbb{R}^{d-1} : \|a' - x\| < \frac{\epsilon}{1-\lambda}\} \times \{0\}$$

is the projection of $B(a, \epsilon)$ on X as seen from z. Hence $B(a', \frac{\epsilon}{1-\lambda}) \subseteq X$ and thus $a' \in X^{\circ}$.

(ii) Let $a \in C_X \cap C_Y$ such that $a \neq z$. Now there are $x \in X, y \in Y$ such that $a \in l(x, z)$ and $a \in l(y, z)$. There is only one line in $\mathbb{R}^d$ through a and z and that line can only intersect $\mathbb{R}^{d-1} \times \{0\}$ in one point. Hence $x = y$. From now on denote $a' = x = y$. Now it suffices to show that

$$a' \in X^{\circ} \cap Y^{\circ} \iff a \in {C_X}^{\circ} \cap {C_Y}^{\circ}.$$

Using the first part of this lemma, we find the desired result.
We have thus shown that for $a \in C_X \cap C_Y$, we have

$$X^{\circ} \cap Y^{\circ} = \emptyset \iff (\mathrm{Conv}(X, z))^{\circ} \cap (\mathrm{Conv}(Y, z))^{\circ} = \emptyset.$$

□

Corollary 2.10
Let X, Y be convex in a $(d-1)$-dimensional hyperplane H in $\mathbb{R}^d$. Let $z \in \mathbb{R}^d \backslash H$. For $a \in \mathrm{Conv}(X, z)$, let a' be the unique point in X such that $a \in l(a', z)$. Then we have

$$a \in \mathrm{Conv}(X, z)^{\circ} \iff a' \in X^{\circ},$$

$$X^{\circ} \cap Y^{\circ} = \emptyset \iff (\mathrm{Conv}(X, z))^{\circ} \cap (\mathrm{Conv}(Y, z))^{\circ} = \emptyset.$$

The results from Corollary 2.10 follow directly from Lemma 2.9. We only have to translate and rotate the space until H coincides with $\mathbb{R}^{d-1}$. The proofs of Lemma 2.9 then still hold.

Proposition 2.11
Let $P \subset \mathbb{R}^d$ be a d-polytope with boundary $\mathcal{F}$ and triangulation $\mathcal{T}$. Let $f \in \mathcal{F}$ be a facet of P and $x \in \mathbb{R}^d \backslash P$. Then we have

(i) $\mathcal{T}_f := \{T \cap f : T \in \mathcal{T}\}$ *is a triangulation of* f.

(ii) $\mathrm{Conv}(f, x) = \bigcup_{T \in \mathcal{T}_f} \mathrm{Conv}(T, x)$.

(iii) $\mathrm{Conv}(P, x) = P \cup \left(\bigcup_{f \in \mathcal{F}} \mathrm{Conv}(f, x) \right)$.

Proof. We write $P = \{x \in \mathbb{R}^d | Ax \leq b\}$ for $A \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^m$.

(i) Note that f is a $(d-1)$-polytope. We show that $\mathcal{T}_f$ is a triangulation of f by checking the properties *(T1-T3)* from Definition 2.6. For all $x \in f \subset P$, there is $T \in \mathcal{T}$ such that $x \in T$. Hence $x \in T \cap f \in \mathcal{T}_f$ *(T1)*.

Let v be a vertex of f. Since f is inclusion maximal and lies on the boundary of P, v is also a vertex of P. Then there is $T \in \mathcal{T}$ such that v is a vertex of T. Hence $v \in T \cap f$. Being a vertex of T, v cannot be written as a convex combination of other elements in T. Since $T \cap f \subseteq T$, v can also not be written as a convex combination of other elements in $T \cap f$. Therefore v is a vertex of $T \cap f$.

Now let v be a vertex of $T \cap f \in \mathcal{T}_f$. Since f is inclusion maximal and lies on the boundary of P, v must be a vertex of $T \in \mathcal{T}$. Since $\mathcal{T}$ is a triangulation, v is a vertex of P. We see that $v \in f \subset P$ and v is a vertex of P. Hence, v is a vertex of f. We conclude that

$$v(f) = \cup_{T \in \mathcal{T}_f} v(T).$$

Hence, $\mathcal{T}_f$ satisfies property *(T2)*.

Let $T_1, T_2 \in \mathcal{T}_f$ with $T_1 \neq T_2$ be non-empty. Then there are $T_1^*, T_2^* \in \mathcal{T}$ such that $T_1 = T_1^* \cap f$, $T_2 = T_2^* \cap f$. Then also $T_1^* \neq T_2^*$. Since $\mathcal{T}$ is a triangulation, we have $(T_1^*)^{\mathrm{o}} \cap (T_2^*)^{\mathrm{o}} = \emptyset$. We note that T_1, T_2 have affine dimension $d-1$ while T_1^*, T_2^* have affine dimension d. Moreover, we note that T_1^*, T_2^* are themselves polytopes with facets T_1, T_2, respectively.

Suppose $x \in {T_1}^{\mathrm{o}} \cap {T_2}^{\mathrm{o}}$. Using Lemma 2.8, we can walk $\delta > 0$ in any direction in h_f^+ without leaving T_1^* or T_2^*. In particular, there is $p \in (T_1^*)^{\mathrm{o}} \cap (T_2^*)^{\mathrm{o}}$. This is a contradiction. We conclude that ${T_1}^{\mathrm{o}} \cap {T_2}^{\mathrm{o}} = \emptyset$ *(T3)*.

(ii) Let $y \in \bigcup_{T \in \mathcal{T}_f} \mathrm{Conv}(T, x)$. Using Lemma 2.7, there is $t \in T \in \mathcal{T}_f, \lambda \in [0, 1]$ such that $y = \lambda t + (1 - \lambda)x$. So y is a convex combination of $t \in f$ and x, i.e. $y \in \mathrm{Conv}(f, x)$.

Let $y \in \mathrm{Conv}(f, x)$. Using Lemma 2.7, there is $t \in f, \lambda \in [0, 1]$ such that $y \in \lambda t + (1 - \lambda)x$. By (1) of this proposition, $\mathcal{T}_f$ is a triangulation of f, so there exists $T \in \mathcal{T}_f$ with $t \in \mathcal{T}_f$. Therefore

$$y \in \mathrm{Conv}(T, x) \subseteq \bigcup_{T \in \mathcal{T}_f} \mathrm{Conv}(T, x).$$

(iii) We prove inclusion both ways. $P \subset \text{Conv}(P, x)$ is immediate. Let $y \in \bigcup_{f \in \mathcal{F}} \text{Conv}(f, x)$. Using Lemma 2.7, we have $z \in f, \lambda \in [0, 1]$ such that $y = \lambda z + (1 - \lambda)x$. Since $z \in f \subset P$, we have $y \in \text{Conv}(P, x)$ and thus

$$P \cup \left(\bigcup_{f \in \mathcal{F}} \text{Conv}(f, x) \right) \subseteq \text{Conv}(P, x).$$

To prove inclusion the other way, let $y \in \text{Conv}(P, x) \backslash P$. Using Lemma 2.7, let $p \in P, \lambda \in [0, 1]$ such that $y = \lambda p + (1 - \lambda)x$. Consider the line segment

$$l(p, x) = \{\lambda p + (1 - \lambda)x | \lambda \in [0, 1]\}.$$

Since P is closed (Proposition 2.4), $x \notin P, p \in P$ and using that $\mathcal{F}$ is the boundary of P, we know that $\mathcal{F} \cap l(p, x) \neq \emptyset$. Hence there is a facet $f \in \mathcal{F}$ containing a point $y' \in l(p, x)$ such that $y \in l(y', x)$. Therefore,

$$y \in \text{Conv}(f, x) \in \cup_{f \in \mathcal{F}} \text{Conv}(f, x).$$

□

The following lemma tells us that we only need to consider facets whose hyperplanes separate the newly added point from current polytope vertices in order to extend the triangulation.

Lemma 2.12
Let P be a d-polytope with vertices $x_1, \ldots, x_k$ and collection of facets $\mathcal{F}$. Let $x_{k+1} \in \mathbb{R}^d \backslash P$ such that $x_1, \ldots, x_{k+1}$ are all vertices of $P^* = \text{Conv}(P, x_{k+1})$. Define $\mathcal{F}^- = \{f \in \mathcal{F} : x_{k+1} \notin h_f^+\}$. Then,

$$P^* = P \cup \left(\bigcup_{f \in \mathcal{F}^-} \text{Conv}(f, x_{k+1}) \right).$$

Proof. From Proposition 2.7 we know that $P^* = P \cup \left(\bigcup_{f \in \mathcal{F}} \text{Conv}(f, x_{k+1}) \right)$. We thus have to show that we capture all of P^* when only considering facets that separate x_{k+1} from $x_1, \ldots, x_k$. To do so, let

$$f \in \mathcal{F}^+ := \{f \in \mathcal{F} : x_{k+1} \in h_f^+\}$$

and consider $z \in \text{Conv}(f, x_{k+1}) \backslash P$. We will show that there is $f' \in \mathcal{F}^-$ such that $z \in \text{Conv}(f', x_{k+1})$.
By Lemma 2.7, there is $y \in f$ such that $z \in l(y, x_{k+1})$. We distinguish two possibilities.

First, suppose that $l(y, x_{k+1})$ intersects P in a point other than y. So a part of $l(y, z_{k+1})$ goes through P. By convexity of P there is exactly one point $y' \in l(y, x_{k+1})$ on the boundary of P (since P is closed) where $l(y, x_{k+1})$ "leaves" P. This point lies on a facet f', using Proposition 2.4. Since $z \notin P$, we know that $z \in l(y', x_{k+1})$. Clearly, P lies on one side of f' and x_{k+1} on the other. This means that $h_{f'}$ separates x_{k+1} from P and hence $f' \in \mathcal{F}^-$. Using Lemma 2.7 we conclude that $z \in \bigcup_{f \in \mathcal{F}^-} \text{Conv}(f, x_{k+1})$

Second, suppose that $l(y, x_{k+1})$ intersects P in y only. Recall that $P = \{x \in \mathbb{R}^d : Ax \leq b\}$ and $f = \{x \in P : (Ax)_i = b_i\}$ for some i. Since $x_{k+1} \in h_f^+$, we know that for all $x \in l(y, x_{k+1})$: $(Ax) \leq b$. However, since

$$l(y, x_{k+1}) \cap P = \{y\},$$

there is at least one hyperplane $\{(Ax)_j = b_j\}$ for $j \neq i$ such that $(Ax)_j > b_j$ for all $x \in l(y, x_{k+1})$, $x \neq y$. Since $l(y, x_{k+1})$ is a continuous line segment and $(Ay)_j \leq b_j$, this implies that $(Ay)_j = b_j$. We now see that y lies on a second facet

$$f' = \{x \in \mathbb{R}^d : Ax \leq b, (Ax)_j = b_j\}.$$

We observe that $x_{k+1} \notin h^+_{f'}$ and hence $f' \in \mathcal{F}^-$. Using Lemma 2.7 we conclude that

$$z \in \bigcup_{f \in \mathcal{F}^-} \text{Conv}(f, x_{k+1}).$$

□

We are now ready to prove the validity Algorithms 1 (EXTEND TRIANGULATION) and 2 (UPDATE BOUNDARY).

Theorem 2.13 (EXTEND TRIANGULATION)
Algorithm 1 finds a valid triangulation of $P^* = \text{Conv}(x_1, \ldots, x_{k+1})$ *given a triangulation* $\mathcal{T}$ *of* $P = \text{Conv}(x_1, \ldots, x_k)$ *and the corresponding boundary* $\mathcal{F}$.

Proof. Suppose we have a triangulation $\mathcal{T}$ of P and $\mathcal{F}$ the collection of facets of P. We add one point x_{k+1} using our triangulation algorithm, yielding $P^* = \text{Conv}(P, x_{k+1})$ and $\mathcal{T}^*$. We show that $\mathcal{T}^*$ triangulates P^* by checking the three properties of Definition 2.6.

First we show that for every $x \in P^*$, there is $T \in \mathcal{T}^*$ such that $x \in T$ (*T1*).
For $x \in P$, this follows immediately since $\mathcal{T} \subset \mathcal{T}^*$.
Suppose $x \in P^* \backslash P$. By Lemma 2.12, there is $f \in \mathcal{F}^-$ such that $x \in \text{Conv}(f, x_{k+1})$. By Proposition 2.11 part 2, there is $T \in \mathcal{T}_f$ such that $x \in \text{Conv}(T, x_{k+1}) \subseteq \mathcal{T}^*$.

Second, we show that the vertex set $v(P^*) = \{x_1, \ldots, x_{k+1}\}$ equals the set of vertices of simplices in $\mathcal{T}^*$ (*T2*), $v(\mathcal{T}^*) := \bigcup_{T \in \mathcal{T}^*} v(T)$. We know that the vertex set of P equals the vertex set of $\mathcal{T}$:

$$v(\mathcal{T}) = v(P) = \{x_1, \ldots, x_k\}.$$

It follows immediately from the algorithm that $x_{k+1} \in v(\mathcal{T}^*)$, so that $v(P^*) \subseteq v(\mathcal{T}^*)$.

Let $T \in \mathcal{T}^* \backslash \mathcal{T}$. Then by construction, $T = \text{Conv}(t, x_{k+1})$ for some $t \in \mathcal{T}_f$. Using that all vertices of facet f are also vertices of P and that $\mathcal{T}_f$ is a triangulation of f, we conclude $v(T) \subseteq v(P^*)$ and hence $v(\mathcal{T}^*) \subseteq v(P^*)$

Finally, let $T_1, T_2 \in \mathcal{T}^*$ with $T_1 \neq T_2$. We show that their interiors are disjoint, ${T_1}^o \cap {T_2}^o = \emptyset$ (*T3*). If either T_1 or T_2 is in $\mathcal{T}$, then the property follows easily, since $\mathcal{T}$ is a triangulation of P and any newly added triangle by construction does not lie in the interior of P. Suppose $T_1, T_2 \notin \mathcal{T}$. Then there are $(d-1)$-simplices $t_1 \in \mathcal{T}_{f_1}$, $t_2 \in \mathcal{T}_{f_2}$ for $f_1, f_2 \in \mathcal{F}^-$ such that $T_1 = \text{Conv}(t_1, x_{k+1})$, $T_2 = \text{Conv}(t_2, x_{k+1})$. We now distinguish two possibilities:

First, suppose that $f_1 = f_2 = f$. Then $t_1, t_2 \in \mathcal{T}_f$ and thus ${t_1}^o \cap {t_2}^o = \emptyset$. Using Corollary 2.10 we know that ${T_1}^o \cap {T_2}^o = \emptyset$.

Second, suppose that $f_1 \neq f_2$. The intersection of f_1 and f_2 is a face of dimension at most $d-2$ and lies on the boundary of both facets. Hence ${f_1}^o \cap {f_2}^o = \emptyset$ from which it follows that ${t_1}^o \cap {t_2}^o = \emptyset$. Let $y \in {T_1}^o$. Then by Lemma 2.7 there is $y' \in t_1$, such that $y \in l(y', x_{k+1})$. Since $y' \in f_1$ lies on the boundary of P and $x_{k+1} \notin h^+_{f_1}$, we know that except for in y', $l(y', x_{k+1})$ lies completely outside of P and thus does not intersect t_2. By Lemma 2.9, $y' \in {t_1}^o$ and thus $y' \notin {t_2}^o$. Again using Lemma 2.9, we conclude that $y \notin {T_2}^o$. Similarly, we find that $y \in {T_2}^o \Rightarrow y \notin {T_1}^o$.

We conclude that in both cases, ${T_1}^o \cap {T_2}^o = \emptyset$. □

Theorem 2.14 (UPDATE BOUNDARY)
Given a polytope $P \subset \mathbb{R}^d$ with triangulation $\mathcal{T}$, Algorithm 2 copmutes all facet defining hyperplanes of the boundary $\mathcal{F}$ of P.

Proof. Using Proposition 2.4, the boundary of a polytope is equal to the union of all facets. Algorithm 2 directly uses the definition of a facet to find all facets of P and their hyperplanes.

□

Now we have defined the two steps that shape the Beneath-and-Beyond triangulation algorithm.

Algorithm 3: Beneath-and-Beyond triangulation algorithm

Data: A set of points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ such that $\text{aff}(X)$ is d-dimensional and such that $x_1, \ldots, x_n \in \partial(P)$, where $P = \text{Conv}(X)$.
Result: A complete triangulation of P

```
Initialization: P = {x_1, ..., x_{d+1}}, T = {(x_1, ..., x_{d+1})} such that (x_1, ..., x_{d+1})
  is a d-simplex, F = {f ⊂ P : |f| = d};
for x ∈ X\{x_1, ..., x_{d+1}} do
    T ← EXTEND TRIANGULATION(T, P, F, x) ;
    F ← UPDATE BOUNDARY(P, T) ;
    P ← P ∪ {x}
end
return T
```

Theorem 2.15
The Beneath-and-Beyond triangulation algorithm (Algorithm 3) gives a valid triangulation for a d-polytope $P = \text{Conv}(x_1, \ldots, x_n) \subset \mathbb{R}^d$.

Proof. Note that the algorithm iterates over a finite set of points. We can thus conclude that it terminates. By Theorems 2.13 and 2.14, the two steps EXTEND TRIANGULATION and UPDATE BOUNDARY produce valid output. What remains to prove is that the initialization is valid. By an argument of induction, we then conclude that the algorithm indeed computes a valid triangulation.

The initialization starts with a d-simplex $x_1, \ldots, x_{d+1}$. We note that it is always possible to find a d-simplex, since the total input has affine dimension d. We need to select combinations of $d + 1$ points and check their affine dimension. This may potentially be time-consuming, but it is nevertheless a feasible and solvable problem. The triangulation of such a d-simplex is trivial, as is the boundary. It is thus easily seen that the initialization is valid. □

2.4 Triangulation of embedded polytopes

So far we have only considered polytopes that are full-dimensional in their original space. We may however want to triangulate polytopes that can be embedded in lower dimensional spaces. Then, the Beneath-and-Beyond triangulation algorithm from Section 2.3 does not immediately apply. In this section, we develop our own method to transform a polytope into its affine lower-dimensional space. We then show that the method of triangulation can still be applied after transformation.

Let us start by presenting an example of polytopes that are not full-dimensional.

Consider the space of three-dimensional portfolios where we require for every portfolio that the weights of the three assets in it sum to one. The set of feasible portfolios then lies in the plane $x + y + z = 1$, which can be embedded in $\mathbb{R}^2$. The polytope P of near-optimal portfolios may be such as in Figure 6. Clearly the polytope's affine subspace lies in $\mathbb{R}^2$. It will be problematic when trying to partition P using three-dimensional simplices (tetrahedra). We should instead use two-dimensional simplices (triangles) in the plane $x + y + z = 1$ to partition P.

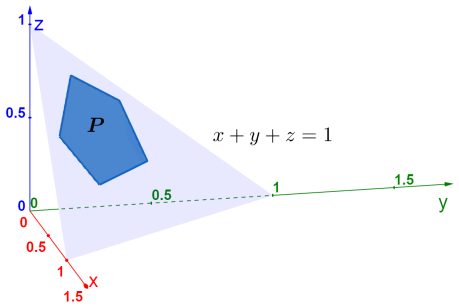


Figure 6: Example set of near-optimal portfolios in a three-asset portfolio space

Example 2.16. In portfolio spaces, we generally deal with a body in a high dimension that can be embedded in a lower dimensional space.

When using higher dimensional portfolio spaces with more equality constraints, we may have to use simplices that are multiple dimensions lower than our original portfolio space.

We introduce the approach to triangulate a polytope P that is not-full-dimensional. The idea is to transform P into its affine (lower-dimensional) space to construct a new polytope P' which is full-dimensional. Then we can apply a triangulation algorithm to find the transformed triangulation $\mathcal{T}'$. In a final step, we find triangulation $\mathcal{T}$ of P by transforming back to the original dimension.

In the following, consider a k-polytope $P \subset \mathbb{R}^d$ with vertices $v_1, \ldots, v_n \in \mathbb{R}^d$ $(k < d)$. P can be embedded in $\mathbb{R}^k$, hence P satisfies equality constraints $Ax = b$ for some full rank $A \in \mathbb{R}^{\left((d-k)\times d\right)}$, $b \in \mathbb{R}^{(d-k)}$. All solutions of the system $Ax = b$ are given by

$$x = A^+ b + (I - A^+ A)w \tag{14}$$

for arbitrary vector $w \in \mathbb{R}^d$, where A^+ denotes the Moore-Penrose inverse (pseudo-inverse) of A and I denotes the d-dimensional identity matrix. Note that $M = (I - A^+ A)$ has rank k. To reduce the dimension of P, we construct an orthonormal basis of M. Denote by B the orthonormal matrix with k basis vectors of M in the columns. We apply the dimension reduction to identity (14) to find

$$w_i = B^\intercal (v_i - A^+ b) \qquad\qquad (i = 1, \ldots, n) \tag{15}$$

Now it is ensured that $P' = \text{Conv}\left(\{w_0, \ldots, w_n\}\right)$ is full-dimensional in $\mathbb{R}^k$.

Let us summarize the described procedure in the following definition.

Definition 2.17 (Transformation to affine space)**.** Let P be a k-polytope in $\mathbb{R}^d$ $(k < d)$. Let $A \in \mathbb{R}^{(d-k)\times d}$ be of full rank and $b \in \mathbb{R}^{d-k}$ such that for all x in P, we have $Ax = b$. We find the *transformed polytope* $\phi(P)$ in the affine space of P as follows:

- Let A^+ be the Moore-Penrose pseudo-inverse of A
- Let $M = I - A^+ A$

- Let B be a k-dimensional orthonormal basis of M
- We define the transformation function $\phi : P \to \mathbb{R}^k$ by

$$\phi(x) = B^\intercal(x - A^+ b)$$

We show that the above method transforms polytope P into a congruent polytope $P' = \phi(P)$ that is full-dimensional in the affine space of P. We furthermore show that a triangulation $\mathcal{T}$ of P is transformed to a triangulation $\mathcal{T}' = \phi(\mathcal{T})$ of P'. We gather the necessary information in Lemmas 2.18 and 2.19.

Lemma 2.18
Let $B \in \mathbb{R}^{d\times m}$ be an orthonormal matrix of rank m and let $c \in \mathbb{R}^d$ for $m < d$. Define $\phi : \mathbb{R}^d \to \mathbb{R}^k$ by $x \mapsto B^\intercal(x + c)$. Then ϕ is an isometry with respect to the Euclidean distance.

Proof. We note that we can write ϕ as the composition of two functions $f = g \circ h$, where

$$g(x) = B^\intercal x, \quad h(x) = x + c.$$

If g and h are both isometries, then so is ϕ. Note that h is just a vector translation in $\mathbb{R}^d$. It is thus easily seen that h is an isometry. To prove that g is an isometry, we show that g preserves the Euclidean norm, i.e. $\|g(x)\| = \|x\|$. We observe that

$$\|g(x)\|^2 = \|B^\intercal x\|^2 = x^\intercal B \cdot B^\intercal x = x^\intercal x = \|x\|^2,$$

where $B \cdot B^\intercal$ is the identity matrix because B is orthonormal. Since the Euclidean norm $\|\cdot\|$ is non-negative, this concludes the proof.

□

Lemma 2.19
Let $\phi : \mathbb{R}^d \to \mathbb{R}^k$ be defined by $\phi(x) = B(x + c)$ where $B \in \mathbb{R}^{d\times m}$ and $b \in \mathbb{R}^m$ and let $P \subseteq \mathbb{R}^d$ be a polytope. Then ϕ preserves convex combinations, i.e.

$$\forall x_1, \ldots, x_n \in \mathbb{R}^d, \lambda_1, \ldots, \lambda_n \in \mathbb{R}_{\geq 0} \text{ such that } \sum_i \lambda_i = 1, \text{ we have,}$$
$$\phi(\lambda_1 x_1 + \cdots + \lambda_n x_n) = \lambda_1 \phi(x_1) + \cdots + \lambda_n \phi(x_n).$$

and moreover, $\phi(P)$ is a polytope.

Proof. The proof of that ϕ preserves convex combinations follows directly from the distributive property of the inner product. To prove that $\phi(P)$ is a polytope, it suffices to show that $\phi(P)$ is the convex hull of a finite set of points in $\mathbb{R}^k$. First, we show that $\phi(P)$ is convex. Then we show that the vertices of P are mapped to vertices of $\phi(P)$ and span the whole of $\phi(P)$.

To show that $\phi(P)$ is convex, we use that ϕ preserves convex combinations. Let $x^*, y^* \in \phi(P)$ and let $\lambda \in [0, 1]$. Let $x, y \in \mathbb{R}^d$ such that $\phi(x) = x^*, \phi(y) = y^*$. Since $x, y \in P$, we have by convexity of P that $\lambda x + (1 - \lambda)y \in P$. Then

$$\lambda x^* + (1 - \lambda)y^* = \lambda\phi(x) + (1 - \lambda)\phi(y) = \phi(\lambda x + (1 - \lambda)y) \in \phi(P).$$

We conclude that $\phi(P)$ is indeed convex.

Suppose P has vertices $v_1, \ldots, v_n$. Let $y \in \phi(P)$. Then there is $x \in P$ such that $\phi(x) = y$. We can write

$$x = \sum_{i=0}^{n} \lambda_i v_i \text{ where } \sum_i \lambda_i = 1 \text{ and } \lambda_i \geq 0 \text{ for all } i.$$

Hence, using that ϕ preserves convex combinations, we have

$$y = \phi(x) = \sum_{i=0}^{n} \lambda_i \phi(v_i).$$

Every element of $\phi(P)$ can be thus written as a convex combination of $\phi(v_0), \ldots, \phi(v_n)$. Hence $\phi(v_1), \ldots, \phi(v_n)$ span the set $\phi(P)$. To show that $\phi(v_1), \ldots, \phi(v_n)$ are indeed vertices, we show that they cannot be written as a convex combination of other elements of $\phi(P)$ themselves. Suppose that for some $i \in [n]$, $\phi(v_i)$ can be written as a convex combination $\{\phi(v_i), \ldots, \phi(v_{i-1}), \phi(v_{i+1}), \ldots, \phi(v_n)\}$. This implies that v_i can be written as a convex combination of $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$. This is a contradiction since vertices by definition cannot be written as a convex combination of other vertices. Hence $\phi(v_i)$ cannot be written as a convex combination of $\phi(v_1) \ldots, \phi(v_n)$ and is a vertex of $\phi(P)$.

We conclude that $\phi(v_0), \ldots, \phi(v_n)$ are all vertices of $\phi(P)$ and that $\text{Conv}(\phi(v_1), \ldots, \phi(v_n)) = \phi(P)$.

□

Theorem 2.20
Let P be a k-polytope in $\mathbb{R}^d$ for some $k \leq d$. Let $\mathcal{T}$ be a triangulation of P. Let $\phi : \mathbb{R}^d \to \mathbb{R}^k$ be defined as in Definition 2.17. Then

$$\phi(\mathcal{T}) := \bigcup_{T \in \mathcal{T}} \phi(T)$$

is a triangulation of $\phi(P)$.

Proof. By Lemma 2.19, $\phi(P)$ is a polytope and thus admits a triangulation. Since simplices are polytopes themselves, we know that for any simplex $T \in \mathcal{T}$, $\phi(T)$ is a simplex in $\phi(P)$. To show that $\phi(\mathcal{T})$ is a triangulation of $\phi(P)$, we show that it satisfies the properties of Definition 2.6. We show properties (*T1*) and (*T2*) using that ϕ preserves convex combinations (Lemma 2.19). We show property (*T3*) using that ϕ is an isometry (Lemma 2.18).

We prove that $\phi(P) = \bigcup_{T \in \mathcal{T}} \phi(\mathcal{T})$ by showing inclusion in both ways.

Suppose that $y \in \phi(P)$. Then there is $x \in P$ such that $\phi(x) = y$ and there thus is $T \in \mathcal{T}$ such that $x \in T$. It now follows from Lemma 2.19 that $y \in \phi(T) \in \phi(\mathcal{T})$. Hence $\phi(P) \subseteq \bigcup_{T \in \mathcal{T}} \phi(\mathcal{T})$.

Suppose that $y \in \bigcup_{T \in \mathcal{T}} \phi(\mathcal{T})$. Then there is $T \in \mathcal{T}$ such that $y \in \phi(T)$. There exists $x \in T$ such that $\phi(x) = y$. Since $T \in P$ we have $x \in P \Rightarrow y \in \phi(P)$. Hence $\bigcup_{T \in \mathcal{T}} \phi(\mathcal{T}) \subseteq \phi(P)$.

We showed in the proof of Lemma 2.19 that ϕ maps the vertices of P to the vertices of $\phi(P)$. Hence,

$$\bigcup_{T \in \phi(\mathcal{T})} v(T) = \bigcup_{T \in \mathcal{T}} v(\phi(T)) = \bigcup_{T \in \mathcal{T}} \phi(v(T)) = \phi(v(P)) = v(\phi(P)).$$

To show ($T3$), let $\phi(T_1), \phi(T_2) \in \phi(\mathcal{T})$ with $\phi(T_1) \neq \phi(T_2)$. Note that ${T_1}^{\circ} \cap {T_2}^{\circ} = \emptyset$. We will show that $\phi(T_1)^{\circ} \cap \phi(T_2)^{\circ} = \emptyset$ by contradiction. Suppose $y \in \phi(T_1)^{\circ} \cap \phi(T_2)^{\circ}$.

Denote by $v_1, \ldots, v_{k+1}$ the vertices of T_1 and by $w_1, \ldots, w_{k+1}$ the vertices of T_2. Since $y \in \phi(T_1)^{\circ}$, there is

$$\alpha_1, \ldots, \alpha_{k+1} \text{ with } \alpha_i > 0 \text{ and } \sum_{i=1}^{k+1} \alpha_i = 1$$

such that

$$y = \sum_{i=1}^{k+1} \alpha_i \phi(v_i).$$

Similarly, since $y \in \phi(T_2)^{\circ}$, there is

$$\beta_1, \ldots, \beta_{k+1} \text{ with } \beta_i > 0 \text{ and } \sum_{i=1}^{k+1} \beta_i = 1$$

such that

$$y = \sum_{i=1}^{k+1} \beta_i \phi(v_i).$$

Define

$$x_1 = \sum_{i=1}^{k+1} \alpha_i v_i, \;\; x_2 = \sum_{i=1}^{k+1} \beta_i w_i.$$

Note that $x_1 \in {T_1}^{\circ}$ and $x_2 \in {T_2}^{\circ}$. As ϕ preserves convex combinations, we have $y = \phi(x_1) = \phi(x_2)$. By Lemma 2.18, ϕ is an isometry. Hence ϕ is injective and $x_1 = x_2$. This means that $x_1 = x_2 \in {T_1}^{\circ} \cap {T_2}^{\circ}$ which is a contradiction. We conclude that indeed $\phi(T_1)^{\circ} \cap \phi(T_2)^{\circ} = \emptyset$.

We have shown that $\phi(\mathcal{T})$ is a finite collection of $k-$simplices in $\mathbb{R}^k$ that satisfy the properties of Definition 2.6. We conclude that $\phi(\mathcal{T})$ is indeed a triangulation of $\phi(P)$. □

We have shown that ϕ as in Definition 2.17 is an isometry to a full-dimensional polytope that preserves triangulations. Note that ϕ as a function to the transformed polytope is surjective (and hence bijective) and is thus invertible. We can thus switch back and forth between P and $\phi(P)$. This enables us to triangulate k-simplices P in $\mathbb{R}^d$ for $k < d$. We first transform P to full-dimensional $\phi(P)$. We can then use a triangulation algorithm such as Algorithm 3 to find a triangulation $\phi(\mathcal{T})$ of $\phi(P)$. Next, we transform the triangulation back to the original space to find $\mathcal{T}$ a triangulation of P.

Lemma 2.21
Let $P \subset \mathbb{R}^d$ with $dim(P) = k < d$. Let $\phi : \mathbb{R}^d \to \mathbb{R}^k$ be defined as in Definition 2.17. Let $T \subseteq \phi(P)$ be a k-simplex. Then $f^{-1}(T)$ is a k-simplex.

Proof. T is a k-simplex. So there are unique vertices $t_0, \ldots, t_k \in \mathbb{R}^k$ such that $T = \text{Conv}(t_0, \ldots, t_k)$. Moreover, for all $x \in T$ there is unique $\lambda_0, \ldots, \lambda_k \in \mathbb{R}$ such that $x = \sum_{i=0}^{k} \lambda_i t_i$.

We know that ϕ is a bijective isometry that preserves convex combinations. Hence $f^{-1}(T)$ is a polytope in $\mathbb{R}^d$ with vertices $f^{-1}(t_0) \ldots, f^{-1}(t_k) = v_0, \ldots, v_k$. What remains

for us to prove, is that $v_0, \ldots, v_k$ together span k dimensions. we know that $k+1$ points together span at most k dimensions. Suppose $v_0, \ldots, v_k$ span $m < k$ dimensions. So there is linear dependency in the vectors $v_1 - v_0, \ldots, v_k - v_0$. Hence there is $x \in \text{Conv}(v_0, \ldots, v_k)$ such that we can write x as a convex combination of $v_0, \ldots, v_k$ in two distinct ways, i.e.,

$$\text{there are } \alpha_0, \ldots, \alpha_k \in \mathbb{R}, \text{ and } \beta_0, \ldots, \beta_k \in \mathbb{R} \text{ with } (\alpha_0, \ldots, \alpha_k) \neq (\beta_0, \ldots, \beta_k),$$

such that

$$\sum_{i=0}^{k} \alpha_i v_i = x = \sum_{i=0}^{k} \beta_i v_i.$$

We then see that also

$$\sum_{i=0}^{k} \alpha_i \phi(v_i) = \phi(x) = \sum_{i=0}^{k} \beta_i \phi(v_i),$$

since ϕ is injective and preserves convex combinations. This is a contradiction. Hence $v_0, \ldots, v_k$ span exactly k dimensions. We conclude that $f^{-1}(T) = \text{Conv}(v_0, \ldots, v_k)$ is a k-simplex. □

2.5 Algorithm Performance

In this section we evaluate Algorithm 3 from Section 2.3. We discuss the algorithm's complexity and the effects of our assumptions on the input. We also put the algorithm to work in the setting of portfolio optimization to test its computation time and compare it to the Qhull triangulation algorithm.

2.5.1 Time complexity

Unfortunately Algorithm 3 is not time efficient. The complexity of triangulating polytope P depends on P's number of vertices n and its affine dimension d. In every iteration i of Algorithm 3, we loop over the number of facets of the current polytope P_i. As is shown in [14, Chapter 17], an upper bound for the number of facets of a d-polytope with n vertices is of the order $O\binom{n}{d/2}$. The number of facets of intermediate polytope P_i is of great influence on the iteration's computation time; when the input size is large, looping over the number of facets to check whether they are separating or not accounts for most of the effort. Moreover, the number of simplices necessary to triangulate a polytope may thus grow very quickly with the input size. The worst case time complexity of Algorithm 3 is subsequently of order $O\binom{n}{d/2}$.

It is important to point out that there exist many different classes of polytopes. The type of polytope we are dealing with may be of great influence on the number of facets and thus the computation time of the algorithm. Different polytope classes can have different bounds for the number of facets in the boundary and the number of simplices in a triangulation. For instance, so called *cyclic polytopes* are among the polytopes with the largest numbers of facets, reaching the upper bound of order $O\binom{n}{d/2}$. While this class provides an upper bound for the number of facets of a d-polytope with n vertices, there are many polytopes that have a much smaller number of facets. In particular, a lower bound for the number of facets in a *simplicial polytope* is $n(d-1)-(d+1)(d-2)$ as was shown by Barnette in [2]. Worst case complexity may not be the most practical measure when talking about the computation time of a triangulation algorithm as expected time

complexity is often many times smaller. For more on different polytope classes, we refer the reader to [14, Chapter 15].

The large variation in the number of facets of polytopes also brings about that the relation between the input size and the algorithm's complexity may not always be evident. As is pointed out in [14, Section 26.3.1], this leads to a serious shortcoming for incremental insertion algorithms, such as the Beneath-and-Beyond algorithm. It may very well happen that an intermediate polytope P_i in iteration i of the algorithm has a much larger number of facets than the final polytope $P = P_n$. Thus, the order of vertex insertion may make big difference in computation time.

2.5.2 Comments and improvements

There are a few easy improvements that can be made on the triangulation algorithm as described in Algorithm 3. Most importantly, updating the boundary can be done more efficiently by considering the previous boundary as input also. This results in a significant reduction of computation time. We chose not to include it in the description of the algorithms above for the sake of simplicity. We do however include it in the version we use for testing the algorithm performance in Section 2.5.3. Furthermore, there are several different ways to store a triangulation. A representation of simplices and facets in graphs may improve space complexity. We did not use graph representation in this research. Other improvements that can be made include optimizing the order of vertex insertion or adaptation of the algorithm to the polytope class that is dealt with.

At the start of Section 2.3, we made a few assumptions on the input on top of the ones the Beneath-and-Beyond algorithm generally assumes. The most important among these is the assumption that all input points are vertices of the final polytope P. These assumptions mainly improve the algorithm's simplicity. They do not significantly improve its worst-case time complexity, although in practice we see that the algorithm does terminate faster. Note that these assumptions do not apply in general, but are focused on our specific application of the triangulation algorithm for portfolio optimization.

2.5.3 Test results

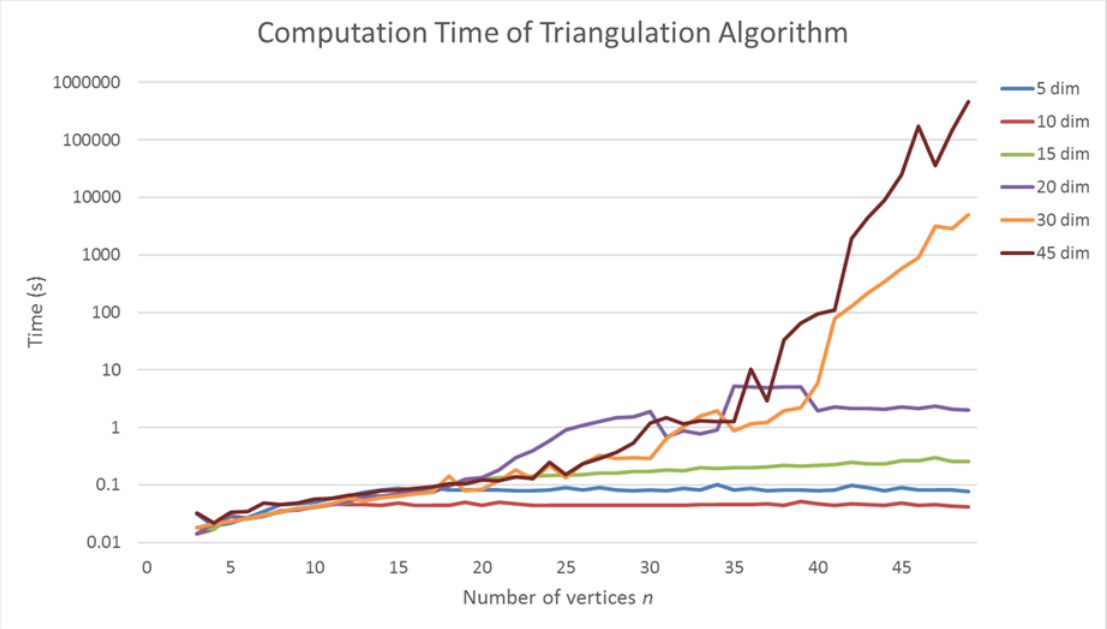
We have tested the algorithm performance in a portfolio setting. In doing so, we focused on the influence of the polytope dimension d (the number of asset classes) and of the number of vertices n. We compare the time it took our algorithm to triangulate polytopes of 5, 10, 15, 20, 30, and 45 dimensions respectively. For every polytope we varied the number of vertices to be included, ranging from 3 to 50 vertices. We show the results in a graph in Figure 7.

Looking at Figure 7, we see that the dimension as well as the number of vertices of the polytope have a clear effect on the computation time of our algorithm. Polytopes of higher dimensions take a considerably longer time to be triangulated. Overall, computation time seems to grow with the number of vertices in the polytope.

The influence of the number of vertices is especially clear in the higher dimensional cases ($d \geq 20$) of our tests. Computation time appears to grow more than exponentially with the number of vertices. For even higher dimensions, computation times soon are so large that testing is no longer feasible. Lastly, We observe that there is not much difference in computation times for this example when dimensions are relatively small ($d \leq 15$).

Note that the computation times for the polytopes in Figure 7 are very alike when the number of vertices is below twenty. This makes sense, since the affine dimension

Figure 7: Computation times of Algorithm 3 for polytopes of several dimensions with a varying number of vertices. Note that the time axis has a logarithmic scale.



of the polytopes is similar when the polytopes have little vertices. Computation times start growing quickly when the affine dimensions start growing.

We conclude that our implementation of the Beneath-and-Beyond algorithm as a triangulation algorithm performs better than Qhull's triangulation algorithm in the setting of portfolio optimization. Our algorithm is able to triangulate polytopes with up to 50 vertices in up to around 30 dimensions within a reasonable amount of time. Qhull's triangulation algorithm is efficient for large point sets in up to eight dimensions, but is not able to triangulate point sets of higher dimensions. We will use our implementation of the Beneath-and-Beyond algorithm to extend theory on near-optimal portfolio regions in the following chapters. In particular, it will be used to triangulate near-optimal region estimates, increasing knowledge of their structure.

3 Robustness

The main motivation for computing a near-optimal region in portfolio optimization is to increase optimization robustness. Robustness of a portfolio optimization may be defined by consistent output when input is subject to small changes. In this chapter we explore and test robustness within the near-optimal region. To do so, we need a way to sample uniformly from polytopes, which we discuss in Section 3.1. We test robustness of a near-optimal region in Section 3.2. In Section 3.3, we look for a single most robust portfolio in the near-optimal region. Finally, in Section 3.4, we discuss the concept of *diversification* within the near-optimal region.

3.1 Sampling from polytopes

We will use uniform sampling from a given polytope to test near-optimal regions' robustness. Furthermore, the difference between one polytope P and another P' may be estimated using uniform samples by drawing a sample S from P and calculating the average distance of the sample points in S to P'. In this section we explore sampling from polytopes. In Section 3.1.1, we present and test three different sampling methods and compare them in terms of efficiency. One of the methods requires being able to choose uniformly random directions in $\mathbb{R}^d$. We show how to do this in Section 3.1.2.

3.1.1 Polytope sampling methods

Let $P = \text{Conv}(v_1, \ldots, v_n) \subset \mathbb{R}^d$ be a $d-$polytope. We want to draw a random sample from P. To be precise, we want to randomly draw points from a uniform distribution over P. It is not straightforward how to uniformly draw a random sample from a d-polytope for $d \geq 3$. A naive approach might be to uniformly draw weights for the polytope vertices and take the resulting sum as a sample point, as was done in [16]. Note however that areas close to nested vertices will then be overrepresented in the sample. We present three methods to sample uniformly and discuss their (dis)advantages. In the following, let a d-polytope P and vertices $v_1, \ldots, v_n$ in $\mathbb{R}^d$ be given.

Accept/reject sampling

We may not be able to directly sample uniformly from any convex region, but we can directly sample uniformly from regular polytopes[4]. The idea of accept/reject sampling is to sample from a regular polytope that contains P and to keep the sample points lying inside P only, discarding the points lying outside P. The most simple way to do this starts with constructing a hypercube $H \subset \mathbb{R}^d$ that contains P. Let $w_1, \ldots, w_{2^d}$ be the vertices of H. We can then draw a point x uniformly from H by randomly choosing weights $\lambda_1, \ldots, \lambda_{2^d} \geq 0$ such that $\sum_i \lambda_i = 1$ by letting $x = \sum_i \lambda_i w_i$.

For sampling from polytope P estimating a near-optimal regions OPT, efficiency can be improved by sampling from a simplex containing P instead of a hypercube; as all portfolio weights are non-negative and together sum to one, we know that for a near-optimal region OPT, we have

$$OPT \subseteq \text{Conv}(e_1, \ldots, e_d), \text{ and hence } P \subseteq \text{Conv}(e_1, \ldots, e_d),$$

where e_i is the unit vector in the i-th coordinate.

[4]Regular polytopes are polytopes with the highest degree of symmetry, both reflectional and rotational. Every face of a regular polytope is again regular. Two- and three-dimensional examples are the square, regular pentagon, cube, and regular octahedron.

This method can be useful, especially in low dimensions. It is very easy to implement and its simplicity makes it easy to understand. A great disadvantage of this approach is that the probability of drawing a valid point in P decreases very quickly as d increases. In fact, for most shapes of polytope classes, the probability of drawing a valid point goes to zero as $d \to \infty$. As a result, it will take a lot of time to compute a sample for high-dimensional P.

Triangulation method

Another way to sample uniformly from P makes use of a triangulation of P. Given a triangulation $\mathcal{T}$ of P, we randomly draw a simplex T from $\mathcal{T}$ with probability proportional to the relative volume of T. Then, we draw a point uniformly from T by uniformly drawing weights for the vertices of T.

When a triangulation is given, this method is efficient and easy to implement. However, as we mentioned in Section 2.4, the construction of a triangulation may be a computationally intensive process.

Approximation of a uniform distribution using a Hit-and-Run method

This method generates a sample that is not exactly uniformly distributed, but does converge quickly to a uniform distribution as Smith shows in [39]. It is a process that generates sample points iteratively. The process is described below in Algorithm 4 and illustrated in Figure 8.

Algorithm 4: Hit-and-Run sampling

Data: Full-dimensional d-polytope P and some point $z_0 \in P$

Result: Sample S of size n from P that follows a distribution that converges to the uniform distribution as $n \to \infty$

for $i = 1, \ldots, n$ **do**
- Pick a uniformly random direction v in $\mathbb{R}^d$
- Let $\theta_{\min} = \min\{\theta \in \mathbb{R} : z_{i-1} + \theta v \in P\}$ and $\theta_{\max} = \max\{\theta \in \mathbb{R} : z_{i-1} + \theta v \in P\}$.
- Pick $\theta \sim \text{unif}(\theta_{\min}, \theta_{\max})$
- Let $z_i = z_{i-1} + \theta v$ and add z_i to S.

end

The main advantage of the hit-and-run method is that it is efficient in high dimensions. A disadvantage is that small samples may be biased towards the area in which the starting point z_0 lies. Note that the implementation of this method is not straightforward. Firstly, most programming languages do not provide a feature to directly choose a uniformly random direction in $\mathbb{R}^d$. It can however be done using standard normal random variables as we will show in section 3.1.2. Secondly, to be able to approximate $\theta_{\min}$ and $\theta_{\max}$, we need to have some way of quickly validating whether a point lies inside P or not. When we know P's $\mathcal{H}$-representation, validation can be done by checking all constraints that define P.

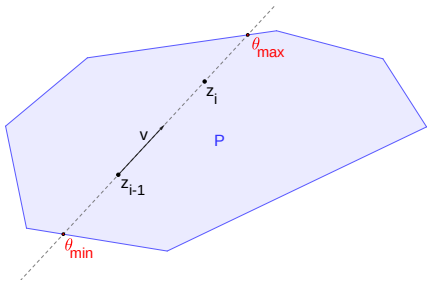
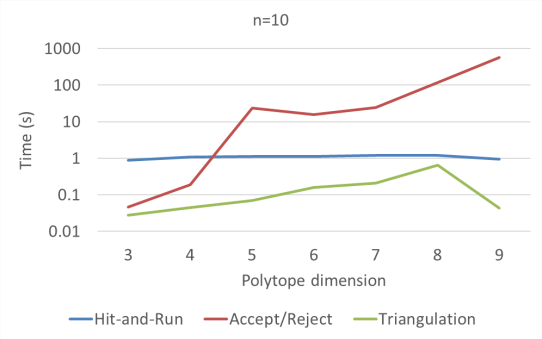


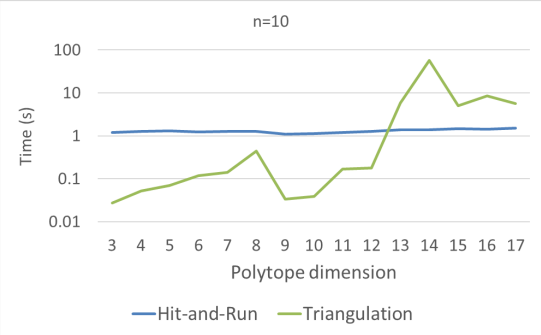
Figure 8: Iteration of Hit-and-Run algorithm for sampling from a 2-dimensional polytope

We test and compare the three sampling methods for polytopes of several dimensions. The results are shown in Figures 9a-9d. We see that there are different drivers for the computation times of the different sampling methods.

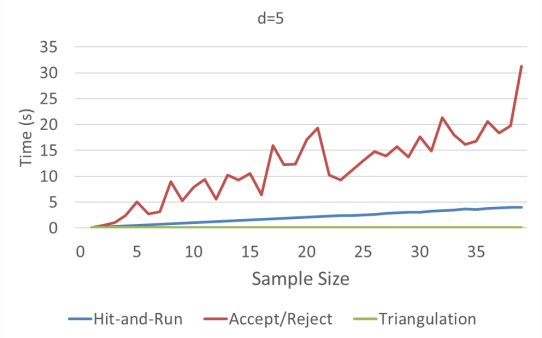
We see that for the Hit-and-Run method, the time it takes to sample from a polytope does not depend on the polytope's dimension. It does however grow linearly with the sample size. For the triangulation method, sampling time grows quickly with the polytope's dimension, as can be seen in Figure 9a. It however does not depend on the sampling size. This makes sense, as once a triangulation is computed, new points can be sampled very efficiently. Computation time for the accept/reject sampling method seems to grow exponentially with the polytope dimension, as we see in Figure 9b. Moreover, its expected computation time grows linearly with the sample size.



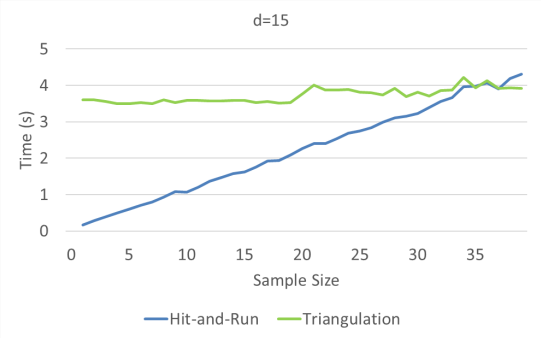
(a) Computation time for the three sampling methods for polytopes of dimensions smaller than ten.



(b) Computation times of sampling from polytopes of dimension under twenty. Accept/reject sampling is left out of the analysis due to high computation times.



(c) Computation times of sampling from polytopes of dimension 5 against the sample size.



(d) Computation times of sampling from polytopes of dimension 15 against the sample size. Accept/reject sampling is left out of the analysis due to high computation times.

Figure 9: In Figures 9a and 9b, uniformly random samples of size ten are drawn from polytopes of different dimensions using the three sampling methods described in Section 3.1.1. In Figures 9c and 9d, unformly random samples of different size are drawn from polytopes of dimension five and fifteen respectively. All displayed values are averages over five measurements. The polytopes were generated from the example of 133 asset classes, where only a restricted number of asset classes were included.

When comparing the above three methods, we conclude that for sampling from high-dimensional polytopes, the hit-and-run approach is favourable due to its superior computation time. When drawing large samples from low-dimensional polytopes ($d \leq 12$), the triangulation method may be more efficient. We do not recommend the accept/reject method.

3.1.2 Randomly choosing a uniform direction in a general dimension

The Hit-and-Run algorithm (Algorithm 4) makes use of uniformly random directions in $\mathbb{R}^d$. We mentioned that implementing such a random direction is not straightforward. We show how to choose a uniformly random direction in $\mathbb{R}^d$ using the multivariate normal distribution. We note that choosing a uniformly random direction in $\mathbb{R}^d$ is equivalent to choosing a uniformly random point on the d-sphere. We will show how to do this in Theorem 3.3.

We state Lemma 3.1 without proof as it is a standard result from probability theory. The interested reader can find more information and a proof in [10, Chapter 24].

Lemma 3.1
Let $(X_1, \ldots, X_n)$ be a normally distributed random vector with mean $\vec{\mu}$ and covariance matrix $\Sigma \in \mathbb{R}^{n \times n}$. Let $A : \mathbb{R}^n \to \mathbb{R}^n$ be a linear mapping. Let

$$\begin{pmatrix} Y_1 \\ \vdots \\ Y_n \end{pmatrix} = A \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix}.$$

Then $(Y_1, \ldots, Y_n)$ is normally distributed with mean $A\vec{\mu}$ and covariance matrix $A\Sigma A^\intercal$.

Corollary 3.2
Let $A \in \mathbb{R}^{n \times n}$ be a rotation map in $\mathbb{R}^n$ and let $X_1, \ldots, X_n$ be standard normal random variables. Then $A(X_1, \ldots, X_n)^\intercal$ is standard normal.

Proof. The mean $A\vec{\mu} = 0$ follows directly. Since A is a rotation, we know that $A \cdot A^\intercal = I$. Since $X_1, \ldots, X_n$ are independent and have unit variance, we have $\Sigma = I$. Hence,

$$A\Sigma A^T = A \cdot A^\intercal = I.$$

□

Corollary 3.2 tells us that the standard (multivariate) normal distribution is rotation indifferent.

Theorem 3.3
Let $X_1, \ldots, X_n$ be independent standard normal distributed random variables. Define

$$Y = (Y_1, \ldots, Y_n) \quad \textit{where} \quad Y_i = \frac{X_i}{\|(X_1, \ldots, X_n)\|}.$$

Then Y is uniformly distributed on the n-sphere $S^n = \{x \in \mathbb{R}^n : \|x\| = 1\}$.

Proof. Let B_1 and B_2 be two open balls on the surface of S^n with the same radius and define $C_i = \{x \in \mathbb{R}^n : \frac{x}{\|x\|} \in B_i\}$ for $i = 1, 2$. Note that there is a rotation $A \in \mathbb{R}^{n \times n}$ such that $A \cdot B_1 = B_2$. Note that then also $A \cdot C_1 = C_2$. Let f_X denote the density function of random variable X. Using Corollary 3.2 we see that $f_X(C_1) = f_X(C_2)$ and hence $f_Y(B_1) = f_Y(B_2)$.

We have found that two open balls in S^n of the same volume have the same density. Note that the open balls in S^n form a basis of S^n. This allows us to generalize the statement that two areas in S^n of the same volume have the same density. We conclude that Y is indeed uniformly distributed on S^n. □

3.2 Robustness of the Near-Optimal Region

One of the goals of introducing the near-optimal portfolio approach, was to gain more robustness in the optimization result. As was posed in Section 1.1, some issues with general portfolio optimization were that the estimation of means and covariance matrices is uncertain, while the optimization objective is very sensitive to small changes in these estimates. We define robustness here as producing consistent optimization output when the estimated means and covariances are varied with small perturbations. The natural way to measure the distance between two portfolios is by calculating their *turnover*, defined in Definition 3.4.

Definition 3.4 (Turnover)**.** Let $w, w' \in \mathbb{R}^d$ be two portfolios, i.e.,

$$\sum_{i=1}^{d} w_i = \sum_{i=1}^{d} w'_i = 1 \text{ and } w_i, w'_i \geq 0 \text{ for all } i \in [d].$$

We define the *turnover between* w *and* w' as

$$\frac{1}{2} \sum_{i=1}^{d} |w_i - w'_i|.$$

The turnover is used by investors, because it tells which percentage of his wealth an investor has to move to change from his current portfolio w to a new portfolio w'.

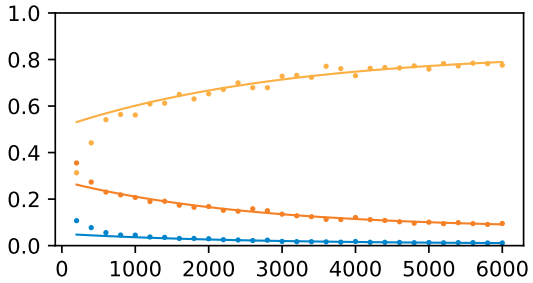


Figure 10: Figure 10 shows, as a function of sample size N, the average overlap between the original and perturbed convex hull (yellow), the average turnover between the original and perturbed mean-variance optimal portfolio (orange) and the average turnover between portfolios in the original convex hull and the closest portfolio in the perturbed convex hull (blue). To perturb the optimization's input, we draw a sample of size N from anormal distribution with mean and covariance as in Table 1 and estimate a perturbed mean and covariance matrix. Hence, the larger the sample size, the closer the perturbed means and covariances are to the original ones.

Figure 10, by De Graaf and Van der Schans in [16], shows that near-optimal portfolios are very likely to still be near-optimal after perturbation of the optimization's input parameters, with an average turnover approaching 0%. It furthermore shows that the difference between the traditional mean-variance optimal portfolio before and after input perturbation is on average a turnover of over 10%. This indicates that the near-optimal region is significantly more robust than the traditional optimization result. We note that the optimization result is now a region instead of a single portfolio. It should not be surprising that a region is more robust than a single point. In Section 3.3, we will however show that we can find a single portfolio inside the near-optimal region that is more robust than the mean-variance optimal portfolio also.

3.3 Centre of Mass

When discussing the robustness of a polytope P, it seems natural to look for a single portfolio in P that is most robust. Intuitively, a good candidate for a robust point within P may the point that lies exactly in the middle of P. In a convex region, this 'middle point' coincides with the region's *centre of mass*. In this section we specify how to find a polytope's centre of mass and we analyze its robustness.

The centre of mass of P is defined as the unique point where all mass of P is balanced. That is, any linear halfspace through the centre of mass will divide P into two parts of equal mass. In the following, we assume a continuous distribution of mass over a solid volume of P. We define the centre of mass using the definition by Beatty [5, Section 5.2.3].

Definition 3.5 (Centre of Mass)**.** The *centre of mass* $m(P)$ of a polytope $P \subset \mathbb{R}^d$ with a continuous distribution with density ρ over a solid volume V is given by

$$m(P) = a + \frac{1}{\text{vol}(V^* - a)} \int_P r\rho(r) d(V^* - a)$$

where V^* is an affine subspace generated by V such that P is full-dimensional in V^*. Furthermore, $a \in \mathbb{R}^{\dim(V^*)}$ is such that after translating over a, $(V^* - a)$ is a linear subspace (i.e. $V^* - a$ contains the zero vector).

In the case of a full-dimensional polytope P, Definition 3.5 reduces to

$$m(P) = \frac{1}{V} \int_P r\rho(r) dV.$$

For simplicity, we assume in the remainder of this section that we only deal with full-dimensional polytopes. In Section 2.4 we showed how to deal with lower-dimensional polytopes so that we can calculate with full-dimensional polytopes instead.

As can be seen from the definition, the computation of the centre of mass for a general object in $\mathbb{R}^d$ is not straightforward. The computation of the integral in Definition 3.5 can be involved. In order to avoid that, the centre of mass may be determined from a triangulation $\mathcal{T}$ of P as defined in Definition 2.6. The centre of mass of P is then calculated as the average of the centres of mass of the simplices of $\mathcal{T}$, weighted according to the relative simplex volumes, as we show in Theorem 3.6.

Theorem 3.6
Let P be a full-dimensional polytope with triangulation $\mathcal{T}$. For every simplex T in $\mathcal{T}$, we denote by V_T the volume of T. We then have equality:

$$m(P) = \sum_{T \in \mathcal{T}} \frac{V_T}{V} m(T), \tag{16}$$

where $m(\cdot)$ denotes the centre of mass.

Proof. We use that the interiors of the simplices in the triangulation are disjoint and that the triangulation is finite. We thus have

$$\begin{aligned} \sum_{T \in \mathcal{T}} \frac{V_T}{V} m(T) = \sum_{T \in \mathcal{T}} \frac{V_T}{V} \frac{1}{V_T} \int_T \rho(r) \cdot r dV_T &\overset{*}{=} \frac{1}{V} \sum_{T \in \mathcal{T}} \int_T \rho(r) \cdot r dV \\ &\overset{**}{=} \frac{1}{V} \int_{\cup_{T \in \mathcal{T}}} \rho(r) \cdot r dV = m(P). \end{aligned}$$

For equalities $(*) and (**)$ we used that V and all simplices in the triangulation are full-dimensional and that the interiors of all simplices are disjoint (property *(T3)* of Definition 2.6).

□

There are explicit formulas for the centre of mass and the volume of a simplex. Therefore, Theorem 3.6 implies that the computation of an object's centre of mass is straightforward when its triangulation is known. Proposition 3.7 gives us the necessary details.

Proposition 3.7
Let $T \subset \mathbb{R}^d$ be a d-simplex with vertices $v_0, \ldots, v_d$. The volume V_T and the centre of mass $m(T)$ of T are given by

$$V_T = \left| \frac{1}{d!} \det(v_1 - v_0, v_2 - v_0, \ldots, v_d - v_0) \right|,$$
$$m(T) = \frac{1}{d+1} \sum_{i=0}^{d} v_i,$$

where we denote by $(v_1 - v_0, v_2 - v_0, \ldots, v_d - v_0)$ the matrix with vectors $v_i - v_0$ in the columns, $i = 1, \ldots, d$.

The results of Proposition 3.7 are standard results from linear algebra and we omit the proofs. We are able determine the centre of mass of any full-dimensional polytope in $\mathbb{R}^d$ relatively easily when a triangulation is given. The same results can be applied for polytopes that are not full-dimensional. We show that the methods from Section 2.4 also apply here and that the transformation defined in Definition 2.17 preserves the centre of mass.

Theorem 3.8
Let transformation ϕ be defined by as in Definition 2.17. Then for any convex polytope P with vertices $v_0, \ldots, v_n$ in $\mathbb{R}^d$, the transformed centre of mass of P equals the centre of mass of the transformed polytope $\phi(P) = \text{Conv}(\phi(v_0), \ldots, \phi(v_{n-1}))$:

$$\phi\big(m(\{v_0, \ldots, v_{n-1}\})\big) = m\big(\{\phi(v_0), \ldots, \phi(v_{n-1})\}\big). \tag{17}$$

Proof. First we adopt some notations.
Let P be a convex polytope in $\mathbb{R}^d$ with vertex points $v_0, \ldots, v_n$ and let $P^* = \phi(P)$. Then, by Lemma 2.20, P^* has vertex points $\phi(v_0), \ldots, \phi(v_n)$ for all $0 \leq i \leq n$. Let m denote the centre of mass of P and m^* the centre of mass of P^*. Let $\mathcal{T}$ be a triangulation of P. By Lemma 2.20, $\mathcal{T}^* = \phi(\mathcal{T})$ is a triangulation of P^*.

Using Proposition 3.6, we write

$$\begin{aligned} m^* &= \sum_{T \in \mathcal{T}^*} \frac{V_T}{V} z(T) = \sum_{T \in \mathcal{T}} \frac{V_T}{V} \sum_{i: v_i \in T} \frac{1}{k+1} \phi(v_i) \\ &= f\left(\sum_{T \in \mathcal{T}} \frac{V_T}{V} \frac{1}{k+1} \sum_{i: v_i \in T} v_i \right) = \phi(m), \end{aligned}$$

where we used the preservation of convex combinations by ϕ (Lemma 2.19) twice.

□

It should be noted that in the above, we use that the convex hull of $\{v_0, \ldots, v_{n-1}\}$ in $\mathbb{R}^d$ can be embedded in $\mathbb{R}_k$. We hence see that each simplex in the triangulation of $\text{Conv}(v_0, \ldots, v_{n-1})$ has $k+1$ vertices.

We now analyze the robustness of the centre of mass of a near-optimal region's estimation. We again use the example from Section 1.2. We assess robustness by looking at the distance in terms of turnover of results before and after perturbation. We compare robustness of the centre of mass to robustness of the centroid (see Definition 1.5), which is the average of all vertices, and the mean-variance optimal portfolio. The

results are displayed in Figure 11 with the sample size on the horizontal axis and the average turnover on the vertical axis.

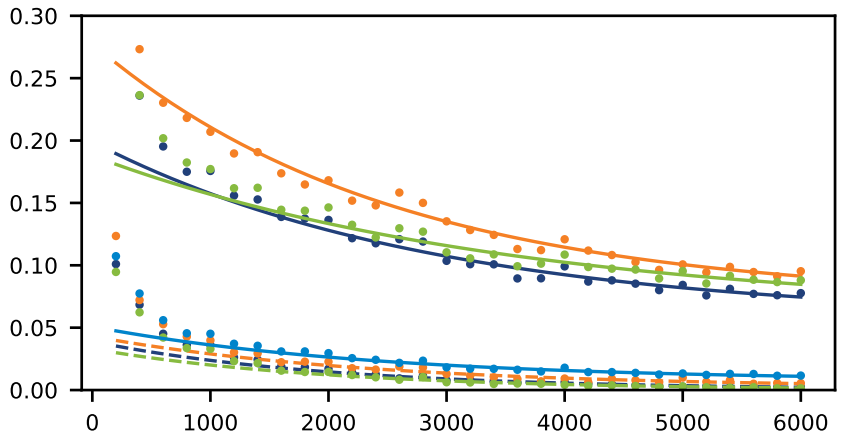


Figure 11: This graph aims to measure and compare the robustness of several portfolios of interest: the mean-variance optimal portfolio (orange), the centroid portfolio (green), and the centre of mass portfolio (dark blue). The solid lines display the average turnover between original values and values of the perturbed input against sample size N. The dotted lines display the average turnover between the original values and the closest portfolio in the perturbed near-optimal region against sample size N. The light blue line displays the average turnover between the original near-optimal region and the perturbed near-optimal regions. The input parameters are perturbed in the same way as for Figure 10.

Looking at the dotted lines in Figure 11, we see that in most cases, the three special portfolios remain near-optimal after the input is perturbed. The light blue line even shows that the near-optimal region as a whole remains very much the same even though the input is perturbed. This effect is visible stronger in this research than it is in [16]. This may be explained by the way we sample. In our results, we made sure that sampling was done uniformly. In [16], samples showed bias towards the centroid, possibly explaining a higher average turnover between the near-optimal regions.

Looking at the solid lines in Figure 11, we see that when measuring the changes one of the special portfolios after perturbing the input, the centre of mass is more robust than the centroid and the mean-variance optimal portfolio. Especially when sample size increases this is clearly visible.

We can thus conclude that for this example, the near-optimal region is more robust than the traditional mean-variance optimal portfolio. Furthermore, the centre of mass portfolio is more robust than the centroid portfolio and the mean-variance optimal portfolio. When measuring with respect to the near-optimal region however, all three portfolios are similar in terms of robustness. This means that if the goal is to still be near-optimal after perturbation of the input parameters, the Mean-Variance optimal portfolio may be sufficient.

3.4 Diversification

Next to robustness, the near-optimal approach brings a different advantage for investors. Additional arguments that were not included in the optimization may be used to select a preferred portfolio from a more limited set with sufficient optimality. Some arguments, such as portfolio liquidity, expert opinion, or performance in environmental sustainability, may be difficult to quantify. Others, such as transaction costs, may play a role but

are usually left out of the optimization nonetheless. One recognized important feature of portfolios that is usually not included in portfolio optimization beyond variance-analysis, is *diversification*. In this section, we explore the concept of diversification and the available measures to quantify it. We furthermore look at diversification within the near-optimal portfolio region.

The literature does not agree on a single measure for portfolio diversification. Portfolio diversification is generally based on reducing risk on a portfolio by spreading its allocation over different asset classes. As mentioned by Pola in [31], the literature explores three general methods of measuring portfolio diversification. The first is based on asset weights only. The second is based on a portfolio's risk contributions. The third method is based on economic scenarios. We discuss several measures based on the first two methods. Economic scenario analysis is not considered here. We note that the way diversification should be measured for a great part relies on context. When asset classes are uncorrelated and almost equal in terms of risk, measuring assets weights seems appropriate. When no sufficient information on assets classes' risk is available, it may even be the only option. When asset classes are of different risks and we have an estimation of their covariances, we can base a diversification measure on risk contributions. We discuss four different measures and try to make them intuitive. All but the first are based on the information-theoretic concept of *entropy*.

The *Herfindahl-Hirschman Index* (HHI) is the only measure we discuss that is based on asset weights only. The index is widely used in areas beyond portfolio management and measures concentration levels. Woerheide and Persson use the HHI in [30] to measure portfolio diversification. It is defined as

Definition 3.9 (Herfindahl-Hirschman Index)**.** Let $w \in \mathbb{R}^d$ be a vector of portfolio weights for d asset classes. The *Herfindahl-Hirschman Index* (HHI) of w is given by

$$HHI(w) = \sum_{i=1}^{d} w_i^2.$$

We note that $HHI(w)$ is minimal for the equally weighted portfolio $w = (\frac{1}{d}, \ldots, \frac{1}{d})$.

The next three measures we define are based on the information-theoretic concept of *entropy*. We define a general entropy measure by

$$-\sum_i p_i \log(p_i). \tag{18}$$

The following three diversification measures differ in how they define p_i in (18).

The second measure we discuss is an *entropy measure of volatility*, as proposed by Meucci in [27]. Suppose we have an estimated covariance matrix Σ of the asset classes. We estimate the volatility of asset i by its variance $\sigma_i = \Sigma(i, i)$. We now define p_i for $i = 1, \ldots, d$ in definition 3.10.

Definition 3.10 (Volatility entropy measure)**.** Let $w \in \mathbb{R}^d$ be a vector of portfolio weights for d asset classes. Suppose asset class i has variance σ_i. The *volatility entropy measure* of w is given by

$$-\sum_i p_i \log(p_i),$$

where

$$p_i = \frac{\sigma_i^2 w_i^2}{\sum_{j=1}^d \sigma_j^2 w_j^2}.$$

The entropy measure of volatility assesses diversification in terms of risk of the individual asset classes. It does not take into account potential correlation between asset classes. According to this measure, the optimally diversified portfolio is the naive risk parity portfolio where asset weights are inversely proportional to the asset volatility.

The third measure we consider is an *entropy measure of risk contribution*, also proposed by Meucci in [27]. It takes into account correlation between asset classes by including the asset correlation matrix R with elements $\rho_{ij} = \frac{\Sigma(i,j)}{\sigma_i \sigma_j}$.

Definition 3.11 (Risk contribution entropy measure)**.** Let $w \in \mathbb{R}^d$ be a vector of portfolio weights for d asset classes. Suppose asset class i has variance σ_i and suppose that asset class i and j are correlated with Pearson correlation coefficient ρ_{ij}. The *risk contribution entropy measure* of w is given by

$$-\sum_i p_i \log(p_i),$$

where

$$p_i = \frac{\sum_{j=1}^d |w_i w_j \sigma_i \sigma_j \rho_{ij}|}{\sum_{k=1}^d \sum_{j=1}^d |w_k w_j \sigma_k \sigma_j \rho_{kj}|} = \frac{\sum_{j=1}^d |w_i w_j \Sigma(i,j)|}{\sum_{k=1}^d \sum_{j=1}^d |w_k w_j \Sigma(k,j)|}.$$

The optimally diversified portfolio according to this measure is the estimated risk parity portfolio.

The final measure we define uses Principal Component Analysis (PCA) to measure portfolio diversification. We follow the approach of Pola in [31]. Roughly, PCA looks at the components that explain most of the variance in the portfolio space and analyzes how much of these components contribute to a certain portfolio. We use the estimated asset covariance matrix Σ as input to evaluate risk. Below, we describe the approach of principal components analysis and explain how each step can be interpreted in the setting of portfolio optimization.

We start with estimated covariance matrix $\Sigma \in [-1,1]^{d \times d}$ of the asset classes. In a first step, we perform an eigenvalue decomposition,

$$\Sigma = E \Lambda E^\intercal.$$

Here, Λ is a diagonal matrix with the eigenvalues $\lambda_1, \ldots, \lambda_d$ of Σ in the diagonal. E is an orthogonal matrix, with the eigenvectors $e_1, \ldots, e_d$ of Σ in the columns. These eigenvectors are also called *principal components* or, in the setting of portfolio management, *principal portfolios*. The principal components can be interpreted as mutually uncorrelated risk drivers. Eigenvalue λ_i corresponds to eigenvector e_i and can be interpreted as a measure for the contribution of risk driver e_i to the total risk in the portfolio space. Without loss of generality, we can assume Λ and E ordered such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$. Note that all eigenvalues and eigenvectors in this eigenvalue decomposition will be real, since covariance matrices are positive-semidefinite.

Since λ_i describes the contribution of e_i to total risk, we can calculate the relative risk contribution R_i of risk driver e_i as

$$R_i = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j}$$

with $R_1 \geq \cdots \geq R_d$. We now select the principal components e_i that explain most of the risk by looking at their relative risk contribution R_i.

$$\tilde{E} = \{e_i \in E : R_i > \delta\} = \{e_1, \ldots, e_k\}$$

for some $\delta \geq 0$ and $k \in [d]$. $\tilde{E}$ consists of the risk drivers that are most important. The risk drivers $e_{k+1}, \ldots, e_d$ that are not in $\tilde{E}$ are henceforth left out. Their contribution to total risk is deemed too small to be relevant for further risk analysis.

Let $w \in \mathbb{R}^d$ denote the weights of the asset classes in a portfolio we want to analyze in terms of diversification. We want to see how w can be built up in terms of our relevant risk drivers $e_1, \ldots, e_k$. We thus project portfolio w into the principal components of $\tilde{E}$,

$$\tilde{w} = \tilde{E}^\intercal w.$$

$\tilde{w}$ tells us how much of each of the risk drivers is represented in portfolio w. We now find an estimate for the risk exposure h_i to risk driver e_i by

$$h_i = \tilde{w}_i^2 \lambda_i.$$

We can thus use PCA to estimate the exposure of a portfolio w to the relevant risk drivers in our portfolio space. Rudin and Morgan argue in [36] that the exposure to different risk drivers can be used to define a measure for diversification. They argue that a well diversified portfolio should be exposed to the different risk drivers evenly. They thus define relative exposure p_i to risk driver e_i by

$$p_i = \frac{h_i}{\sum_{j=1}^{k} h_j}.$$

The procedure is summarized in Definition 3.12.

Definition 3.12 (Principal components measure)**.** Let $w \in \mathbb{R}^d$ be a vector of portfolio weights for d asset classes. Suppose the asset classes have estimated covariance matrix Σ with eigenvalues $\lambda_1, \ldots, \lambda_d$ in descending order, and eigenvectors $e_1, \ldots, e_d \in \mathbb{R}^d$. Denote

$$R_i = \frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i}.$$

The *principal components entropy measure* of w for risk contribution coefficient $\delta \geq 0$ is given by

$$-\sum_i p_i \log(p_i),$$

where

$$p_i = \frac{(\tilde{E}^\intercal w)_i^2 \lambda_i}{\sum_j (\tilde{E}^\intercal w)_j^2 \lambda_j},$$

and $\tilde{E} = \{e_i : R_i > \delta\}$.

We test the different diversification measures on the example from Section 1.2. We are given a portfolio space with three asset classes: Stocks, Bonds, and Treasury Bills. For each of the above defined diversification measures, we calculate the optimally diversified portfolio in the near optimal region as well as the portfolio that is optimally

diversified globally. We show the results in Figure 12. We use the figure to interpret the diversification measures.

Figure 12

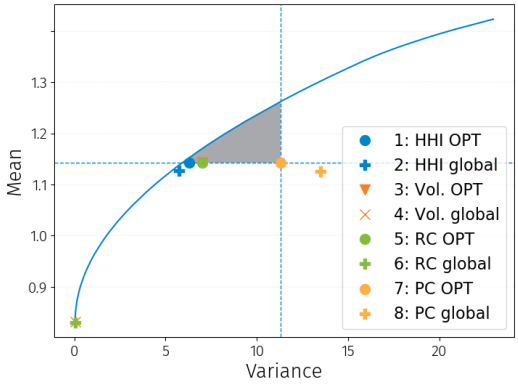


Table 3

Measure		*Allocation* Stocks	Bonds	T-bills
HHI	OPT	.360	.333	.307
	global	.333	.333	.333
Volatility	OPT	.306	.442	.252
	global	.002	.003	.995
Risk Contr	OPT	.306	.442	.252
	global	.002	.003	.995
PCA	OPT	.141	.771	.089
	global	.048	.901	.051

Figure 12 and Table 3 show optimally diversified portfolios according to different measures in a three-asset portfolio space. In Figure 12, we see the optimally diversified portfolios in the risk/return space. Table 3 shows the asset weights in the portfolios. For every diversification measure, we optimize twice. We look at the optimally diversified portfolio within the near-optimal region OPT, as well as at the optimally diversified portfolio in the global portfolio space.

We aim to clarify the different diversification measures by explaining the results in Figure 12.

The HHI measure (blue) evaluates the portfolio weights only and globally converges to the equally weighted portfolio. We see that the most diversified portfolio in the near optimal region is almost equally weighted as well. In this example, we see that estimated variance of the HHI portfolio is indeed very small.

The volatility measure (red) as well as the risk contribution measure (green) converge globally to the minimum variance portfolio. In this example, the optimized portfolio weights for both measures are the same. This causes the optimized volatility portfolios to disappear behind the green dots in the graph. The volatility entropy measure aims to minimize entropy in relative (uncorrelated) variance. In this example, Treasury Bills are estimated to have very little variance. Minimum entropy is thus found when exposure towards Treasury Bills is very large. Note however that the most diversified near-optimal portfolio according to the HHI (portfolio 1) is more exposed towards Treasury Bills than the most diversified near-optimal portfolio according to the volatility measure (portfolio 3). This is caused by their allocation to Stocks, which is the most risky asset in this example. We thus pose that the objective of the minimum volatility optimization problem in this example is to maximize exposure towards Treasury Bills and minimize exposure to Stocks. An explanation for the similarity between the volatility measure and the risk contribution measure may be that the different asset classes in this example are not very correlated. For both measures, we see that estimated risk in the near-optimal region is small.

The most diversified portfolio according to the measure using PCA shows different behaviour. When looking at the PCA, we see that the the method identifies two principal components that contribute significantly to total risk. The asset classes of Stocks and Bonds are almost fully built up out of these two components. The third principal

components falls more or less together with the asset class of Treasury Bills. Due to this asset's very low volatility, the associated principal component hardly contributes to the model's risk (less than 1%) and is considered irrelevant by the PCA method. The model thus balances risk contribution between the first two principal components, resulting in a large exposure towards Bonds and a small allocation to Stocks. Figure 13 shows that the asset of Treasury Bills is not represented by the selected principal components. It also shows that the projections of Stocks and Bonds on to the two principal components are nearly orthogonal. Together with the observation that the third principal components falls together with the Treasury Bills asset, this corresponds to our earlier conclusion that the asset classes are not very correlated. The ignorance of PCA towards exposure to the non-risky asset of Treasury Bills in this example leads to an optimally diversified portfolio that has relatively high estimated risk.

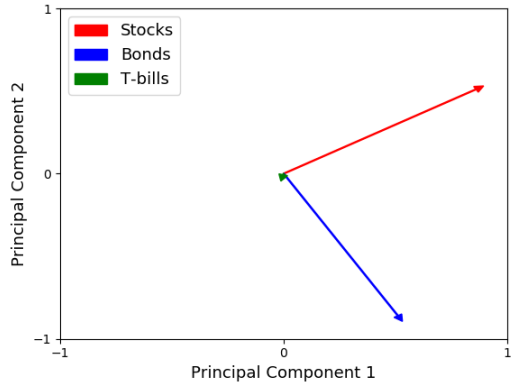


Figure 13: The asset classes of the example from Section 1.2 are projected on to the first two principal components from the performed PCA. Note that Treasury Bills are in this case virtually not represented by the first two principal components, making the model's estimated risk ignorant towards that asset class.

	PC 1	PC 2	PC 3
Risk Contr.	69.01%	30.87%	0.12%
Stocks	.860	.511	.005
Bonds	.511	-.860	-.005
T-bills	-.002	-.007	1.000

Table 4: Statistics of the Principal Components Analysis. The three components' relative risk contributions are displayed in the first row. Below we show the scores of the assets classes projected onto the components.

We note that the results on the different measures in this example do not lead to an overall performance indication of the measures. These results merely illustrate the measures' behaviours and drivers. We thus do not recommend a particular measure based on these results.

As a further comment on the different measures, we note once more that the setting of the optimization may play a large role as well. If no reliable estimations of covariance or correlations of the assets are available, all but the HHI measure are immediately infeasible. When we are however dealing with a set of classes with much correlations, one may consider the risk contribution or principal components measures more appropriate.

3.5 Conclusion

In this chapter we have shown how to sample uniformly from a polytope. We explored three different sampling methods: the accept/reject method, the triangulation method, and the Hit-and-Run method. We have found that the accept/reject method is very slow. The triangulation method is convenient for drawing large samples from polytopes that are easy to triangulate. When dealing with a polytope with a large dimension, the

triangulation method may however take a very long time. The Hit-and-Run method is suitable for polytopes with general dimension. We recommend using the Hit-and-Run method for sampling from general polytopes.

We next tested the robustness of a near-optimal region. We found that the near-optimal region is significantly more robust than the traditional Mean-Variance optimal portfolio. We showed how to compute a polytope's centre of mass in Section 3.5 and showed that a near-optimal region's centre of mass is more robust than the Mean-variance optimal portfolio.

We next explored how to assess diversification within the near-optimal region, presenting four different diversification measures. We tested these measures in an example and gained an understanding of their behaviour. We do not recommend one specific measure, as the measures' performances are dependent on the given context.

The centre of mass and the optimally diversified portfolio (with respect to one of the different diversification measures) are two examples of portfolios an investor might want to select after having found the region of near-optimal portfolios. The construction of both two special portfolios exemplify how an investor may select his portfolio from the near-optimal region. The centre of mass is constructed using knowledge from the near-optimal region's structure from its triangulation. The optimally diversified portfolio shows how additional arguments can be included in the final selection process.

4 Convex hull computation

In this chapter we focus on how to estimate a near-optimal region by a polytope. The method by De Graaf and Van der Schans from [15] and [16] is computationally very expensive. We aim to improve their method in terms of efficiency without losing too much on precision.

We first explain and analyze the existing method in Section 4.1. We improve on the method's efficiency in Section 4.2. In Section 4.3, we present a new method, inspired by the existing method, to estimate convex regions by polytopes. In Section 4.3.4, we show that our new method can be applied to mean-variance optimization. Section 4.3.5 presents an adapted version of the new method that may be used in practice.

4.1 Existing method

There are several ways to approximate a convex region by a polytope. De Graaf and Van der Schans developed an estimation technique in [15], which is focused on estimating a convex region in the setting of portfolio optimization. We discuss their method in detail in this section. Note that we try to define and analyze the methods for general risk and return measures as much as possible. When empirically testing estimation algorithms, we however always use the Mean-Variance setting to estimate a portfolio's risk and return. In Section 4.1.2, we describe and analyze the method as presented in [15] and [16] by De Graaf and Van der Schans in detail.

The literature proposes several other polytope computation algorithms. The Double Description Method, as described by Fukuda and Prodon in [13], allows to switch back and forth between a polytope's $\mathcal{V}$-representation and its $\mathcal{H}$-representation (see Section 2.1). Note that the Double Description method cannot readily be applied to estimate a near-optimal region, as a polytope's $\mathcal{H}$-representation only takes linear constraints, while risk- and return constraints may be non-linear. Other polytope construction algorithms may take a point set as input. For instance, Qhull [4] offers a functionality that computes the minimal polytope containing a set of input points. We do not consider these other polytope computation algorithms here. Though a further study may be useful, we choose to focus on the method by De Graaf and Van der Schans.

4.1.1 Theoretical setting

We present the setting in which we estimate a near-optimal region. Recall the definition of a near-optimal region OPT from Section 1.1. We repeat it with a slight modification in Definition 4.1.

Suppose we have a portfolio space consisting of $d \in \mathbb{N}$ asset classes. We define a near-optimal region OPT by linear equality constraints $\{Ax = b\}$, linear inequality constraints $\{Gx \leq h\}$, a convex constraint on the portfolio return $\{f_\mu(x) \leq C_\mu\}$ and a convex constraint on the portfolio risk $\{f_\Sigma(x) \leq C_\Sigma\}$. We assume that $A \in \mathbb{R}^{m \times d}$ has full row-rank $m < d - 1$ and $b \in \mathbb{R}^m$. We furthermore assume that the system $\{Gx \leq h\}$ is such that its solution space is non-empty and has affine dimension d. That is, $\{Gx \leq h\}$ does not contain equality constraints or contradictory constraints.

Definition 4.1. For a d-asset portfolio space, we define a *near-optimal region* OPT as

$$OPT := \{w \in \mathbb{R}^d : Aw = b, Gw \leq h, f_\mu(w) \leq C_\mu, f_\Sigma(w) \leq C_\Sigma\},$$

with $A \in \mathbb{R}^{m \times d}$ of rank $m < d-1$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{k \times d}$, $h \in \mathbb{R}^k$ such that $\dim(\{Gw \leq h\}) = d$, and with $C_\mu, C_\Sigma \in \mathbb{R}$, and $f_\mu, f_\Sigma : \mathbb{R}^d \to \mathbb{R}$ convex.

We remark that in the case that f_μ as well as f_Σ are linear in the input, then OPT is a polytope provided that it is bounded. Possible non-linearity of f_μ and f_Σ may however result in OPT being constrained by continuous curves. The convexity of all constraints ensures that OPT is a convex region. We use Definition 4.1 in the remainder of this chapter for the convex region we aim to estimate with a polytope.

4.1.2 Method

Let OPT be defined as in Definition 4.1. We use the method by De Graaf and Van der Schans to estimate OPT by constructing a polytope P of maximal size within OPT. We do so by adding new vertices to P iteratively. We require a valid near-optimal portfolio $w_0 \in OPT$ for initialization. In every next iteration $i \in \{1, 2, 3, \dots\}$, w_i is defined as a point in OPT that lies furthest away from $\text{Conv}(w_0, \dots, w_{i-1})$,

$$w_i = \underset{w \in OPT}{\text{argmax}}\ d(w, P_{i-1}), \tag{19}$$

where $P_{i-1} = \text{Conv}(w_0, \dots, w_{i-1})$ and $d(\cdot, \cdot)$ denotes the Euclidean distance, i.e.

$$d(w, P_{i-1}) := \min_{x \in P_{i-1}} \|w - x\|. \tag{20}$$

This step is repeated until

$$d(w_i, P_{i-1}) < \epsilon$$

for some predefined $\epsilon > 0$.

De Graaf and Van der Schans solve optimization problem (19) using a numerical optimizer. Such a numerical optimizer can only be guaranteed to give valid output if it is provided with a possible solution as a starting point. That is, the optimizer should be called with not only information on (19), but also a starting point

$$z_i \in OPT \backslash P_{i-1}.$$

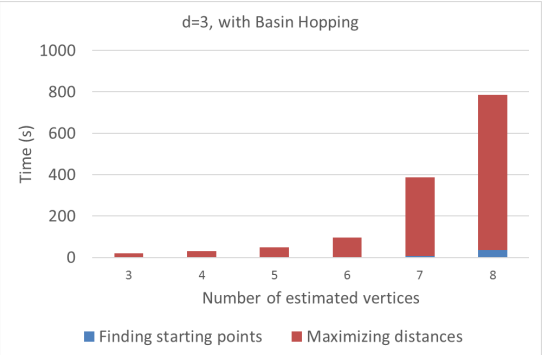
We summarize the method in Algorithm 5.

Algorithm 5: Near-optimal region estimation method by De Graaf and Van der Schans

1 *Initialize:* $w_0 \in OPT$, $P_0 = w_0$
2 **for** $i = 1, 2, \dots$ **do**
3 Find starting point $z_i \in OPT \backslash P_{i-1}$.
4 Find $w_i = \underset{w \in OPT}{\text{argmax}}\ d(w, P_{i-1})$ with z_i as starting point.
5 **if** $d(w_i, P_{i-1}) > \epsilon$ **then**
6 $P_i = \text{Conv}(P_{i-1}, w_i)$
7 **end**
8 **else**
9 **return** P_{i-1}
10 **end**
11 **end**

The most time consuming and complex steps of the method are those in steps 3 and 4 of Algorithm 5, that are to be solved alternatingly. Figure 14 shows time consumption by steps 3 and 4 of the algorithm depending on the number of polytope vertices estimated. We see that especially step 4 takes a lot of time to perform. We discuss both steps in detail separately in subsections 4.1.4 and 4.1.3 respectively.

(a) Computation time analysis of the example from Section 1.2. All displayed times are averages over three measurements



(b) Computation time analysis of estimating a near-optimal region consisting of 133 asset classes. The displayed times originate from a single measurement

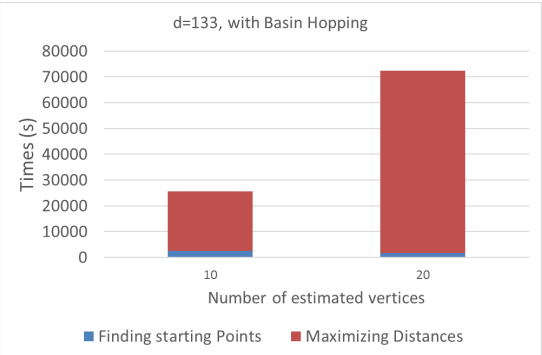


Figure 14: Figures 14a and 14b show the computation time of steps 3 (finding starting points) and 4 (maximizing distances) in Algorithm 5 as implemented by De Graaf and Van der Schans against the numebr of estimated vertices of the near-optimal region.

4.1.3 Distance maximization

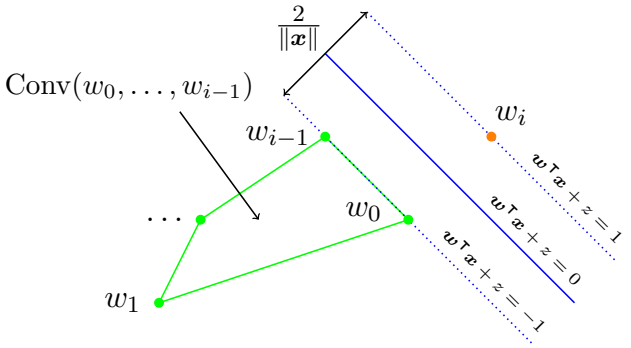
In this section we further investigate Step 4 of Algorithm 5, in which optimization problem (19) is solved. As mentioned, for each iteration i, this optimization problem requires a valid starting point z_i. We discuss how to get such a starting point in Section 4.1.4. For now, we assume that valid starting points are given.

Solving optimization problem (19) is not straightforward, since it involves calculating the distance in (20). De Graaf and Van der Schans propose calculating the distance in (20) by using Support Vector Machine (SVM) theory. They then maximize the distance using a local optimization algorithm. To ensure they do not end up with a point that is only locally an optimum, they insert the local optimization algorithm into a Basin Hopping algorithm to find a global optimum. We intuitively explain the SVM technique and the Basin Hopping algorithm that are part of solving problem (19).

Support Vector Machines

De Graaf and Van der Schans calculate distance $d(w_i, P_{i-1})$ using a linear classification technique from SVM theory. They construct a separating linear plane between $P_{i-1} = \text{Conv}(w_0, \ldots, w_{i-1})$ and w_i with maximal separation margin. They then measure the distance between the plane and w_i using the separation margin. The method is illustrated in Figure 15 by Van der Schans in [16] and is described in more detail in [15, Section 4.2].

Figure 15: Graphical representation of a SVM-classification problem. The blue line separates the green portfolios from the orange portfolio and has maximal separation margin $\frac{2}{\|x\|}$.



Basin Hopping

In iteration i, De Graaf and Van der Schans define the estimated distance $w \mapsto d(w, P_{i-1})$ as the objective function they want to maximize. Given a valid starting point and the constraints defining OPT, a local optimizer computes a portfolio w_i that lies furthest away from P_{i-1} locally. Note however that we were looking for a portfolio that lies furthest away from P_{i-1} globally. That is why De Graaf and Van der Schans use a Basin Hopping algorithm. As is further explained in [11], a Basin Hopping algorithm uses a local optimizer to find a global optimum with probability approaching one. It works by perturbing the input sequentially and running the local optimizer to select the "most extreme optimum so far". After a certain number of iterations, the algorithm is not able to select an optimum that is more extreme than the one in selection. The Basin Hopping algorithm concludes that the current selection is the global optimum and terminates.

4.1.4 Starting point

In this section we describe Step 3 of Algorithm 5 in detail. Suppose we want to estimate a near-optimal region OPT as in Definition 4.1 by iteratively solving optimization problem (19). In iteration i we may have a current estimation by polytope $P_{i-1} = \text{Conv}(w_0, \ldots, w_{i-1}) \subset \mathbb{R}^d$. To be able to find a new vertex w_i with maximal distance to P_{i-1}, we need to specify a valid starting point $z_i \in OPT \backslash P_{i-1}$. De Graaf and Van der Schans have developed a procedure to find z_i, of which a simplified version is described in Algorithm 6.

The algorithm tries to find a starting point by looping over vertices $w_0, \ldots, w_{i-1}$ in random order and then randomly displacing them. After picking a vertex $w \in \{w_0, \ldots, w_{i-1}\}$ (step 1), a uniformly random direction v in the kernel of the linear equality constraint matrix A of OPT is chosen using Theorem 3.1.2 (step 3). The algorithm now tries to walk a small distance $\delta > 0$ in direction v, starting in w (step 4). The algorithm checks whether the point $w + \delta \cdot v$ lies in $OPT \backslash P_{i-1}$. If so, we have found our starting point and the algorithm terminates. If not, the algorithm tries a different random direction v. If the algorithm has tried a specified number of directions without succeeding, it concludes that vertex w cannot be displaced to find a starting point and it moves to the next vertex. If the algorithm fails for all vertices, it concludes that a

new starting point cannot be found and the estimation of OPT is terminated.

Algorithm 6: Find valid starting point by random displacement

Data:

- Near-optimal region OPT as defined in Definition 4.1
- Polytope $P_{i-1} = \text{Conv}(w_0, \ldots, w_{i-1}) \subset OPT$
- Maximum number of tries per vertex $T \in \mathbb{N}$, stepsize $\delta > 0$

Result: Starting point $z \in OPT \backslash P_{i-1}$

```
for w ∈ {w_0, ..., w_{i-1}} do
    for t = 1, ..., T do
        Pick v⃗ a uniformly random direction in Ker(A)
        Let z = w + δ · v⃗
        if z ∈ OPT\P_{i-1} then
            return z
        end
    end
end
return None
```

We note that Algorithm 8 tries to find a new starting point in a naive way. Each vertex is randomly displaced without taking into account the position of other vertices. Especially in high dimensional cases, a situation may appear where a successful random direction $\vec{v}$ is chosen with very small probability.

The method of De Graaf and Van der Schans succeeds in producing an estimation of the near-optimal region. As their method was focused on testing the viability of the use of near-optimal regions, computation time and complexity were no priority. Heavy methods such as the Basin Hopping algorithm make the program computationally expensive, especially when the portfolio space contains many asset classes. In Section 4.2, we further analyze the program to see which parts are most involved and may be simplified or even omitted.

4.2 Improving the existing method

In this section we try to improve the method by De Graaf and Van der Schans. We specifically look at steps 3 and 4 of Algorithm 5. We find that we can improve time efficiency by changing step 4. We try to improve on step 3, but...

4.2.1 Distance Maximization

De Graaf and Van der Schans use a Basin Hopping algorithm to make sure that they find the global optimum for optimization problem (19). As the Basin Hopping algorithm possibly requires many iterations of optimization, it may make the process time consuming. One can wonder whether it is really necessary to make use of the Basin Hopping algorithm instead of using a local optimizer only. The method was designed for estimating a region of near-optimal portfolios by the convex hull of vertices. When the number of vertices used to estimate the region with is large, one can argue that the order in which we find them does not matter, since local optima will have to be added later on in the process anyway. In that case a local optimization may suffice.

The method of De Graaf and Van der Schans uses a Sequential Linear Squares Programming (SLSQP) algorithm, based on the work of Kraft in [22], as a local optimizer to solve optimization problem (20). Their entire method was written in the programming language Python 3.6 [35]. De Graaf and Van der Schans use an SLSQP solver from Python package SciPy, described in [20]. SLSQP is not the only local optimizer offered by the SciPy package. Moreover, there are other packages available in Python that offer local optimizers. We explore the different local optimizers from SciPy and from another software package called NLOPT [19]. We then test if they can replace the Basin Hopping algorithm entirely for the example from Section 1.2 with the goal to improve computation time.

Let OPT be defined as in Definition 4.1. To solve an iteration of optimization problem (19), we need a solver that can optimize a non-linear function

$$w \mapsto d(P_{i-1}, w)$$

subject to several constraints. These constraints need not be linear, as we can see from Definition 4.1. Not all optimizers accept non-linear constraints. Moreover, the optimizer needs to accept equality constraints as well as inequality constraints. Of the optimizers implemented by SciPy, only SLSQP accepts both equality constraints and non-linear constraints. Hence, there are no other optimizers than SLSQP in the SciPy package that are suited for the optimization problem we are trying to solve.

We now consider a second library for optimization, called NLOPT [19]. The NLOPT library offers several local optimizers that accept both non-linear (inequality) constraints and equality constraints. We tried them all and made a quick selection for further testing. We decide to test NLOPT's SLSQP implementation and its implementation for Constraint Optimization BY Linear Approximations (COBYLA).

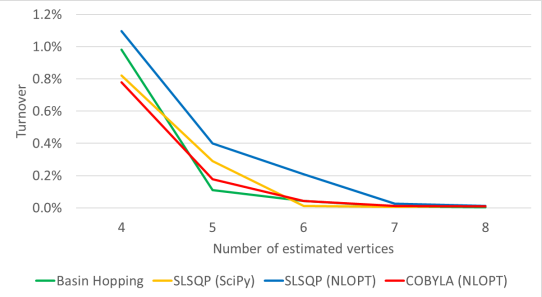
When testing the optimizers for estimating several near-optimal regions, we notice differences in their error behaviour. Especially NLOPT's COBYLA implementation produces unexpected errors frequently. For our test setting, COBYLA sometimes resulted in an unknown error which to our knowledge has so far not yet been specified in the literature. For more on this unknown error, the reader can look at an online discussion that we opened in [41]. Also SciPy's SLSQP solver is unstable in performing a successful optimization. Sometimes the optimization results in an error, which is explained in [22, Section 2.2.3], but cannot be avoided. When testing the SLSQP implementation by NLOPT, we did not encounter the error messages that appeared for the SciPy implementation.

When using a Basin Hopping Algorithm on top, these error messages are just ingnored and do not affect the end result. However, when using the local optimizer as the only optimization algorithm, it may very well make an impact on the optimization's end result. An unexpected error message then leads to the algorithm's premature termination and hence inaccurate estimation. We thus conclude that with respect to algorithm stability, NLOPT's SLSQP is the most appropriate local optimizer for our purpose.

We can bypass the local optimizers' error messages by just repeating until the optimizer is successful. We did so to produce the results displayed in Figure 16, in which we compare computation time and accuracy of the three local optimizers. Note that the computation time in the figure includes failed tries.

From Figure 16a we can see that performance of estimations using the Basin Hopping algorithm as well as the three local optimizers are comparable in terms of average turnover (see Definition 3.4). All four optimization methods succeed in reaching a turnover approaching zero when enough vertices are estimated. In Figure 16b we see

(a) Turnover between estimations with different local optimizers and Basin Hopping to OPT.



(b) Computation times of estimations with different local optimizers and Basin Hopping.

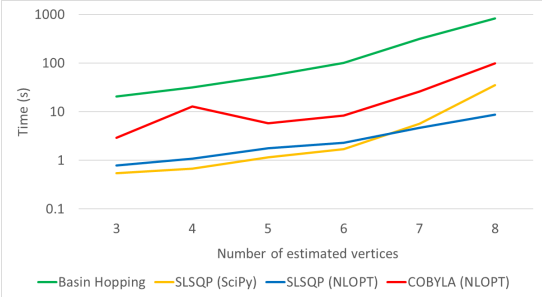


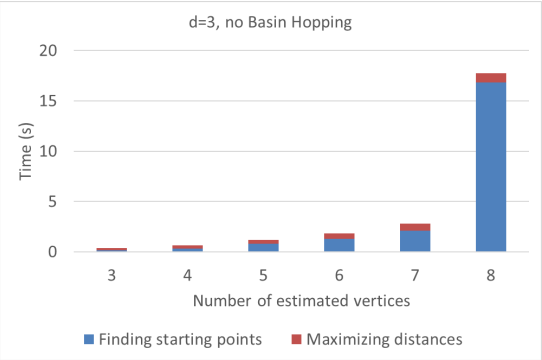
Figure 16: Comparison of local optimizers for the example from Section 1.2. All values are averages over five measurements.

that the differences in terms of computation time are a lot larger. The method using Basin Hopping takes on average up to almost 1000 seconds to estimate eight vertices of the Chopra near-optimal region, while NLOPT's SLSQP method on average takes 10 seconds to estimate eight vertices of the same region. We conclude that for the example from Section 1.2, the three local optimizers succeed in estimating the near-optimal region with a high precision, while they outperform the Basin Hopping method in terms of computation time. Together with the observations on error behaviour of SciPy's SLSQP and NLOPT's COBYLA optimizers, we conclude that NLOPT's SLSQP algorithm is the best choice for near-optimal region estimation based on the example from Section 1.2.

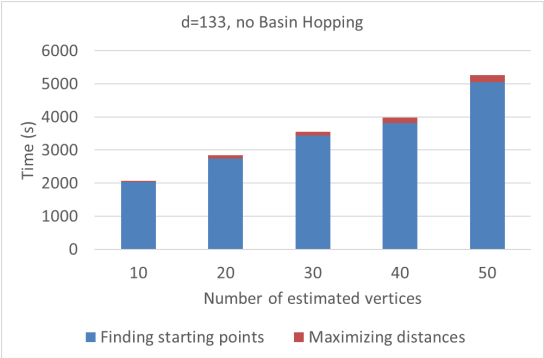
4.2.2 Starting point

In this section we investigate Step 3 from Algorithm 5 in which a starting point for the optimization in Step 4 is selected. In Section 4.1.3 we proposed removing the Basin Hopping algorithm for maximizing the distance in Step 4 of Algorithm 5. When doing so, total computation time of the algorithm decreases dramatically. We recalculate the detailed computation times from Figure 14 with COBYLA's SLSQP implementation as optimizer without a Basin Hopping algorithm. We show the results in Figure 17.

Figure 17: Figures 17a and 17b show average computation times of steps 3 and 4 in Algorithm 5. The results are obtained in the same way as the results in Figure 14, with the only difference being the replacement of the Basin Hopping algorithm by NLOPT's SLSQP algorithm to maximize the distance in (19).



(a) Computation time analysis of the example from Section 1.2. All displayed values are averages over five measurements

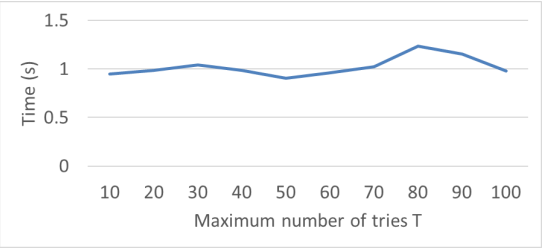


(b) Computation time analysis of estimating a near-optimal region consisting of 133 asset classes. The displayed times are averages over three measurements.

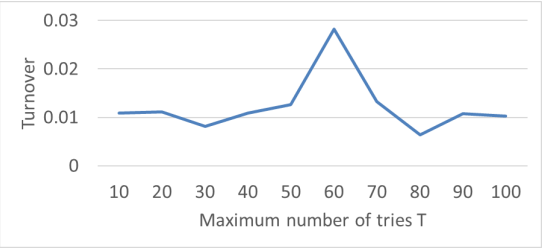
We see in Figure 17 that now Basin Hopping is removed from the optimization, the time for maximizing the distance in the optimization decreases drastically. A significant part of the total computation time is now spent on finding proper starting points; in some cases over 90 percent of the total computation time is spent on finding appropriate starting points. It is remarkable to find that an optimization's main struggle is finding proper starting points. In this subsection try to improve the way the starting points are selected. Recall the method to find starting points by De Graaf and Van der Schans in Algorithm 6 in Section 4.1.4.

When studying the behaviour of Algorithm 6 for several examples sets, we notice that vertex displacements in steps 2-5 often either succeed after a few tries or do not succeed at all. The default value for T in the model by De Graaf and Van der Schans was set at 100. We lower this default value and compare to the model with default 100 for two examples. Computation times of the convex hull estimation are shown in Figure 18.

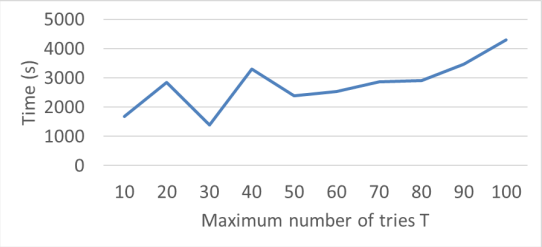
(a) Computation times for the example from Section 1.2 ($d = 3$). The results are averages of six measurements.



(b) Average turnovers for the example from Section 1.2 ($d = 3$). The results are averages of six measurements.



(c) Computation times for the example with 133 asset classes ($d = 133$). The results are averages of two measurements



(d) Average turnovers for the example with 133 asset classes ($d = 133$). The results are averages of two measurements.

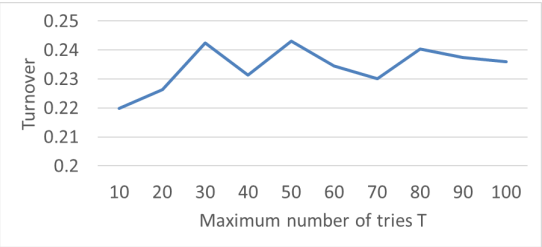


Figure 18: Figures 18a–18d show computation times and turnovers of Algorithm 5 on the vertical axes against different values for the maximum number of displacement tries T to find a starting point in Algorithm 6 on the horizontal axes. Turnovers were calculated with respect to the computation with the default maximal number of tries $T = 100$.

We see in Figure 18a that the maximum number of tries T in Algorithm 6 does not have a clear effect on the estimation's computation time. In Figure 18c, we see that the computation time appears to increase as the maximum number of displacement tries T in Algorithm 6 rises. In both Figures 18b and 18d, the relation of the maximum number of tries T to the average turnover is not clear.

We conclude lowering the maximum number of tries T displacing a vertex to find a starting point in Algorithm 6 may decrease the computation time to estimate a near-optimal region. It however does not happen in all cases. As we do not know the relation between the maximum number of displacement times T and the accuracy of the near-optimal region estimations, we do not draw further conclusion on the maximum number of displacement tries in Algorithm 6. Further analysis and testing is required.

We conclude that we were not able to improve the efficiency of Algorithm 6. When looking for an alternative approach to finding a starting point for problem 19, we find that we need to reformulate the entire method. We do so and find a new estimation approach, which we present in Section 4.3.

4.3 A Heuristic Approach

In this section we formulate an alternative approach for estimating a near-optimal portfolio region, in which we will use polytope theory from Section 2. We again assume OPT as in Definition 4.1. In Section 4.3.1, we rephrase the setting defined in Section 4.1. We present and prove an algorithm to estimate OPT with some additional assumptions in Section 4.3.2. In Section 4.3.4, we show that the made assumptions are valid for the

setting of Mean-Variance portfolio optimization.

Our approach is inspired by the approach of Van der Schans an De Graaf and roughly follows the same method. When the Basin Hopping algorithm is removed from the method, the majority of MDO's computation time is often spent on finding starting points for optimization. If we are able to change Algorithm 6 to find an optimization starting point for (19), we may improve on the method's efficiency. In this section we formulate an algorithm that heuristically looks for new starting points. The idea is to triangulate the intermediate polytopes P_i estimating OPT so that we know their boundaries. Next, we use the information on the boundary to search for a valid starting point $z \in OPT \backslash P_i$. To make this precise, we need to reformulate the setting presented in Section 4.1.1.

4.3.1 Theoretical Setting Reformulated

Let OPT be as defined in Defintion 4.1. As noted, the convexity of all the constraints ensures that OPT is a convex region. We also note that due to the equality constraints $\{Aw = b\}$ with $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$, we know that $\dim(OPT) \leq d - m$. From this point onwards, let us assume that $\dim(OPT) = d - m$.

Formulating algorithms for convex hull computation using polytope theory is considerably easier when we are dealing with a full-dimensional near-optimal region. We use Definition 2.17 from Section 2.4 to transform OPT into its affine dimension $d - m$.

Definition 4.2. Let OPT be as defined in Definition 4.1 with affine dimension $d - m$ with the extra restriction that f_μ is a linear function and f_Σ is strictly convex and continuous. We define the *transformation function of OPT to its affine space* as $\phi : \mathbb{R}^d \to \mathbb{R}^{d-m}$, where

$$\phi(w) = B^\intercal(w - A^+ b)$$

with A, b as in Definition 4.1 and the columns of B are an orthonormal basis of $(I_d - A^+ A)$.

We use ϕ from Definition 4.2 to transform the constraints that define OPT. For convenience, we gather the constraints defining OPT in

$$\mathcal{C} = \{g_1, \ldots, g_{k+2}\},$$

where

$$g_i(w) = \begin{cases} h_i - G_i w & \text{for } i = 1, \ldots, k \\ C_\mu - f_\mu(w) & \text{for } i = k+1 \\ C_\Sigma - f_\Sigma(w) & \text{for } i = k+2. \end{cases}$$

Note that now for all $w \in OPT$ and for all $g \in \mathcal{C}$, we have $g(w) \geq 0$. In particular,

$$OPT = \{w \in \mathbb{R}^d : Aw = b, g(w) \geq 0 \text{ for all } g \in \mathcal{C}\}. \tag{21}$$

Recall that ϕ is a bijective affine isometry and is thus invertible. We now define $\widehat{OPT}$ as

$$\widehat{OPT} = \{x \in \mathbb{R}^{d-m} : \hat{g}(x) \geq 0 \text{ for all } \hat{g} \in \widehat{\mathcal{C}}\}, \tag{22}$$

where $\widehat{\mathcal{C}} = \{\hat{g}_1, \ldots, \hat{g}_{k+2}\}$ with

$$\hat{g}_i(x) = \begin{cases} h_i - G_i \phi^{-1}(x) & \text{for } i = 1, \ldots, k \\ C_\mu - f_\mu(\phi^{-1}(x)) & \text{for } i = k+1 \\ C_\Sigma - f_\Sigma(\phi^{-1}(x)) & \text{for } i = k+2. \end{cases}$$

Note that we have a one-on-one relation between OPT and $\widehat{OPT}$

$$x \in \widehat{OPT} \iff \phi^{-1}(x) \in OPT,$$

so that $\widehat{OPT} = \phi(OPT)$. We call the elements of $\widehat{\mathcal{C}}$ the inequality constraints of $\widehat{OPT}$, as $x \in \widehat{OPT}$ if and only if $\hat{g}(x) \geq 0$ for every $\hat{g} \in \widehat{\mathcal{C}}$. We also note that all $g \in \mathcal{C}$ as well as all $\hat{g} \in \mathcal{C}$ are concave, because ϕ is affine. In particular, $\hat{g}_1, \ldots, \hat{g}_{k+1}$ are linear, and $\hat{g}_{k+2}$ is strictly concave.

We have now described the setting of near-optimal regions in portfolio optimization in a way we can use to formulate an estimation algorithm in the next section.

4.3.2 Constructing a full-dimensional starting simplex

We aim to estimate a near-optimal portfolio region OPT by a polytope P in $\mathbb{R}^d$. Our method consists of two parts. We start by constructing a simplex S inside OPT that is full-dimensional in $\text{aff}(OPT)$. Next, we extend S iteratively by maximizing the distance the boundary of OPT. This second step is similar to the procedure in Algorithm 8 with the main difference being that we choose our optimization starting points heuristically by looking at the current polytope's boundary, while Algorithm 8 chooses the starting points randomly. In Section 4.3.2, we present an algorithm to construct a full-dimensional simplex. In Section 4.3.3, we discuss a method to find starting points for optimization problem (19) heuristically.

We show how to construct a full-dimensional simplex S from only the constraints and two initial near-optimal portfolio's w_0, w_1. We present a method to find it in Algorithm 7 and show its validity in Theorem 4.3.

Algorithm 7: Construct a full-dimensional simplex in OPT

Data:

- Near-optimal region OPT as in Definition 4.1 with affine dimension $d-m$
- Transformation function ϕ as in Definition 4.2 transforming OPT to full-dimensional $\widehat{OPT} = \{x \in \mathbb{R}^{d-m} : \hat{g}(x) \geq 0 \text{ for all } \hat{g} \in \widehat{\mathcal{C}}\}$ as in formula (22).
- Two distinct near-optimal portfolios $w_0, w_1 \in \partial(OPT)$.

Result: A $(d-m)$-simplex $W = \{w_0, \ldots, w_{d-m}\}$ with vertices w_i on the boundary of OPT.

1 Let $X_1 = \{x_0, x_1\} = \{\phi(w_0), \phi(w_1)\}$.
2 **for** $i = 2, \ldots, d-m$ **do**
3 Let $\mathbf{x} = \frac{1}{|X_{i-1}|} \sum_{x \in X_{i-1}} x$
4 Let $E := \{\hat{g} \in \widehat{\mathcal{C}} : \hat{g}(\mathbf{x}) = 0\}$
5 **if** $E = \emptyset$ **then**
6 $v \in (\text{Span}(X_{i-1}))^{\perp} \backslash \{0\}$
7 **end**
8 **else**
9 Let $G^+ = \{x \in \mathbb{R}^{d-m} : \hat{g}(x) > 0 \text{ for all } \hat{g} \in E\}$.
10 Let $v \in (\text{Span}(X_{i-1}))^{\perp} \cap G^+ \backslash \{0\}$.
11 **end**
12 Let $x_i = \underset{y \in \widehat{OPT}}{\text{argmax}} |(y - \mathbf{x}) \cdot v|$
13 $X_i = X_{i-1} \cup \{x_i\}$
14 **end**
15 $W = \phi^{-1}(X_{d-m})$
16 **return** W

Theorem 4.3
Let OPT be as in Definition 4.1 with affine dimension $d-m$, let $\phi : \mathbb{R}^d \to \mathbb{R}^{d-m}$ be as in Definition 4.2, and let $w_0, w_1 \in \partial(OPT)$. Then Algorithm 7 computes a $(d-m)$-simplex $W \subset OPT$ with vertices $w_0, \ldots, w_{d-m} \in \partial(OPT)$.

Proof. We first show that the steps in the algorithm are always feasible. We next show that these steps also produce the desired output.

The steps in the algorithm that need clarification on their feasiblity are steps 5 – 12. Suppose we are in iteration $i \in \{2, \ldots, d-m\}$ of the loop from step 2. Intuitively, steps 5 – 12 choose a direction v that is orthogonal to the span of the current set X_{i-1} and define x_i by maximizing the distance in direction v to add a dimension to X_{i-1}. Note that since $|X_{i-1}| = i$, we have $\dim(X_{i-1}) \leq i - 1 < d - m$. By assumption $\dim(OPT) = d - m$, so that it is possible to add a dimension to X_{i-1}, i.e.

$$\text{Span}(X_{i-1})^{\perp} \neq \emptyset.$$

Hence, it is possible to choose a valid direction v such that $|(x_i - \mathbf{x}) \cdot v| > 0$. To show that both steps 6 and 10 choose such a valid direction v, we make the observation that $x \in \widehat{OPT}$ and that $E = \emptyset$ if and only if $\mathbf{x} \in \widehat{OPT}^{\circ}$. We now distinguish two cases.

First, suppose that $\mathbf{x} \in \widehat{OPT}^{\circ}$. Then for any $v \in \mathbb{R}^{d-m}$, there is $\delta > 0$, such that $\mathbf{x} + \delta v \in \widehat{OPT}$.

Now suppose that $\mathbf{x} \notin \widehat{OPT}^{\circ}$, i.e. $\mathbf{x} \in \partial(\widehat{OPT})$. Then $E \neq \emptyset$. There is hence at least one inequality constraint $\hat{g} \in \widehat{\mathcal{C}}$ that holds with equality for $\mathbf{x}$. Since $\hat{g}$ is concave, we can use Jensen's inequality to see that

$$0 = \hat{g}(\mathbf{x}) \geq \frac{1}{i} \sum_{j=0}^{i-1} \hat{g}(x_j).$$

Since $x_0, \ldots, x_{i-1} \in \widehat{OPT}$ and thus $\hat{g}(x_0), \ldots, \hat{g}(x_{i-1}) \geq 0$, we now see that

$$\hat{g}(x_0) = \cdots = \hat{g}(x_{i-1}) = 0.$$

Note that since $\hat{g}_{k+2}$ is strictly concave, we know that $g_{k+2} \notin E$, so that E consists of linear functions only. Hence also for all $y \in \mathrm{Span}(X_{i-1})$, $\hat{g}(y) = 0$. We now claim that we can walk $\delta > 0$ in the direction of v without leaving $\widehat{OPT}$.

We show that our claim is true by a proof of contradiction. Suppose that for all $\delta > 0$, $\mathbf{x}_\delta v \notin \widehat{OPT}$. So

$$H = \{h \in \widehat{\mathcal{C}} : h(\mathbf{x} + \delta v) < 0\} \neq \emptyset.$$

Note that since $v \in G^+$, for all $g \in E$, $g(\mathbf{x} + \delta v) \geq 0$. Hence $H \cap E = \emptyset$ and thus for all $h \in H$, $h(\mathbf{x}) > 0$. Now, for all $h \in H$ we have that since h is continuous, there is $\epsilon_h > 0$ such that $h(\mathbf{x} + \epsilon_h v) = 0$. Now define $\delta' = \min_{h \in H} \delta_h$ and observe that $\hat{g}(\mathbf{x} + \delta' v) \geq 0$ for all $\hat{g} \in \widehat{\mathcal{C}}$ and thus $\mathbf{x} + \delta v \in \widehat{OPT}$. Hence our claim is true.

We note that $G^+ \cap (\mathrm{Span}(X_{i-1}))^\perp \neq \emptyset$ since $\dim(\widehat{OPT}) = d - m > i - 1$ by assumption.

We thus see that we always find $v \in \mathrm{Span}(X_{i-1})^\perp$ and x_i such that $|(x_i - \mathbf{x}) \cdot v| > 0$. We conclude that the steps in the algorithm are feasible.

We show that the output $W = \{w_0, \ldots, w_{d-m}\}$ of Algorithm 9 is a $d - m$ simplex in OPT and that $w_i \in \partial(OPT)$ for $i \in [d - m]$. To show that W is a $d - m$ simplex, we show that X_{d-m} from the algorithm is a full-dimensional simplex in $\widehat{OPT} \equiv \phi(OPT)$. We then use Lemma 2.21 to conclude that $W = \phi^{-1}(X_{d-m})$ is a $(d - m)$-simplex in OPT.

We show that the algorithm constructs full-dimensional $X_{d-m} \subset \mathbb{R}^{d-m}$ by an argument of induction. Since ϕ is a bijection, we know that $x_0 \neq x_1$ and hence X_1 has affine dimension $\dim(X_1) = 1$. Let $i \in \{2, \ldots, d - m\}$ and suppose that for all $j \leq i - 1$, $\dim(X_j) = j$. We show that then $\dim(X_i) = i$.

We know that $\dim(X_{i-1}) = i - 1$. Then by definition,

$$\mathrm{rank}(x_1 - x_0, \ldots, x_{i-1} - x_0) = i - 1.$$

Let $v \in \mathbb{R}^{d-m}$ be as defined by the algorithm. As we have shown in part 1 of the proof, $v \in \mathrm{Span}(X_{i-1})^\perp \backslash \{0\}$ and there exists $\delta > 0$ such that $\mathbf{x} + \delta v \in \widehat{OPT}$. Hence, x_i has a nonzero component in $\mathrm{Span}(X_{i-1})^\perp$. Therefore, $x_i - x_0$ is linearly independent of $(x_1 - x_0, \ldots, x_{i-1} - x_0)$ and thus $\mathrm{rank}(x_1 - x_0, \ldots, x_i - x_0) = i$, which in turn implies that $\dim(X_i) = i$. We conclude that $\dim(X_{d-m}) = d - m$ and hence $\dim(W) = d - m$, using Lemma 2.21.

To see why $w_0, \ldots, w_{d-m} \in \partial(OPT)$, note that by construction $x_2, \ldots, x_{d-m} \in \partial(\widehat{OPT})$. Let $i \in \{2, \ldots, d - m\}$. Then there is $\hat{g} \in \hat{\mathcal{C}}$ such that $\hat{g}(x_i) = 0$. Then there

is $g \in \mathcal{C}$ such that $g(\phi^{-1}(x_i)) = 0$ and hence $w_i = \phi^{-1}(x_i) \in \partial(OPT)$. By assumption $w_0, w_1 \in \partial(OPT)$.

□

We note that the steps in Algorithm 7 are all easy to implement in a scientific programming language such as Python [35]. Algorithm 7 thus provides a practical method to find a simplex in a near-optimal region OPT as in Definition 4.1 with affine dimension $d - m$ and initial portfolios $w_0, w_1 \in \partial(OPT)$.

4.3.3 Heuristically finding new starting points

We now show how we can use Algorithm 7 to estimate the near-optimal region OPT with a polytope. Recall the Algorithm 5 from [15] described in Section 4.1.2. It iteratively adds points to the polytope by maximizing the distance to the current polytope within OPT, i.e.

$$\text{Given } P = \text{Conv}(w_0, \ldots, w_{i-1}), \quad \text{define } w_i = \underset{w \in OPT}{\text{argmax}} \{d(P, w)\}. \tag{23}$$

Also recall that after removing Basin Hopping from the optimization procedure, the main challenge and most time consuming step in the Algorithm 5 was to find a starting point $z \in OPT \backslash P$ to initialize optimization problem (23). In our method for estimating OPT, we initialize P as the full-dimensional simplex computed by Algorithm 7. We then follow the approach of De Graaf and Van der Schans by solving problem (23). We specify the starting point for this the optimization problem using a triangulation of P. The method of finding the starting point is specified in Algorithm 8. The complete method of estimating OPT is formalized in Algorithm 9.

Algorithm 8: Find a starting point for optimization problem (23) heuristically.

Data:

- Near-optimal region OPT as in Definition 4.1 with affine dimension $d - m$
- $(d - m)$-polytope $P = (w_0, \ldots, w_{i-1})$ with $w_0, \ldots, w_{i-1} \in \partial(OPT)$.

Result: A point $x_i \in OPT \backslash P$ with distance $d(x_i, P) > 0$

```
1  Let F be the set of facets of P
2  for f ∈ F do
3  |  Let x ∈ f^o
4  |  Let v be the normalized vector in R^{d-m} that is orthogonal to the facet
   |    defining hyperplane h_f facing outward of P
5  |  Let α ∈ R be maximal such that x + αv ∈ OPT
6  |  y = x + αv
7  |  if x ≠ y then
8  |  |  break
9  |  end
10 end
11 return y
```

We argue that all steps in Algorithm 8 are feasible. In step 1 we can find the set of all facets $\mathcal{F}$ of P using Algorithms 1 and 2 from Section 2.3. In step 3, we can take $x \in f^{\text{o}}$ by simply letting x be the average of all facet vertices. In step 4, we can find

v using standard linear algebra. We can find maximal $\alpha \in \mathbb{R}$ in step 5 by increasing α stepwise since OPT is convex. Checking whether $x + \alpha v \in OPT$ can be done by checking all constraints $\hat{g}(y + \alpha v)$.

We note that if $P \subsetneq OPT$, then there exists a facet $f \in \mathcal{F}$ such that for the complement h_f^- of the facet defining halfspace of f,

$$OPT \cap h_f^- \neq \emptyset.$$

We now present our near-optimal region estimation algorithm in Algorithm 9.

Algorithm 9: Estimating OPT

Data:

- Near-optimal region OPT as in Definition 4.1 with affine dimension $d - m$
- Transformation function ϕ as in Definition 4.2 transforming OPT to full-dimensional $\widehat{OPT} = \{x \in \mathbb{R}^{d-m} : \hat{g}(x) \geq 0 \text{ for all } \hat{g} \in \widehat{\mathcal{C}}\}$ as in (22).
- Two distinct portfolios $w_0, w_1 \in \partial(OPT)$.
- Required precision $\epsilon > 0$

Result: A $(d - m)$-polytope P estimating OPT up to precision ϵ

```
1  Let P be a (d − m)-simplex as computed by Algorithm 7
2  for i = 1, 2, ... do
3  |  Let z_i be the starting point computed by Algorithm 8
4  |  Let x_i = argmax_{w∈OPT} {d(P, w)} using z_i as a starting point
5  |  if d(P, x_i) > ε then
6  |  |  P = P ∪ {x_i}
7  |  end
8  |  else
9  |  |  break
10 |  end
11 end
12 return P
```

We point out that finding the full-dimensional simplex as in Algorihtm 7 is necessary for the convex hull computation as in Algorithm 9. When a polytope with dimension lower than $\dim(OPT)$ is used instead, orthogonal vectors on its facets will stay in that same lower dimension. This will result in the estimate region missing an entire dimension of OPT.

Algorithm 9 uses polytope theory to find starting points for problem 19 heuristically. Note that in order to compute the boundary of P in Algorithm 8, we need to triangulate P first. For portfolio spaces with many asset classes (d is large) this can be time consuming as was shown in Section 2.5.3.

4.3.4 Mean-Variance optimization

We show that in the special case of a near-optimal region OPT based on Mean-Variance Optimization, OPT satisfies the assumptions we made in Section 4.3.1. We start by

specifying the Mean-Variance model, which we first introduced in Section 1.1, by specifying the return constraint f_μ and the risk constraint f_Σ from Definition 4.1.

Definition 4.4. Consider a portfolio space of d asset classes with estimated means $\mu \in \mathbb{R}^d_{>0}$ and covariance matrix $\Sigma \in [-1,1]^{d\times d}$, where no two asset classes have covariance exactly zero. we define a *Mean-Variance near-optimal region OPT* as

$$OPT := \{w \in \mathbb{R}^d : Aw = b, Gw \leq h, w^\intercal \mu \geq C_\mu, w^\intercal \Sigma w \leq C_\Sigma\},$$

with $A \in \mathbb{R}^{m\times d}$ of rank $m < d-1$, $b \in \mathbb{R}^m$, $G \in \mathbb{R}^{k\times d}$, $h \in \mathbb{R}^k$ such that $\dim(\{Gw \leq h\}) = d$, and cutoffs $C_\mu, C_\Sigma \in \mathbb{R}$ such that OPT is non-empty.

Definition 4.4 is the one De Graaf and Van der Schans used to define a near-optimal region and to develop their method in [15] and corresponds to Definition 1.3. De Graaf and Van der Schans used a *strategic portfolio* w^* on the efficient frontier to define

$$C_\mu = w^{*\intercal}\mu - \delta_\mu \quad \text{and} \quad C_\Sigma = w^{*\intercal}\Sigma w^* + \delta_\Sigma \tag{24}$$

for margin values $\delta_\mu, \delta_\Sigma > 0$.

In Section 4.3.2 we assumed that OPT as defined in Definition 4.1 has affine dimension $d - m$. We show that this is indeed the case when we define OPT according to Definition 4.4.

Theorem 4.5
Suppose we have a space of portfolios in d asset classes with expected returns $\mu = (\mu_1, \ldots, \mu_d)^\intercal \in \mathbb{R}^d_{>0}$ and covariance matrix $\Sigma \in [-1,1]^{d\times d}$. Let a near-optimal portfolio region OPT be defined as in Definition 4.4. If there exists $x \in OPT$ for which $x^\intercal\mu > C_\mu$, $x^\intercal\Sigma x < C_\Sigma$ and $Gx < h$, then OPT has affine dimension $d - m$.

Proof. Note that

$$OPT \subset \{w \in \mathbb{R}^d : Aw = b\},$$

which has affine dimension $d - m$. Hence we know that OPT has affine dimension at most $d - m$.

We now show that $\dim(OPT) \geq d - m$. We write

$$\mu = (\mu_1, \ldots, \mu_d)^\intercal, \;\; \Sigma = \begin{pmatrix} \sigma_{1,1} & \cdots & \sigma_{1,d} \\ \vdots & \ddots & \vdots \\ \sigma_{d,1} & \cdots & \sigma_{d,d} \end{pmatrix}.$$

Let $x \in OPT$ be such that $x^\intercal\mu > C_\mu$ and $x^\intercal\Sigma x < C_\Sigma$ and $(Gx)_i < h_i$ for all i. Let $b_1, \ldots, b_{d-1} \in \mathbb{R}^d$ be normalized linearly independent vectors in $\mathrm{Ker}(A)$. Define

$$\begin{aligned} \alpha &= \min_{i,j}\left(\frac{h_j - (Gx)_j}{G_j - b_i}\right), \\ \beta &= \frac{x^\intercal\mu - C_\mu}{\max_i b_i^\intercal \mu}, \\ \gamma &= \frac{C_\Sigma - x^\intercal\Sigma x}{3d^2}, \end{aligned}$$

and let $\epsilon = \min(\alpha, \beta, \gamma)$. Now, for $j \in \{1, \ldots, d-m\}$ define $v_j \in \mathbb{R}^d$ as

$$v_j = x + \epsilon \cdot b_j.$$

[4] This is not a strong assumption, since in practice no two asset classes are completely independent.

Then for every j, we have

$$\begin{aligned}
A_i v_j &= A_i x + \epsilon A_i b_j = A_i x = b_i && \text{(for all } i \in [m]\text{)} \\
G_i v_j &= G_i x + \epsilon\, G_i b_j \leq G_i x + \alpha G_i b_j \leq G_i x \frac{h_i - G_i x}{G_i b_j} G_i b_j = h_i && \text{(for all } i \in [k]\text{)} \\
v_j^\intercal \cdot \mu &= x^\intercal \mu + \epsilon\, b_j^\intercal \mu \geq x^\intercal > C_\mu \\
v_j^\intercal \Sigma v_j &= x^\intercal \Sigma x + 2\epsilon\, x^\intercal \Sigma b_j + \epsilon^2 b_j^\intercal \Sigma b_j \\
&\leq x^\intercal \Sigma x + 2\epsilon\, d^2 + \epsilon^2 d^2 \leq x^\intercal \Sigma x + 3\epsilon\, d^2 \leq C_\Sigma.
\end{aligned}$$

We thus see that $v_j \in OPT$. Moreover, since $b_1, \ldots, b_{d-m}$ are linearly independent, $v_1, \ldots, v_{d-m}$ are also linearly independent. Hence $\dim(OPT) \geq d - m$.

We conclude that $\dim(OPT) = d - m$.

□

Furthermore, we note that $x \mapsto x^\intercal \mu$ is linear and that $x \mapsto x^\intercal \Sigma x$ is continuous and strictly convex since no two asset classes have correlation exactly zero.

Recall that another requirement for Algorithm 9 is that we need to specify two near-optimal portfolios $w_0, w_1 \in \partial(OPT)$. In the Mean-Variance setting where cutoffs C_μ, C_Σ are defined as in (24), we can find such $w_0, w_1 \in \partial(OPT)$ by maximizing and minimizing some coordinate $i \in [d]$ using strategic portfolio w^* as starting point for the optimization,

$$w_0 = \operatorname*{argmax}_{x \in OPT} x_i, \qquad w_1 = \operatorname*{argmin}_{x \in OPT} x_i.$$

Suppose that the near-optimal region OPT is defined as in Definition 4.4. We can then readily apply Algorithm 9 to find a convex hull estimating OPT up to a precision $\epsilon > 0$.

4.3.5 A Pragmatic Algorithm

We note that Algorithm 9 makes use of Algorithm 3, defined in Section 2.3, for triangulation of a polytope. Recall that computation time for triangulating polytopes increases exponentially with the polytope dimension. This will not be an issue for the example from Section 1.2, where the affine dimension of the near-optimal region is only two. In practice, there are often many different asset classes, and Algorithm 9 may be very slow due to the triangulations that are performed. In order to still be able to generate an estimation of a high-dimensional near-optimal region, we make some adaptations to Algorithm 9.

We use that we do not need any triangulation algorithm to generate a full-dimensional simplex in Algorithm 7. Moreover, the triangulation and the boundary of a simplex are trivial, so that we can perform a some iterations of Algorithm 9 without having to call Algorithm 3. We present the adapted version in Algorithm 10.

Algorithm 10: Pragmatically estimate OPT heuristically

Data:

- Near-optimal region OPT as in Definition 4.1 with affine dimension $d-m$
- Transformation function ϕ as in Definition 4.2 transforming OPT to full-dimensional $\widehat{OPT} = \{x \in \mathbb{R}^{d-m} : \hat{g}(x) \geq 0 \text{ for all } \hat{g} \in \widehat{\mathcal{C}}\}$ as in (22).
- Two distinct portfolios $w_0, w_1 \in \partial(OPT)$.

Result: A $(d-m)$-polytope P estimating OPT.

Let $P = \text{Conv}(x_0, \ldots, x_{d-m})$ be a $(d-m)$-simplex as computed by Algorithm 7
P has boundary $\mathcal{F} = \{F_0, \ldots, F_{d-m}\}$ with $F_i = \{x_j \in \{x_0, \ldots, x_{d-m}\} : j \neq i\}$.
for $i = 0, 1, 2, \ldots, d-m$ **do**
    Let $x = \frac{1}{d-m} \sum_{j \in [d-m], j \neq i} x_j$
    Let v be the normalized vector in $\mathbb{R}^{d-m}$ that is orthogonal to the facet defining hyperplane h_{F_i} facing outward of P
    Let $\alpha \in \mathbb{R}$ such that $x + \alpha v \in OPT$
    if $\alpha \neq 0$ **then**
        $z_i = x + \alpha v$
        Let $y_i = \underset{w \in OPT}{\text{argmax}} \{d(P, w)\}$ using z_i as a starting point
        $P = P \cup \{y_i\}$
    end
end
return P

Note that for a near-optimal region OPT with affine dimension $\dim(OPT) = d-m$, Algorithm 10 estimates at most $2(d-m+1)$ vertices. When OPT has high affine dimension, this may very well be enough to make a robust near-optimal region for the investor to choose his portfolio from.

We tested four different near-optimal region computation methods and show the results In Figure 19. We see that the computation time of the three methods presented in this section (SLSQP by NLOPT, Heuristic, and Pragmatic) improve drastically on computation time when compared to the method by De Graaf and Van der Schans (Basin Hopping method). We also see that these three methods succeed in reaching average turnovers from the Basin Hopping method of below 5%. In the high-dimensional case (Figures 19c and 19d), we see that Algorithm 9 (Heuristic) performs best in terms of computation time. The method using SLSQP by NLOPT performs best in terms of average turnover.

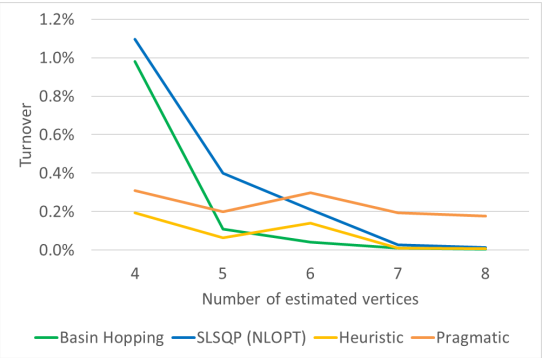
We remark that in testing the estimations of Figures 19c and 19d, there was little difference between the results from Algorithm 9 (Heuristic) and Algorithm 10 (Pragmatic). This makes sense, as the dimension of the estimated near-optimal region (133) was larger than the maximal number of estimated vertices (120), so that no triangulations were performed in these tests.

We also note that the figures of the average turnover may not be representative of the proximity of the estimated near-optimal regions to the actual near-optimal region. We calculated the average turnovers with respect to an estimation of the near-optimal region and not with respect to the actual near-optimal region, as it is not available to us.

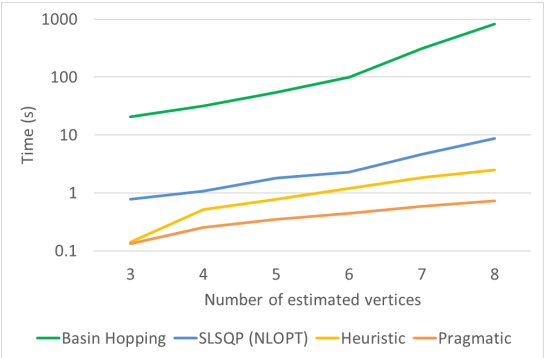
It may be that the estimation we compare to (using Basin Hopping) misses a part of the actual near-optimal region that was captured by one of the other calculated estimates.

In short, we conclude from Figure 19 that Algorithms 9 and 10 are best in terms of computation time, but that Algorithm 5 using NLOPT'S SLSQP implementation to solve Step 4 is better in terms of estimation precision. All three Algorithms drastically improve on computation time compared to the implementation of De Graaf and Van der Schans.

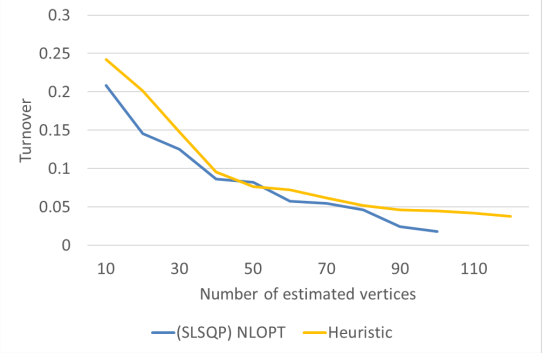
(a) Average turnover ($d = 3$)



(b) Computation time ($d = 3$)



(c) Average turnover ($d = 133$)



(d) Computation time ($d = 133$)

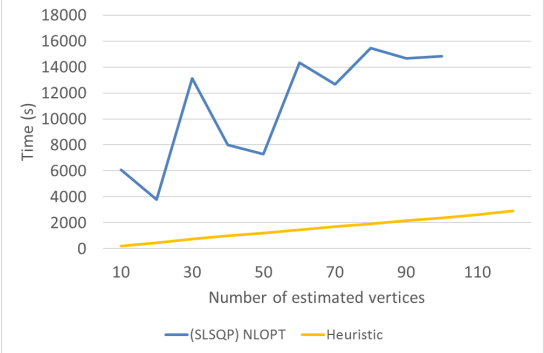


Figure 19: We compare different near-optimal region estimation techniques. Figures 19a and 19b are based on estimations of the example from Section 1.2. For that example, we compare four different estimation methods. Note that the Time axis has a logarithmic scale. The green line represents the estimation as implemented by De Graaf and Van der Schans. The blue line the implementation where the Basin Hopping algorithm is replaced by NLOPT's SLSQP implementation (see Section 4.1.3). The yellow line represents the method from Algorithm 9 and the orange line represents Algorithm 10. Figures 19c and 19d are based on a portfolio space of 133 asset classes. We left out the existing implementation of De Graaf and Van der Schans in the tests, as it takes too much time. We do not display the results from 10, as they very much coincide with the results from the yellow line (this makes sense, since no triangulation was necessary for this example). For all figures, estimated turnover was calculated from the near-optimal region estimation using the implementation from De Graaf and Van der Schans with a maximum number of vertices.

4.4 Conclusion

We have investigated the near-optimal region construction method as developed by De Graaf and Van der Schans in [15] and [16], focusing on improving its computation time. When analyzing the current method's computation time, we saw that most of the time is spent on two steps: finding starting points for optimization problem (19), and actually solving optimization problem (19). We can improve on computation time by removing the Basin Hopping algorithm from the method to solve optimization problem (19). There are several possible local optimizers that can replace the Basin Hopping algorithm. We tried several, of which we recommend the SLSQP optimizer as implemented by the

NLOPT package, because of its stability compared to the other optimizers. When tested, this replacement leads to a decrease in computation time of more than a factor ten. We were not able to improve the method to find a starting point in the existing implementation. There does appear to be a relation between the maximum number of tries of displacing a vertex to find a new starting point in Algorithm 6 and the computation time, but the results are not sufficient to draw conclusions. Further analysis is required.

We proceeded to develop a new method for finding starting points for optimization problem (19), letting go of the existing implementation. We present a new near-optimal region construction method, which again adds points iteratively to the polytope estimate, making use of polytope theory discussed in Chapter 2. When tested for low-dimensional regions, our new method estimates the convex region with precision similar to that of the method using NLOPT's SLSQP, while taking considerably less computation time to do so. For high-dimensional cases, computation time of our new algorithm is still superior to that of the method using NLOPT's SLSQP algorithm, but the estimations seems less accurate.

Our new method may be problematic when applied to a near-optimal region of high affine dimension where many vertices need to be estimated, since it makes use of the triangulation algorithm in Algorithm 3. To avoid using Algorithm 3, we presented a pragmatic version of our algorithm. The pragmatic algorithm should be further tested before we can draw conclusions on its performance.

5 Concluding remarks

The main contribution of this thesis is that we showed how to apply polytope theory to near-optimal portfolio optimization. By doing so, we provided tools to analyze near-optimal regions, in particular through triangulation. We already encountered several applications for triangulation in portfolio optimization: we used triangulations for sampling, calculating a polytope's center of mass, and for a new method for estimating a near-optimal region. Other applications of polytope theory and triangulations in particular could be explored in future research.

We reformulated the Beneath-and-Beyond convex hull computation algorithm as a triangulation algorithm which can be used for triangulating near-optimal region estimates. The result is a triangulation algorithm that can deal with polytopes of higher dimension than the algorithms provided by currently available open software can. Our algorithm is, however, still limited to around twenty to thirty dimensions, depending on the number of vertices in the polytope to be triangulated. This causes limitations in the area of portfolio optimization, where portfolio's can consist of over 100 different asset classes. A possible solution to this issue may be to fix ratios between several asset classes, as is already practice for many investors, effectively decreasing the near-optimal region's affine dimension. The effect on the optimization results should however be investigated.

The theory of near-optimal regions was developed for increasing robustness of portfolio optimization results. We showed how to sample from these regions to test their robustness properly. Then we showed an example where the near-optimal region is indeed more robust than the mean-variance optimal portfolio, similar to the results from De Graaf and Van der Schans in [16]. Furthermore, we showed that we can find a single portfolio inside the near-optimal region that is more robust than the mean-variance optimal portfolio: the region's centre of mass. In our example, however, we see that the mean-variance optimal portfolio remains near-optimal in most cases after perturbing the estimation's input parameters. In this regard, it is as robust as the centre of mass portfolio.

Once the near-optimal portfolio region is computed, an investor still has to select the portfolio he or she will actually invest in. We showed two approaches to select a single portfolio out of the near-optimal region. First, we showed how we can use knowledge of the near-optimal region's structure, using a triangulation, to find the region's centre of mass. Second, we showed how additional arguments, such as diversification, may be included to select a single portfolio from the near-optimal region.

We changed the implementation of the method by De Graaf and Van der Schans, decreasing its computation time. The new implementation is able to estimate near-optimal regions with accuracy similar to the old implementation, but uses less than one tenth of its computation time. We also introduced an alternative estimation method which computes near-optimal region estimates in an even shorter time. We recommend using the new implementation of the method by De Graaf and Van der Schans as the standard near-optimal region estimation model. The old implementation may also be used as a benchmark for testing other methods. The new estimation method is promising, but should be improved on accuracy and should be adapted to make sure it does not get stuck triangulating a high-dimensional polytope, which takes a very long time.

There are several possibilities for even different methods to estimate a near-optimal region OPT. We mention two that could be further investigated. In the introduction of Section 4.1, we mentioned the Double Description method as described by Fukuda

and Prodon in [13], which allows to switch back and forth between a polytope's $\mathcal{V}$-representation and its $\mathcal{H}$-representation. We also mentioned that this method cannot readily be applied to a near-optimal region OPT, as it may be constrained by non-linear functions. If we however linearize all non-linear constraints defining OPT, we effectively estimate OPT with a polytope P, given by its $\mathcal{H}$-representation. Using the Double Description method, we should now be able to retrieve P's $\mathcal{V}$-representation.
Another possibility is to estimate OPT using a large sample. If we know OPT's affine subspace and one interior point, we can use the Hit-and-Run method from Section 3.1.1 to sample uniformly from OPT. Given a large sample (or point set) S in OPT, we can compute its convex hull using one of many convex hull computation algorithms. We refer to the paper [1] by Avis, Bremner, and Seidel for more on convex hull computation algorithms.

Finally, we put forward that most graphs and data that were generated during this research are based on two example portolio spaces. Although the use cases in this thesis are representative, we recommend to test the results in more elaborate use cases.

Recommendations to Ortec Finance

A contribution that may be valuable to Ortec Finance, is that we succeeded in improving the computation time of estimating near-optimal regions. We presented several methods to do so, of which we recommend the one using NLOPT's SLSQP implementation for general use. We recommend this method over the method of Algorithm 9, which constructs a near-optimal region by constructing a full-dimensional simplex and adding new points using its boundary, in spite of its superior computation time, because it shows more accurate results and because it is applicable in all cases. Algorithm 9 may fail an estimation when it tries to triangulate a polytope of high affine dimension. A possible solution to this problem may be a more pragmatic implementation in Algorithm 10. Due to an insufficient amount of tests on that method, we do not recommend Ortec Finance to use it as the standard method. all methods can however be used to compare for consistent results.

We recommend Ortec Finance to further investigate the centre of mass portfolio as the suggested investment portfolio. One of the assumptions the concept of near-optimal regions is based on, is that all near-optimal portfolios are sufficiently satisfactory for the investor. We saw that not only the centre of mass portfolio, but also the centroid and the mean-variance optimal portfolio in most cases remain near-optimal when the estimation's input is perturbed. Hence we see no added value in the centre of mass portfolio with respect to the mean-variance optimal portfolio in terms of robustness. The centre of mass portfolio may however yield other advantages. Its performance in diversification, as explored in Section 3.4, could for instance be investigated.

References

[1] D. Avis, D. Bremner, R. Seidel, How good are convex hull algorithms? *Comput. Geom.*, Vol. 7, pp. 256–301, 1997.

[2] D. Barnette, A Proof of the Lower Bound Conjecture for Convex Polytopes, *Pacific J. Math.*, Vol. 46, No. 2, pp. 349–354, 1973.

[3] C.B. Barber, D.P. Dobkin, H. Hudanpaa, The Quickhull Algorithm for Convex Hulls, *ACM Trans. Math. Software*, Vol. 22, No. 4, pp. 469–483 December 1996.

[4] C.B. Barber, *http://www.qhull.org/html/index.htm*, Arlington, MA (1995–2015).

[5] M.F. Beatty Jr., *Principles of Engineering Mechanics*, Volume 2: Dynamics—The Analysis of Motion, Springer, New York, 2006.

[6] A. Ben-Tal and A. Nemirovski, Robust Convex Optimization, *Math. Oper. Res.*, Vol. 23, Issue 4, pp. 769–805, 1998.

[7] D. Bertsimas, V. Gupta, and I.C. Paschalidis, Inverse Optimization: A New Perspective on the Black-Litterman Model, *Oper. Res.*, Vol. 60, Issue 6, pp. 1389–1403, 2012.

[8] J. Brodie et al, Sparse and stable Markowitz portfolios, *Proc. Natl. Acad. Sci. USA*, Vol. 106, Issue 30, pp. 12267–12276, 2009.

[9] V.K. Chopra, Improving Optimization, *The Journal of Investing*, Vol. 2 No. 3 pp. 51–59, 1993.

[10] H. Cramer, *Mathematical Methods of Statistics*, Almqvist & Wiksells, Uppsala, 1945.

[11] J.P.K. Doye and D.J. Wales, Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms, *Journal of Physical Chemistry A*, Vol. 101, p. 5111–5116, 1997.

[12] H. Edelsbrunner, *Algorithms in Combinatorial Feometry*, Springer-Verlag, Berlin, 1987.

[13] K. Fukuda and A. Prodon *Double Description Method Revisited*, Institute for Operations Research ETHZ Zürich and Department of Mathematics EPFL Lausanne, Switzerland, 1996.

[14] J.E. Goodman, J. O'Rourke, C.D. Tóth, *Handbook of Discrete and Computational Geometry*, Third edition, CRC Press, Boca Raton, FL, 2017.

[15] T. de Graaf, Robust Mean-Variance Optimization, Master Thesis, Leiden University & Ortec Finance, 2016.

[16] T. de Graaf and M. van der Schans, Robust optimization by constructing near-optimal portfolios, Methodological working paper, Ortec Finance Research Center, 2018.

[17] B. Grünbaum, *Convex Polytopes*, 2nd edition, V. Kaibel, V. Klee, G.M. Ziegler, Springer-Verlag, New York, 2003.

[18] S. Ciliberti, I. Kondor, and M. Mézard, On the feasibility of portfolio optimization under expected shortfall, *Quantitative Finance* Vol. 7 Issue 4, pp. 389-–396.

[19] S.G. Johnson, The NLopt nonlinear-optimization package, http://ab-initio.mit.edu/nlopt.

[20] E. Jones, E. Oliphant,P. Peterson, et al, SciPy: Open Source Scientific Tools for Python, http://www.scipy.org/, 2001–2018.

[21] M. Joswig, *Beneath-and-Beyond Revisited*, Technische Universität Berlin, Institut für Mathematik, 2002.

[22] D. Kraft *A Software Package for Sequential Quadratic Programming*, Deutsche Forschungs- und Versuchsanstalt für Luft-und Raumfahrt, Oberpfaffenhofen, 1988.

[23] I. Kondor, G. Nagy, S. Pafka, Noise sensitivity of portfolio selection under various risk measures, *Journal of Banking & Finance* Vol. 31 Issue 5, pp. 1545-–1573.

[24] O. Ledoit and M. Wolf, "Honey, I Shrunk the Sample Covariance Matrix", *The Journal of Portfolio Management*, Vol. 30, Issue 4, pp. 110–119, 2004.

[25] J.A. De Loera, J. Rambau, F. Santos, *Triangulation*, Springer-Verlag, Berlin, 2010.

[26] H. Markowitz, Portfolio selection, *The Journal of Finance*, Vol. 7, Issue 1, pp. 77-–91, 1952.

[27] A. Meucci, Managing Diversification *Bloomberg Education & Quantitative Research and Education Paper*, pp. 74–79, 2009.

[28] V. DeMiguel et al., A Generalized Approach to Portfolio Optimization: Improving Performance by Constraining Portfolio Norms, *Management Science*, Vol. 55, Issue 5, pp. 798–812, 2009.

[29] R.O. Michaud, The Markowitz Optimization Enigma: Is 'Optimized' Optimal?, *Financial Analysts Journal*, Vol. 45, Issue 1, pp. 31–42, 1989.

[30] D. Persson and W. Woerheide, An Index of Portfolio Diversification, *Financial Services Review*, Vol. 2, Issue 2, pp. 73–85, 1993.

[31] G. Pola, Is your portfolio effectively diversified? Various perspectives on portfolio diversification, Working paper, Amundi, 2014.

[32] M.J.D. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in: *Advances in Optimization and Numerical Analysis*, eds. S. Gomez and J.P. Hennart, Kluwer Academic, Dordrecht, 1994, p. 51–67.

[33] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[34] F.R.Q Reyna et al., Optimal Portfolio Structuring in Emerging Stock Markets Using Robust Statistics, *Brazilian Review of Econometrics*, Vol. 25, Issue 2, pp. 139, 2005.

[35] G. van Rossum, *Python tutorial, Technical Report CS-R9526*, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, 1995.

[36] A.M. Rudin and J.S. Morgan, A Portfolio Diversification Index, *The Journal of Portfolio Management*, Vol. 32, Issue 2, p. 81–89, 2006.

[37] G. Schäfer, Discrete Optimization, Lecture Notes, Center for Mathematics and Computer Sciences, 2017.

[38] K. Schöttle, R. Werner, and R. Zagst, Comparison and robustification of Bayes and Black-Litterman models, *Math. Methods Oper. Res.*, Vol. 71, Issue 4, pp. 453–475, 2010.

[39] R.L. Smith, Efficient Monte Carlo Procedures for Generating Points Uniformly Distributed Over Bounded Regions, *Oper. Res.*, Vol. 32, No. 6, pp. 1296–1308, 1984.

[40] G.M. Ziegler *Graduate Texts in Mathematics – Lectures on Polytopes*, Seventh Printing of the First Edition, Springer, New York, 1995.

[41] *https://github.com/stevengj/nlopt/issues/182.*