Journal of Statistical Software

May 2012, Volume 47, Issue 13. *http://www.jstatsoft.org/*

An Exact Algorithm for Weighted-Mean Trimmed Regions in Any Dimension

Pavel Bazovkin
Universität zu Köln

Karl Mosler
Universität zu Köln

Abstract

Trimmed regions are a powerful tool of multivariate data analysis. They describe a probability distribution in Euclidean d-space regarding location, dispersion, and shape, and they order multivariate data with respect to their centrality. Dyckerhoff and Mosler (2011) have introduced the class of weighted-mean trimmed regions, which possess attractive properties regarding continuity, subadditivity, and monotonicity.

We present an exact algorithm to compute the weighted-mean trimmed regions of a given data cloud in arbitrary dimension d. These trimmed regions are convex polytopes in $\mathbb{R}^d$. To calculate them, the algorithm builds on methods from computational geometry. A characterization of a region's facets is used, and information about the adjacency of the facets is extracted from the data. A key problem consists in ordering the facets. It is solved by the introduction of a tree-based order, by which the whole surface can be traversed efficiently with the minimal number of computations. The algorithm has been programmed in C++ and is available as the R package **WMTregions**.

Keywords: central regions, data depth, multivariate data analysis, convex polytope, computational geometry, algorithm, C++, R.

1. Introduction

Trimmed regions are a powerful tool of multivariate data analysis. They describe a probability distribution in Euclidean d-space regarding location, dispersion, and shape, and they order multivariate data with respect to their centrality. Given d-variate data $x_1, x_2, \ldots, x_n$, an *α-trimmed region* $D_\alpha(x_1, x_2, \ldots, x_n)$ is a convex compact set in $\mathbb{R}^d$ that depends on the data in an affine equivariant way, i.e., for every matrix $A \in \mathbb{R}^{m \times d}$ and every $b \in \mathbb{R}^m$ it holds

$$D_\alpha(Ax_1 + b, \ldots, Ax_n + b) = AD_\alpha(x_1, \ldots, x_n) + b\,.$$

The parameter α varies in an interval such that the family $(D_\alpha(x_1, x_2, \ldots, x_n))_\alpha$ is nested

decreasing in α, i.e., $\alpha < \beta$ implies $D_\beta(x_1, \ldots, x_n) \subset D_\alpha(x_1, \ldots, x_n)$. The smallest region is regarded as a particular median of the data.

Several special notions of trimmed regions have been introduced in the literature, among them the *Mahalanobis regions*, the *halfspace regions*, and the *zonoid regions*; for recent surveys, see Serfling (2006), Cascos (2009). Applications include multivariate data analysis (Liu, Parelius, and Singh 1999), classification (Mosler and Hoberg 2006), tests for multivariate location and scale (Dyckerhoff 2002), risk measurement (Cascos and Molchanov 2007), and many others. The various notions of trimmed regions differ in properties like continuity, robustness, and sensitivity regarding the data. Depending on the type of application different properties are relevant. E.g., Mahalanobis regions are ellipses around the mean of the data and based on their covariance matrix; by this they can neither reflect a possible asymmetry of the distribution nor characterize it in a unique way. Both halfspace regions and zonoid regions reflect asymmetries and characterize the distribution. Halfspace regions are more robust against outliers than zonoid regions; if robustness is an issue, the latter need some preprocessing of the data.

Dyckerhoff and Mosler (2011) have introduced the class of weighted-mean trimmed regions, which possess additional attractive properties and include the zonoid regions as a special case. Weighted-mean trimmed regions are continuous in the data as well as in the parameter, which means that both the function $(x_1, \ldots, x_n) \mapsto D_\alpha(x_1, \ldots, x_n)$ and the function $\alpha \mapsto D_\alpha(x_1, \ldots, x_n)$ are continuous in terms of Hausdorff convergence of sets. Moreover, weighted-mean trimmed regions are subadditive and monotone, which properties have a substantial interpretation in terms of d-variate risk and allow the construction of set-valued risk measures that are coherent (Cascos and Molchanov 2007).

To be useful in data applications, a notion of trimmed regions must be computable. Bivariate trimmed regions of any type can be calculated by constructing a *circular sequence*, like in Dyckerhoff (2000) and Cascos (2007), but only a few procedures are known in dimension $d > 2$. Mahalanobis regions are easily determined in any dimension d, as they only employ the mean and the dispersion matrix of the data. Mosler, Lange, and Bazovkin (2009) develop an efficient geometric algorithm for zonoid regions in any dimension, and Hallin, Paindaveine, and Šiman (2010) provide a parametric linear program for calculating halfspace regions.

In this paper we present an exact algorithm to compute the weighted-mean trimmed regions of a given data cloud in arbitrary dimension d. These trimmed regions are convex polytopes in $\mathbb{R}^d$. To calculate them, the algorithm builds on methods from combinatorial and computational geometry. A region's facet is characterized by $d-1$ pairs of data points. Based on them the normal (support vector) of the facet is determined and by properly rotating the support vector an adjacent facet is found. A key problem consists in ordering the facets. It is solved by the introduction of a tree-based order.

Overview of the paper: Section 2 provides a brief introduction into the notion of weighted-mean trimmed (WMT) regions. The main results of the paper are contained in Section 3, which presents the basic geometrical ideas of the algorithm, in particular the construction of a facet on the basis of $d-1$ data point differences and the transition to a neighboring facet by rotating the support vector and exchanging the basis. Section 4 provides a formal description of the algorithm with the analysis of its complexity. Section 5 delineates the R package and the program realization in C++, while Section 6 provides conclusions and a discussion of perspectives.

2. Weighted-mean trimming

This section reviews the general notion of weighted-mean trimmed regions and two of its special cases, the zonoid regions and a modified version of the expected convex hull (ECH) regions – the ECH* regions.

2.1. Definition and principal properties

Weighted-mean trimmed regions are convex bodies in $\mathbb{R}^d$. Recall that a convex body $K \subset \mathbb{R}^d$ is uniquely determined by its support function h_K (see, e.g., Rockafellar 1970),

$$h_K(p) = \max\left\{p^\top x \,|\, x \in K\right\}, \quad p \in S^{d-1},$$

where S^{d-1} denotes the unit sphere in $\mathbb{R}^d$.

To define the weighted-mean α-trimmed region of a given data cloud $x_1, x_2, \ldots, x_n$, we construct its support function as follows: For $p \in S^{d-1}$, consider the subspace $\{\lambda p | \lambda \in \mathbb{R}\}$. By projecting the data on this subspace a linear ordering is obtained,

$$p^\top x_{\pi_p(1)} \leq p^\top x_{\pi_p(2)} \leq \cdots \leq p^\top x_{\pi_p(n)}\,, \tag{1}$$

and, by this, a permutation π_p of the indices $1, 2, \ldots, n$. Note that, if no equalities arise in (1), the permutation π_p is unique, otherwise a class Π_p of several permutations is generated. The set of directions p at which π_p is not unique will be denoted $H(x_1, \ldots, x_n)$,

$$H(x_1, \ldots, x_n) = \left\{p \in S^{d-1} \,|\, \text{there are } i \neq j \text{ such that } p^\top x_i = p^\top x_j\right\}.$$

Consider weights $w_{j,\alpha}$ for $j \in \{1, 2, \ldots, n\}$ and $\alpha \in [0, 1]$ that satisfy the following restrictions (i) to (iii):

(i) $\sum_{j=1}^n w_{j,\alpha} = 1$, $w_{j,\alpha} \geq 0$ for all j and α,

(ii) $w_{j,\alpha}$ increases in j for all α,

(iii) if $\alpha < \beta$ then

$$\sum_{j=1}^{k} w_{j,\alpha} \leq \sum_{j=1}^{k} w_{j,\beta}\,, \quad k = 1, \ldots, n\,. \tag{2}$$

Then, as it has been shown in Dyckerhoff and Mosler (2011), the function $h_{D_\alpha(x_1,\ldots,x_n)}$,

$$h_{D_\alpha(x_1,\ldots,x_n)}(p) = \sum_{j=1}^{n} w_{j,\alpha} p^\top x_{\pi_p(j)}\,,\ p \in S^{d-1} \tag{3}$$

is the support function of a convex body $D_\alpha = D_\alpha(x_1, \ldots, x_n)$, and $D_\alpha \subset D_\beta$ holds whenever $\alpha > \beta$.

Now we are ready to give the general definition of a family of weighted-mean trimmed regions.

Definition 1. *(Dyckerhoff and Mosler) Given weights $w_{1,\alpha}, \ldots w_{n,\alpha}$ that satisfy the restrictions (i) to (iii), the convex body $D_\alpha = D_\alpha(x_1, \ldots, x_n)$ having support function (3) is named the* weighted-mean α-trimmed region *of $x_1, \ldots, x_n$, $\alpha \in [0, 1]$.*

The next proposition explains the name by stating that a weighted-mean trimmed region is the convex hull of weighted means of the data. Further it describes the region's extreme points.

Proposition 1. *It holds*

$$D_\alpha(x_1,\dots,x_n) = conv\left\{\sum_{j=1}^{n} w_{j,\alpha} x_{\pi(j)} \,\middle|\, \pi \text{ permutation of } \{1,\dots,n\}\right\}, \tag{4}$$

and the set of extreme points of D_α is given by

$$Ext\big(D_\alpha(x_1,\dots,x_n)\big) = \left\{\sum_{j=1}^{n} w_{j,\alpha} x_{\pi_p(j)} \,\middle|\, p \in S^{d-1} \setminus H(x_1,\dots,x_n)\right\}. \tag{5}$$

Due to their attractive analytical properties, WMT regions are useful statistical tools. Besides being *continuous* in the data and in α, they are *subadditive*, that is,

$$D_\alpha(x_1+y_1,\dots,x_n+y_n) \subset D_\alpha(x_1,\dots,x_n) \oplus D_\alpha(y_1,\dots,y_n),$$

and *monotone*: If $x_i \le y_i$ holds for all i (in the componentwise ordering of $\mathbb{R}^d$), then

$$\begin{aligned} D_\alpha(y_1,\dots,y_n) &\subset D_\alpha(x_1,\dots,x_n) \oplus \mathbb{R}^d_+, \quad \text{and} \\ D_\alpha(x_1,\dots,x_n) &\subset D_\alpha(y_1,\dots,y_n) \oplus \mathbb{R}^d_-, \end{aligned}$$

where $\oplus$ signifies the Minkowski sum of sets. For proofs and more results, like projection properties, the reader is again referred to Dyckerhoff and Mosler (2011).

2.2. Special notions of weighted-mean trimming

The general notion of WMT regions provides a flexible approach to the trimming of multivariate data. Depending on the choice of the weights $w_{j,\alpha}$ different families of trimmed regions are obtained. They include the zonoid regions (Koshevoy and Mosler 1997), the ECH and ECH* regions (Cascos 2007), the geometrically trimmed regions, and many others. For an illustration in dimension $d = 3$ see Figure 1. Here the left panel shows zonoid regions for different parameters α, while the right one consists of ECH* regions for the same data and α. Note from Figure 1 that the surface of a zonoid region appears to have less facets than an ECH* region.

Historically, the first type of WMT regions was *zonoid trimmed regions* $ZD_\alpha(x_1,\dots,x_n)$ for $0 < \alpha \le 1$ proposed by Koshevoy and Mosler (1997),

$$ZD_\alpha(x_1,\dots,x_n) = \left\{\sum_{i=1}^{n} \lambda_i x_i \,\middle|\, 0 \le \lambda_i \le \frac{1}{n\alpha}, \sum_{i=1}^{n} \lambda_i = 1\right\}.$$

The corresponding support function is

$$h_{ZD_\alpha}(p) = \sum_{j=1}^{n} w_{j,\alpha} p' x_{\pi_p(j)},$$

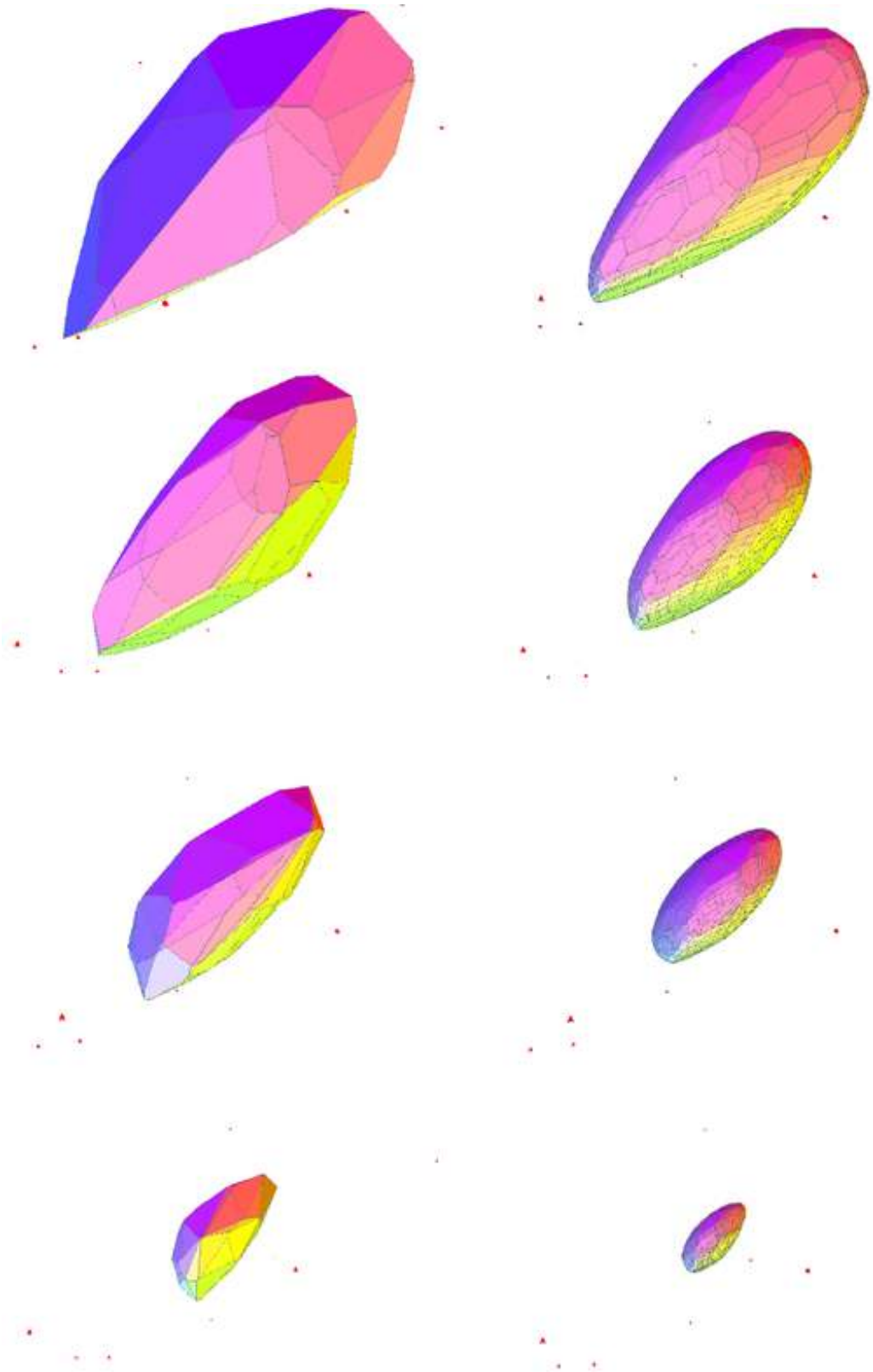

Figure 1: Examples of WMT regions in $\mathbb{R}^3$. Representation of the zonoid (left) and ECH* (right) regions for the same data and depths.

with weights

$$w_{j,\alpha} = \begin{cases} 0 & \text{if } j < n - \lfloor n\alpha \rfloor\,, \\ \frac{n\alpha - \lfloor n\alpha \rfloor}{n\alpha} & \text{if } j = n - \lfloor n\alpha \rfloor\,, \\ \frac{1}{n\alpha} & \text{if } j > n - \lfloor n\alpha \rfloor\,. \end{cases} \tag{6}$$

Many properties of the zonoid regions are developed in Mosler (2002); particularly important is that they contain full information about the data.

Another important notion of WMT regions is that of ECH* regions (Cascos 2007). Their support function

$$h_{\mathrm{ECH}^*_\alpha}(p) = \sum_{j=1}^{n} w_{j,\alpha}\, p^\top x_{\pi_p(j)}$$

employs the weights

$$w_{j,\alpha} = \frac{j^{1/\alpha} - (j-1)^{1/\alpha}}{n^{1/\alpha}}\,. \tag{7}$$

For a detailed discussion of these and other special weighted-mean trimmed regions, like ECH and geometrically trimmed regions, the reader is referred to Dyckerhoff and Mosler (2011).

3. Geometry of the algorithm

In this section we present the basic ideas of the algorithm. Specifically, it relies on notions from convex geometry.

3.1. Trimmed region as a convex polytope

Consider a data cloud, which is a finite set of data points, $\{x_1, x_2, \ldots, x_n\} \subset \mathbb{R}^d$, and assume that the points are all different and in *general position* (i.e., no more than $d-1$ of them lie on the same hyperplane).

For given α, the α-trimmed region $D_\alpha = D_\alpha(x_1, x_2, \ldots, x_n)$ is a convex polytope in $\mathbb{R}^d$ that is bounded and closed. Such a polytope is the nonempty and bounded intersection of finitely many closed halfspaces. Thus the polytope can be completely described by its bounding hyperplanes. The intersection of a bounding hyperplane with the polytope is named a *facet* if it has the affine dimension $d-1$. Similarly, it is named a *ridge* if it has the dimension $d-2$. In dimension $d = 3$ a facet is a *face*, and a ridge is an *edge*.

In the sequel, we calculate the weighted-mean trimmed regions by their facets. Two computational tasks will have to be repeatedly performed:

1. Calculate a facet.
2. Find an adjacent facet.

A ridge is the intersection of two facets. Therefore, investigating the ridges is a way to extract information about the adjacency of facets. Each ridge of a given facet provides an indicator whether another facet is adjacent or not. A bounding hyperplane is fully described by its (outwards pointing) normal and one additional point, in particular one of its vertices. Hence, for every facet we determine its normal and a vertex as well as the adjacency indicator of

each of its ridges. By doing this successively for all facets, a complete representation of the trimmed region is obtained.

Mosler, Lange, and Bazovkin (2009) develop an exact algorithm for calculating zonoid trimmed regions. They demonstrate that, in the case of zonoid regions, the normal of a facet is characterized by d points of the data cloud.

Regarding a general WMT region, we will firstly characterize its facets. Let F be a given facet of $D_\alpha(x_1, \ldots, x_n)$ and p denote its normal. Then F has at least d vertices, which all are supported by p. Due to (3) and (4) each vertex v has the form

$$v = \sum_{j=1}^{n} w_{j,\alpha} x_{\pi_p(j)} \quad \text{with some } \pi_p \in \Pi_p \,. \tag{8}$$

Consequently, not all $p'x_i$ can be different: It holds $p \in H(x_1, \ldots, x_n)$, and Π_p has at least d elements. Now let us consider the p-ordered series of indices

$$\pi_p(1), \pi_p(2), \ldots, \pi_p(n) \,.$$

In the sequel we will mention those pairs of indices $(\pi_p(j), \pi_p(k))$ as *active* that satisfy the equation $p'x_{\pi_p(k)} = p'x_{\pi_p(j)}$ plus a restriction on their weights w_j and w_k, which will be specified below. The equation means that the difference $x_{\pi_p(k)} - x_{\pi_p(j)}$ is orthogonal to p,

$$x_{\pi_p(k)} - x_{\pi_p(j)} \perp p \,. \tag{9}$$

At a given p, all indices that belong to an active pair will be mentioned as *active indices*, all others as *passive indices*.

From now on, we will distinguish *data points* and *data vectors*. By a data vector we mean the difference of two data points. To determine p, $d-1$ data vectors are needed. Each of them is based on an active pair of indices and thus satisfies the orthogonality relation (9). As, by assumption, the data are in general position, any such $d-1$ data vectors are linearly independent. They will be mentioned as a *basis* of F and denoted by $\mathcal{V}_F$. Note that the basis of a facet is not unique: To form a basis, out of all active pairs of indices any $d-1$ pairs that yield linearly independent data vectors may be chosen. To summarize:

Theorem 1. (Basis of a facet) *The normal of a facet F is orthogonal to exactly $d-1$ linearly independent data vectors, which form a basis of F. The facet is characterized by a basis and one of its vertices.*

Next we develop the two essential steps of calculating a facet and finding an adjacent facet in detail.

Task 1: Calculating a facet

In our algorithm we have to construct a basis for each facet of the polytope. Let p be the normal of a given facet F, choose some $\pi_p \in \Pi_p$, and consider the series of p-ordered indices $\pi_p(1), \pi_p(2), \ldots, \pi_p(n)$. This series contains $d-1$ active pairs of indices, $\pi_p(j), \pi_p(k)$, that define a basis $\mathcal{V}_F$.

The special case of zonoid regions (having weights (6)) appears to be particularly simple: A facet is identified by exactly d data points (carrying serially p-ordered indices), which yield

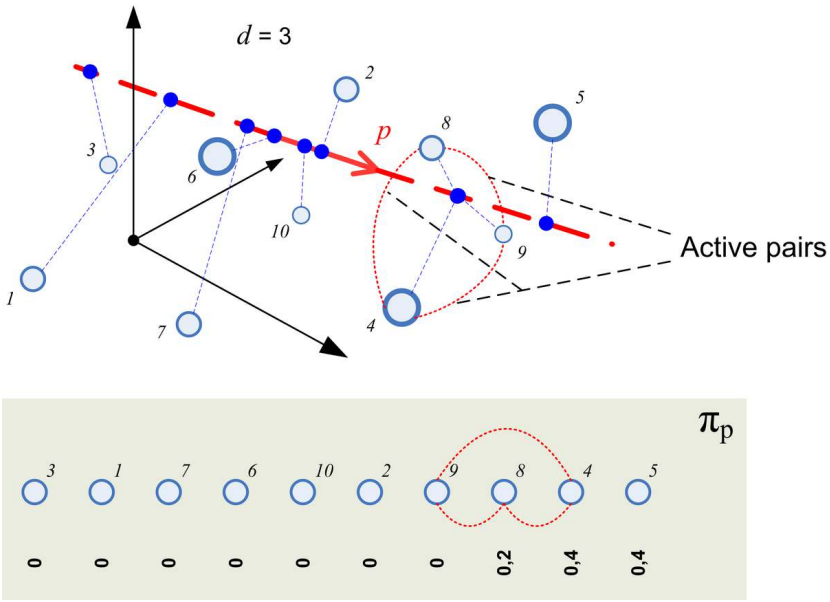

Figure 2: Characterizing the normal p of a facet (zonoid region, $d = 3, n = 10, \alpha = 0.25$): Data points and their projections; p-ordered indices; weights; active pairs of indices.

$d-1$ linearly independent difference vectors that are orthogonal to p (Mosler *et al.* 2009). As an example, Figure 2 exhibits ten points in $\mathbb{R}^3$ and their projections to the line generated by p. The lower panel contains the p-ordered series of indices and the weights (6) for $\alpha = 0.25$. Here, three indices (9, 8, and 4) are active, as well as three index pairs ((9, 8), (9, 4), and (8, 4)). A basis of the facet is given, e.g., by the data vectors $x_8 - x_9$ and $x_4 - x_8$. Note that for these weights (at $\alpha = 0.25$) and *any* direction p the indices $\pi_p(7), \pi_p(8)$, and $\pi_p(9)$ become the active ones.

Other types of weighted-mean trimmed regions employ less simple weights. With them the number of active indices involved in the identification of a facet F may be larger than d. E.g., Figure 3 illustrates the characterization of a facet of an ECH* region, with weights (7) and $\alpha = 0.25$. It shows another example of ten points in $\mathbb{R}^3$ and their projections, given some p. In this example, four indices (7, 6, 4, and 2), and two index pairs ((7, 6) and (4, 2)) are active, and a basis consists of $x_6 - x_9$ and $x_2 - x_4$, being unique up to sign.

In general, we consider the following disjoint blocks $\mathcal{A}_l$ of indices, $l = 1, \ldots, L$,

$$\mathcal{A}_l = \{\pi_p(i) \,|\, i \in \{a_l, a_l + 1, \ldots, a_l + n_l - 1\},\ p' x_{\pi_p(i-1)} = p' x_{\pi_p(i)} \text{ for } i > a_l\}\,,$$

where $a_{l-1} < a_l$ holds ($a_0 = 0$), and define: A pair of indices is called *active* if a block $\mathcal{A}_l$ exists that contains both of them. In particular, each block contains at least two elements, $n_l \geq 2$, and it holds $w_{a_l,\alpha} < w_{a_l+n_l-1,\alpha}$, which is the restriction on weights announced above. Moreover, $\mathcal{A}_l \cap \mathcal{A}_m = \emptyset$ if $l \neq m$, and

$$\mathcal{V}_F = \bigcup_{l=1}^{L} \{x_{\pi_p(i)} - x_{\pi_p(i+1)} \,|\, \pi_p(i), \pi_p(i+1) \in \mathcal{A}_l\}\,.$$

Note that in the case of zonoid regions only one block of active indices arises; it holds $L = 1$.

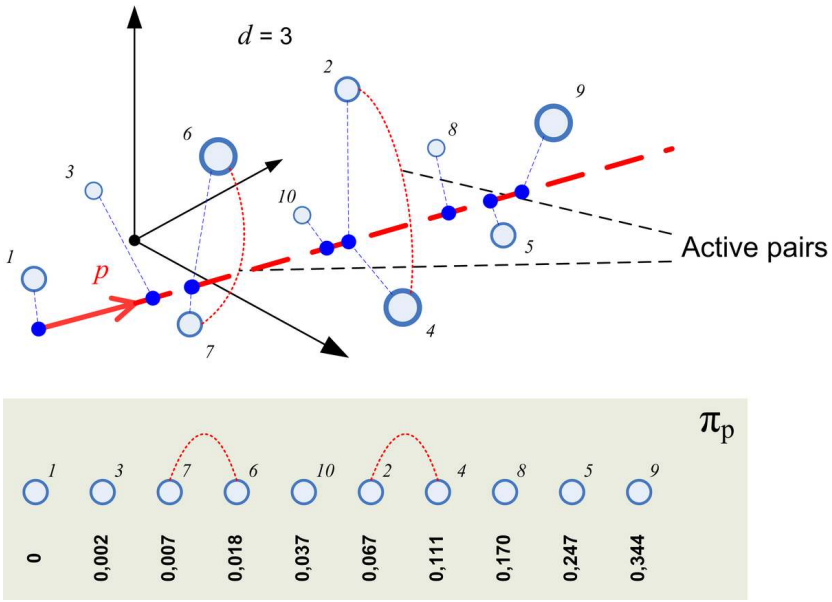

Figure 3: Characterizing the normal p of a facet (ECH* region, $d = 3, n = 10, \alpha = 0.25$): Data points and their projections; p-ordered indices; weights; active pairs of indices.

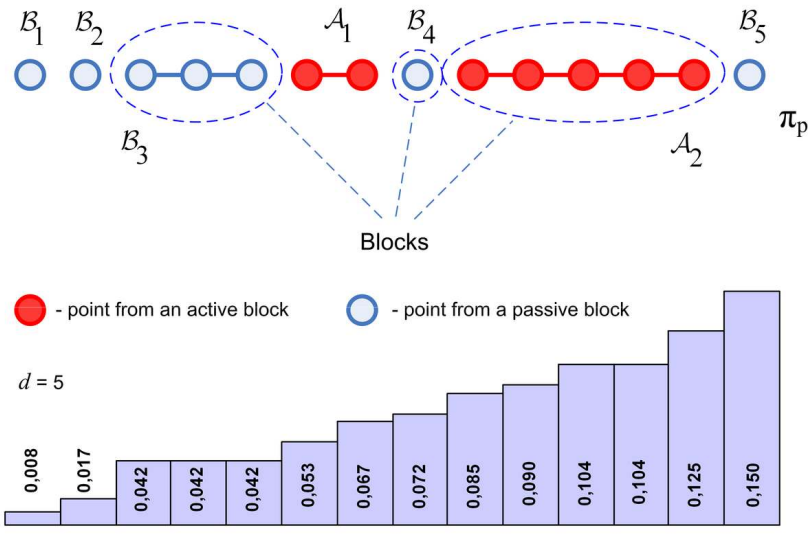

Figure 4: Series of blocks of active and passive indices; weights as indicated.

The remaining indices, which are not in $\bigcup_{\ell=1}^{L} \mathcal{A}_l$, are the *passive* ones. Among them we distinguish disjoint blocks that have equal weights,

$$\mathcal{B}_k = \{\pi_p(i) \,|\, i \in \{b_k, b_k + 1, \ldots, b_k + m_k - 1\},\ w_{i-1,\alpha} = w_{i,\alpha} \text{ for } i > b_k\}\,,$$

$k = 1, 2, \ldots, K$, while $m_k \geq 1$, $b_{k-1} < b_k$ with $b_0 = 0$, and $w_{b_{k-1},\alpha} < w_{b_k,\alpha}$.
Thus $\pi_p(1), \pi_p(2), \ldots, \pi_p(n)$ divides into a series $\mathcal{S}_F$ of blocks

Task 2: Finding an adjacent facet

To identify adjacent facets we start from a given facet F, which has support vector p and which from now on will be called the *current facet*. Each ridge of F offers a way of "jumping" to a neighboring facet. Therefore we investigate the ridges of the current facet F and, consequently, its adjacent facets. Each element of the basis $\mathcal{V}_F$ may be regarded as a reduction of one degree of freedom of the support vector p. To determine p as the normal of the current facet F, we have to reduce $d-1$ degrees of freedom and calculate the uniquely determined support vector p that is orthogonal to $d-1$ linearly independent data vectors (differences of vectors from the original data cloud). A ridge of the current facet is supported by vectors that result from adding one degree of freedom to the given support vector p. The degree of freedom is added by leaving out one of the $d-1$ data vectors from the basis $\mathcal{V}_F$, or, more generally, by replacing some k data vectors in $\mathcal{V}_F$ with some $k-1$ ones, while keeping linear independence within the basis.

Removing one element from the basis $\mathcal{V}_F$ corresponds to splitting one of the active blocks in $\mathcal{S}_F$, say $\mathcal{A}_l$, into $\mathcal{A}_l^1$ and $\mathcal{A}_l^2$. By this, a modified series of blocks, $\mathcal{S}_{F*}$, is obtained. Observe that, if $\mathcal{A}_l^1$ (or $\mathcal{A}_l^2$) is a singleton, its element becomes a passive index in $\mathcal{S}_{F*}$.

Now, the removed element of the basis has to be substituted by another data vector. For this, any pair (i^*, j^*) of indices that belong to two neighboring blocks of $\mathcal{S}_{F*}$ can be chosen and the corresponding data vector $x_{i^*} - x_{j^*}$ be added to the basis. (However, no pair from $\mathcal{A}_l^1 \times \mathcal{A}_l^2$ must be selected.) Then the new basis defines a facet that is adjacent to the current facet F.

This step may be visualized as follows (see Figure 5 for $d = 3$): Starting at p, the support vector is rotated in a plane (of dimension two in $\mathbb{R}^d$) until another data vector enters the basis $\mathcal{V}_F$, i.e., until another data vector becomes orthogonal to p. Let $\mathcal{E}_p$ denote the set of vertices of the polytope corresponding to the support vector p. We turn p until it stops at the position $\tilde{p}$ where $\tilde{p}^\top x_{i^*} = \tilde{p}^\top x_{j^*}$ for some i^* and j^*, i.e., $(x_{i^*} - x_{j^*}) \perp \tilde{p}$. Then, if $\mathcal{E}_{\tilde{p}} \supset \mathcal{E}_p$, this means that $\tilde{p}$ is a normal to some facet $\tilde{F}$ which is a neighbor to the current facet. Otherwise, p is turned further until the condition is met. Obviously, to meet the condition, the indices i^* and j^* must be in different blocks of $\mathcal{S}_{F*}$. On the other hand, indices can continuously interchange places only with their neighbors, that is, x_{i^*} and x_{j^*} must be in blocks that neighbor each other.

So far we have exchanged a single basis vector against another one. However, the elements within each active block at p can be arbitrarily rearranged, and each active index used in the exchange step just *represents* a class of equivalent active indices. Therefore more than one, say k, active pairs living on $\mathcal{A}_l^1 \times \mathcal{A}_l^2$ may be exchanged simultaneously.

As a result of the basis exchange we have found a single adjacent facet. Our next task is to identify *all* facets that are adjacent to the current facet. For this, it is not necessary to enumerate all pairs of indices from neighboring blocks of $\mathcal{S}_{F*}$. Note that the elements of each active block $\mathcal{A}_l$ are equivalent in the p-order, i.e., $p' x_{i^*} = p' x_{j^*}$ for all $i^*, j^* \in \mathcal{A}_l$. Hence, we may permute indices *within the active blocks* in an arbitrary way, which means employing some other permutation from Π_p in place of the given permutation π_p. Therefore, in generating all possible basis exchanges, we need not consider all active indices for pairing, but may restrict to a *representative index* of each active block, say $r_l \in \mathcal{A}_l$, $l = 1, \ldots, L$. However, in the passive blocks, all indices have to be taken into account.

A pair (i^*, j^*) from two neighboring blocks in $\mathcal{S}_{F*}$ is called a *critical pair* if it consists of

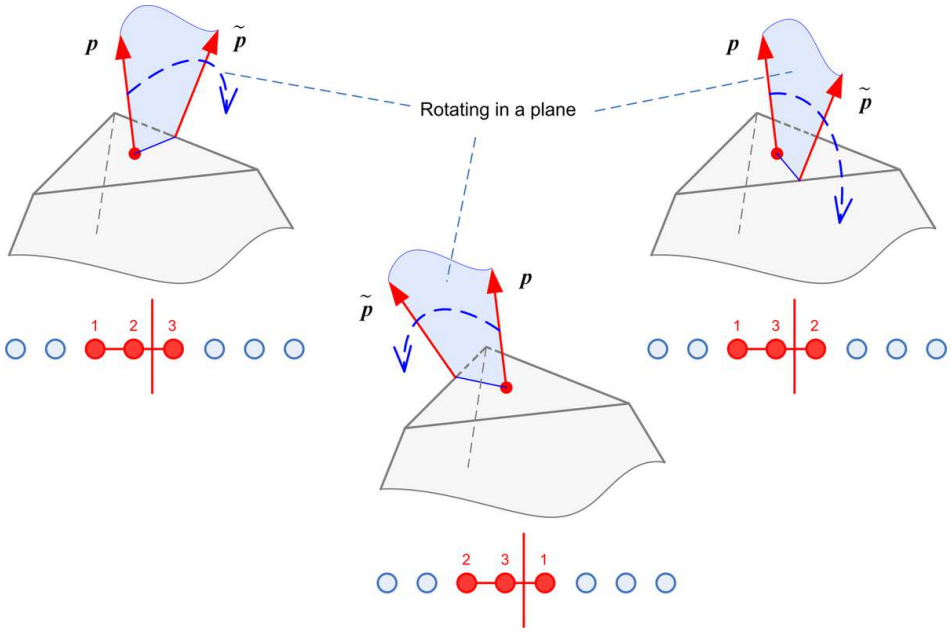

Figure 5: Rotating p in a plane of dimension two in $\mathbb{R}^d$.

indices that are either passive or representative active indices. More formally, we may write the series $\mathcal{S}_{F*}$ of active and passive blocks as

$$\mathcal{S}_{F*} = (\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{L+K})$$

and define

$$\tilde{\mathcal{C}}_m = \begin{cases} \{r_l\} & \text{if } \mathcal{C}_m = \mathcal{A}_l \text{ for some } l, \\ \mathcal{B}_k & \text{if } \mathcal{C}_m = \mathcal{B}_k \text{ for some } k. \end{cases}$$

Then the set of critical pairs (that have to be checked for finding all adjacent facets) is given by

$$\bigcup_{m=1}^{L+M-1} \tilde{\mathcal{C}}_m \times \tilde{\mathcal{C}}_{m+1}\,. \tag{10}$$

The two computational tasks, calculating a facet and finding a neighboring facet, are performed until all facets of the polytope have been visited and computed. As a result of the algorithm, the WMT region is completely described by its facets. Alternatively and in addition, we may be interested in calculating vertices of the polytope. These are easily determined by the following procedure.

Proposition 2. (Calculating vertices) *Consider a facet F having normal p. Each vertex of F exactly corresponds to a permutation of $(\pi_p(1), \ldots \pi_p(n))$ that is restricted to permutations within the $\mathcal{A}_l$.*

Corollary 1. *The minimum possible number of vertices of a facet is d (e.g., for zonoid regions). The maximum possible number of vertices of a facet is $d!$.*

E.g., in the case of ECH*-regions, the number of vertices of a facet varies from d to $d!$.

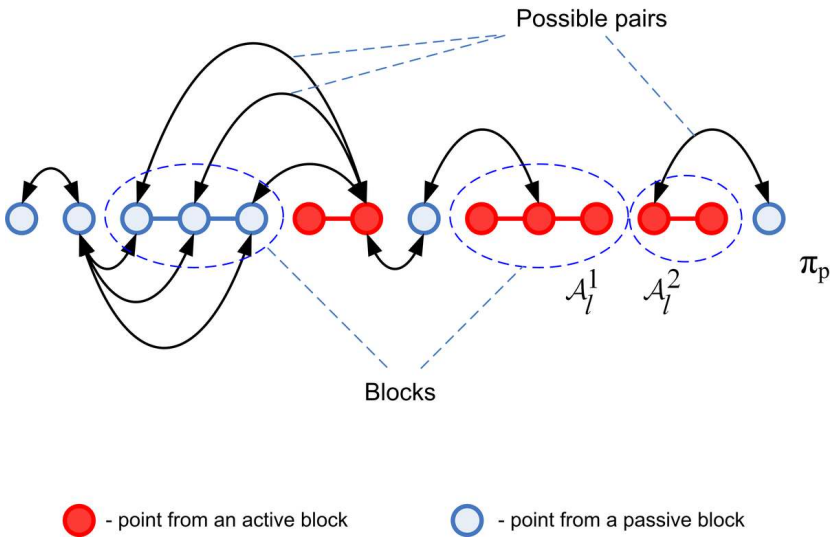

Figure 6: The series $\mathcal{S}_{F*}$ of blocks; with possible critical pairs.

3.2. Spanning tree order

Based on the adjacency information obtained by the above approach we are able to calculate the facets in a sequential order. For this sequence, we use the spanning tree order (STO) discussed in Mosler *et al.* (2009). The STO provides a complete ordering of all facets according to which they are generated in the algorithm. The general idea is:

1. Represent all facets adjacency information by a tree,
2. organize an efficient procedure to traverse the tree.

In the algorithm we apply a breadth-first search like that described in Knuth (1997). Using the STO we generate each facet only once, which is an efficient procedure.

Moreover, as the STO is based on the neighboring relation among facets, we can restrict the calculation of facets to some connected part of the trimmed region's surface, e.g. the part having support vector $p \geq \mathbf{0}$. This proves to be useful in certain applications like multivariate risk measurement.

Note that we calculate the trimmed region by sequentially generating its facets, but not its vertices. In dimension $d = 2$ it is also possible to determine the region by enumerating its vertices; this is done by means of a so called *circular sequence* (Edelsbrunner 1987).

It is easy to see that the proposed procedure applies to any choice of a weighting function satisfying the above WMT restrictions (i) to (iii). Thus the algorithm is able to calculate any weighted-mean trimmed region.

Finally, we would like to turn the reader's attention again to the the adjacency of the sequentially generated vertices. That is a practical advantage because we can restrict the calculation to some specified part of the WMT region which we might only be interested in. In this respect our procedure reminds of the so-called "gift-wrapping" approach, which is used to solve common tasks of constructing convex polytopes, in particular calculating the convex hull of

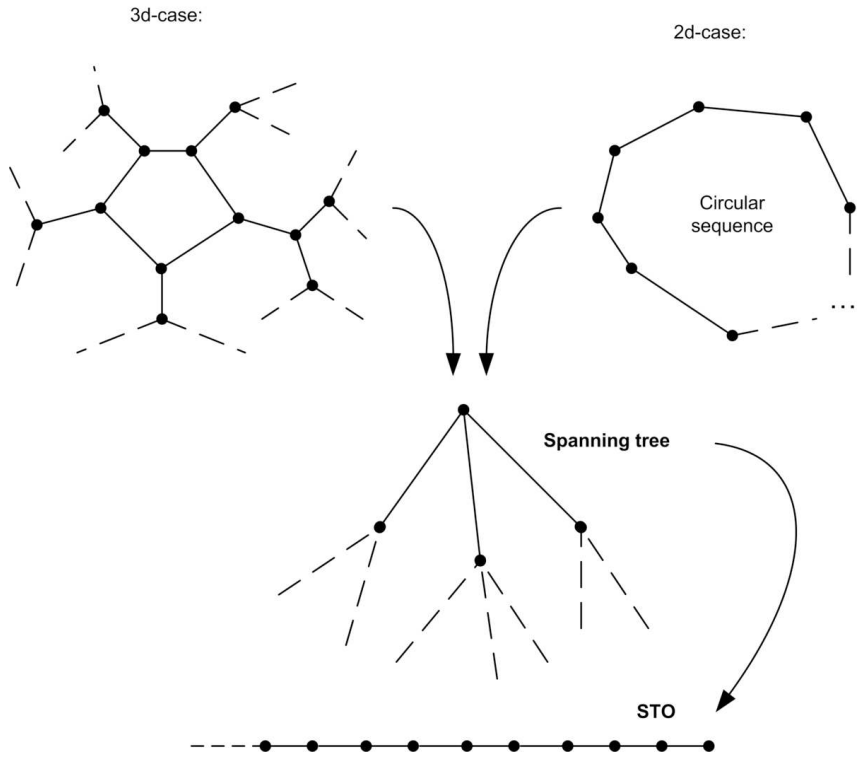

Figure 7: The sample scheme of the procedure.

a given set, where algorithms for higher dimensions have been proposed by Swart (1985) and others.

However, the structure of a WMT region is much more complex, since it aggregates not only local information, as it is the case in the construction of a convex hull, but depends on information on the whole data cloud, including all inner points. If $0 \leq \alpha \leq \frac{1}{n}$ the WMT region turns into a convex hull of data points, which is the trivial case in our task. For this reason our algorithm differs fundamentally from a classical "gift-wrapping" procedure. Other than Swart's and similar approaches we find a facet of the WMT region only once (in contrast to a repeated finding of facets and removing the new one after discovering the duplication), that is, we make no redundant calculations and form a unique chain of facets according to the STO. Furthermore, convex hull algorithms work with a given set of points, while in our problem there is no such set in an explicit form, and facets are constructed without having information on their vertices. Besides this, it was shown above that a facet of a WMT region is, in most cases, no $(d-1)$-dimensional simplex. Actually, the number of vertices can blow up to $d!$, which represents a difficult case for a convex hull algorithm.

4. The algorithm

4.1. Interface and steps

In this subsection we give a formal scheme of the algorithm and an interface to it.

Input

- d (dimension of the data space, $d \geq 2$);
- n (number of data points, $n > d$);
- `cloud` (data $x_1, \ldots, x_n \in \mathbb{R}^d$);
- α (depth parameter);
- w_α (weight vector; alternatively: name of special type of WMT regions).

Output

- `trimmed_region` (all facets of the trimmed region, with coordinates of their vertices);
- *Visualization.*

Steps of the algorithm

A. *Initialization: Read the input.*

B. *Determine a first facet:*

   a. From `cloud`, *form a set* `vec_defining_set` of $d-1$ linearly independent data vectors (= basis).
   b. *Calculate*, to the hyperplane through `vec_defining_set`, a normal vector r.
   c. *Substitute* r for p and choose a permutation $\pi_p \in \Pi_p$. *Determine* the series of active blocks $\{\mathcal{A}_l\}_{l=1\ldots L}$ in this permutation.
   d. $\{\mathcal{A}_l\}_{l=1\ldots L}$ defines `vec_defining_set` and, hence, the first facet `ffacet`.
   e. *Place* `ffacet` $\hookrightarrow$ the head of `queue`.

   Having the initial facet, we can start a sequential calculation of all others.

C. *Determine all facets:*

   a. *Take* `curr_facet` $\hookleftarrow$ front of the `queue`.
   b. *Create* neighboring facets of `curr_facet`.
      I. *Create all ridges* by adding a degree of freedom to p (reducing cardinality of the basis `vec_defining_set` by one).
         i. Take the next $\mathcal{A}_l$ and create all possible splittings of it into two subsets: $\langle \mathcal{A}_l^1, \mathcal{A}_l^2 \rangle$. *Replace* $\{\mathcal{A}_l\}$ by $\mathcal{A}_l^1$ and $\mathcal{A}_l^2$. *If* either $\mathcal{A}_l^1$ or $\mathcal{A}_l^2$ is a singleton, *remove it* from the active blocks. A set `partial_facets(l)` is obtained.
         ii. *Drop off* all elements of `partial_facets(l)` that are no active blocks. A set `ridges(l)` is obtained.
         iii. *Add* `partial_facets(l)` to the set `ridges`. *If* an unprocessed $\mathcal{A}_l$ is left, go to C.(b.)I.i.

Now we have found all ridges and are ready to do "jumps" to neighboring facets. Note that the procedure jumps *only to new facets*. In doing this, we process each facet twice: First, we only preprocess it by marking its ridges; second, we do a normal calculation of the "jumps".

II. For the next ridge in `ridges` do the following:
   i. *If* `curr_facet` is not `preprocessed`, *calculate a hash code* of the ridge and mark it ↪ `hash_table`. Then go to C.(b.)II.
   ii. *Check* in `hash_table`, whether the ridge *is blocked*. *If* yes, go to C.(b.)II.
   iii. *Build* the maximum number of linearly independent data vectors that are based on active pairs. *Put* the vectors as rows into a matrix A. There will be $d-2$ rows.
   iv. Given a normal vector r to `curr_facet`, *put it* as an additional row into A. *Put* any non-zero vector that is linearly independent of the $d-1$ previously chosen rows as a last row into A. Let b be a vector that consists of $d-1$ zeros and a last non-zero entry.
   v. *Solve* the linear equation $A\,z = b$. Its solution z and r span a plane B_2 that is orthogonal to the ridge.
   vi. *Calculate critical pairs* according to (10).
   vii. *Rotate* p in the plane B_2. In doing so, start at $p = r$ and *move* p in a way that the new ordering of points in the permutation corresponds to the previous splitting of an active block.
   viii. *Stop if* p becomes orthogonal to some vector built on a critical pair of indices. *Take this vector* as `new_vector`.
   The neighboring facet is discovered. Now we have to reconstruct its combinatorial stucture.
   ix. *Add* `new_vector` to `vec_defining_set`. `new_facet` is obtained. The current position of p is a normal r to `new_facet`.
   If `new_vector` is built on indices from an active block $\mathcal{A}_j$ and a neighboring passive block, then *augment* $\mathcal{A}_j$ with the passive index.
   If `new_vector` is built on indices from two active blocks, $\mathcal{A}_j$ and $\mathcal{A}_{j+1}$, then *merge* these two blocks.
   If `new_vector` is built on two passive indices, then a new block $\mathcal{A}_j^\top$ is created having them as its two elements.
   x. *Place* `new_facet` ↪ the head of `queue`.

   We have to mark the ridges of the facet immediately, thus preventing another "jump" to the facet. The immediate processing is enabled by putting the new facet to the head of the queue.

III. *If* `curr_facet` is not `preprocessed`, *label it* as `preprocessed` and *place* ↪ `queue`. Then, go to C.a.

IV. For `curr_facet`, *calculate* the vertices and its absolute distance from the origin by (8).

V. *Shift* `curr_facet` by $\overline{x}$ and *transfer it* to `trimmed_region`.

c. *If* `queue` is not empty, go to C.a. Otherwise, *stop*: Then, `trimmed_region` contains all facets of the trimmed region.

We would also direct a reader's attention to three special features of the algorithm:

1. *Using a "double-hash":* The ridges are hashed using a "double-hash" table. That is, a ridge is `blocked` if it has been marked twice.

2. *Calculating the hash code:* The hash code is calculated by creating a bit row from integer numbers describing $\{\mathcal{A}_l\}_{l=1\ldots L}$ and one number describing the absolute position of a ridge (to distinguish parallel ridges).

3. *Determining all adjacent facets:* For optimizing the complexity at this stage the mutual information concerning all ridges can be used. The details of such a heuristic are described in the Appendix A.

4.2. Complexity of the algorithm

Due to the mechanism of the "double-hash" the algorithm has as many loops as the WMT region has facets. Obviously, this is the minimum number of facet generating loops in this sort of algorithms.

At each facet F we have to calculate the normal of the facet and its distance from the origin. Further we have to determine all neighboring facets. This is done by solving linear equations and calculating inner products only. We have shown above that the complexity of this algorithmic loop amounts to $O(d^2 \cdot R(F))$, where $R(F)$ is the number of ridges of the facet. $R(F)$ can vary between d and 2^{d-1}, depending on the type of the WMT region and on α.

To obtain a rough conservative estimate of $R(F)$, we may proceed as follows: First, note that $R(F)$ is bounded by $\bar{R} = \prod_{l=1}^{L} 2^{n_l-1}$. Then suppose that the active blocks $\mathcal{A}_l$ have about equal size and that their number L, as a first approximation, is proportional to the dimension d, say $L \approx d/c$. Under these assumptions $\bar{R} \approx L \cdot 2^{d/L-1} \approx d/c \cdot 2^{c-1}$, that is, $R(F)$ is approximately bounded by the dimension d multiplied with a constant $K = 2^{c-1}/c$ that does not depend on the dimension.

Searching for all neighbors of a facet, we have to calculate n inner products, which gives complexity $O(nd)$. Hence, the complexity of one facet generating loop is described by $O((d^2 + nd) \cdot \bar{R}) \cong O(d^2 n \cdot K)$, since $n > d$. If the average number of facets is denoted by $N(n, d)$, the average computational complexity of the algorithm amounts to $O((d^2 + nd) \cdot \bar{R} \cdot N(n, d)) \cong O(d^2 n \cdot K \cdot N(n, d))$.

For a better understanding of $N(n, d)$ we like to discuss the number of vertices $V(n, d)$ of the WMT region. It is maximal when all weights in the weight vector are distinct. Let us consider hyperplanes that are orthogonal to all data vectors and intersect at the origin. Then the $\mathbb{R}^d$ is split by the hyperplanes into convex cones, and there will be a bijection between the vertices of the WMT region and these convex cones (cf. *direction domains* in Mosler *et al.* 2009). Winder (1966) has shown that the number of such cones equals $2\sum_{i=0}^{d-1}\binom{m-1}{i}$ for m hyperplanes, which is $O(m^d)$. We have at most $\frac{n(n-1)}{2}$ hyperplanes (for zonoid regions this bound reduces to $O(n)$) and, therefore, obtain an $O(\frac{n^{2d}}{2^d})$ upper bound for $V(n, d)$. It means that $V(n, d)$ lies between $O(n^d)$ and $O(\frac{n^{2d}}{2^d})$ depending on the weight vector. In turn, we have already seen, that each facet contains up to $d!$ vertices, which leads to $N(n, d) \ll V(n, d)$.

WMTD type	d	n	Time per facet	Total time (seconds)
Zonoid	3	10	0.009700	0.445
Zonoid	3	15	0.013840	1.531
Zonoid	4	10	0.012474	1.609
Zonoid	4	15	0.015862	14.140
Zonoid	5	10	0.017370	2.398
Zonoid	5	15	0.022335	40.953
ECH	3	10	0.009111	0.843
ECH	3	15	0.012212	2.375
ECH	4	10	0.015255	21.891
ECH	4	15	0.019632	97.765
ECH	5	10	0.023519	117.625
ECH	5	15	0.029733	1032.75
ECH*	3	10	0.009610	0.750
ECH*	3	15	0.012218	1.617
ECH*	4	10	0.015286	22.922
ECH*	4	15	0.020011	94.070
ECH*	5	10	0.022970	139.070
ECH*	5	15	0.029660	890.68
Geometrical	3	10	0.009355	0.930
Geometrical	3	15	0.013056	1.101
Geometrical	4	10	0.015356	23.805
Geometrical	4	15	0.020157	93.406
Geometrical	5	10	0.023036	137.312
Geometrical	5	15	0.029794	1028.51

Table 1: Sample computational results.

Regarding the hash table of created facets, each facet occupies $O(d \cdot \log_2 n)$ storage size, while the hash table, in almost any case, has a constant size C, independent of n and d. Therefore, the use of general memory is of the order $O(N(n,d) \cdot d \cdot \log_2 n + C)$. Facets, once they have been created, are put into a secondary store, thus considerably lowering the storage cost.

Table 1 exhibits the results of a first small simulation study. It gives an idea how the time for computing a single facet depends on d and n and how it varies with several types of WMT regions: zonoid, ECH*, ECH, and geometrically trimmed regions (for the latter two, see Dyckerhoff and Mosler 2011). The data is distributed uniformly on a d-dimensional cube. We focus on the time per facet (TpF) because it characterizes the efficiency of the algorithm in a most obvious way. The total computational time amounts to the latter multiplied by the number of facets, which is a parameter depending only on the data. We observe that the TpF shows the following growth behavior: Approximately linear on n and slightly convex on d, which may be seen as some low order polynomial dependency on dimension.

5. The R package WMTregions

The algorithm has been programmed as an R (R Development Core Team 2012) package and named **WMTregions** (Bazovkin and Mosler 2012). It is available for download from the

Comprehensive R Archive Network at http://CRAN.R-project.org/package=WMTregions. Properly, the main functionality has been realized in C++ and the R part is used as (i) a thin client for the pre-compiled routine, (ii) the user interface and (iii) for the visualization. In the next subsection we consider it in detail. The formal description of the functions and architecture will be followed by two examples of applying the package to simulated and to real data.

5.1. Technical overview

Dependencies

An autonomously compiled C++ program provides a 3d visualization as it is shown, for instance, in Figure 8. The visualization is designed by means of a cross-platform graphical specification **OpenGL**. In turn, in the R package we access the **OpenGL** functionality by means of the **rgl** package (Adler and Murdoch 2012).

The less powerful but applicable for the data of any dimension, vertices based visualizing is realized with the help of the **rggobi** package (Temple Lang, Swayne, Wickham, and Lawrence 2011). The latter also uses graphical toolkit **GTK+** through its R proxy package **RGtk2** (Lawrence and Temple Lang 2010, 2012). To be able to use it you must first install **GTK+** library (**GTK+** Development Team 2012) on your machine. On the most systems this installation is proposed automatically while getting **RGtk2**. If not, you must do it manually before using the package. Moreover, the old versions of **GTK+** and **GGobi** (Swayne, Cook, Temple Lang, and Buja 2010) can cause problems in installing and using **RGtk2** and **rggobi**: If the packages fail, you must reinstall **GTK+** and **GGobi**.

R functions

The package contains functions for calculating and representing WMT regions:

- Function `WMTR(fname = "Cloud.dat", fdir = getwd(), bound = 0)`.

  Goal: Calculates the WMT region.

  Arguments:

  - `fname`: The name of the data input file (see Section 5.1.3) in the directory `fdir`.
  - `fdir`: A path to the directory where the input and output files will be located. The default value is the R working directory.
  - `bound`: An option of additional outputting the lower or the upper boundary of the WMT region (i.e., $\partial\left(D_\alpha(x_1,\ldots,x_n)\oplus\mathbb{R}^d_+\right)\cap D_\alpha(x_1,\ldots,x_n)$ or $\partial\left(D_\alpha(x_1,\ldots,x_n)\oplus\mathbb{R}^d_-\right)\cap D_\alpha(x_1,\ldots,x_n)$, respectively). `-1` corresponds to the lower one; `1` – to the upper one; `0`, the default value, makes no additional output.

  Output:

  - A file `"TRegion.dat"` in the directory `fdir`. The calculated WMT region with facets represented by their normals and intercepts.
  - A file `"TRegion_vertices.dat"` in the directory `fdir`. The calculated WMT region with facets represented by the coordinates of their vertices.

– Auxiliary files "TRegion_bound.dat" and "TRegion_vertices_bound.dat" with a bound of the calculated WMT region.

Description: This function is the main function, which reads input data from an input file fname and writes the result into an output file "TRegion.dat", both files being located in fdir. The format of the files is described below in Section 5.1.3.

- Function visualWMTR(fdir = getwd()).

  Goal: Visualizes the calculated WMT region for the data in $\mathbb{R}^3$.

  Arguments:

  – fdir: A path to the directory where the output files of WMTR() are located. The default value is the R working directory.

  Output: Void value. The visualization of the calculated WMT region and the data cloud points in a separate window under R environment.

  Description: This function realizes the $3d$-visualization of the data based on the computational results of the WMTR() function. The parameter fdir must be the same as was used in WMTR().

- Function showWMTR(fdir = getwd()).

  Goal: Exhibits the calculated WMT region of any dimension by making multiple projections of its vertices into $\mathbb{R}^3$.

  Arguments:

  – fdir: A path to the directory where the output files of WMTR() are located. The default value is the R working directory. Must be the same fdir as in WMTR().

  Output: Void value. The **rggobi** visualization of the calculated WMT region (represented only by its vertices) in separate windows under R environment.

  Description: The function visualizes a calculated WMT region as a convex polytope by representing its vertices in **rggobi** (Temple Lang *et al.* 2011) interactive graphics framework. The visualization is a series of projections into $\mathbb{R}^3$. The whole interaction toolset of the **rggobi** package, such as "2d tour" or the projection pursuit, can be used here. In comparison with visualWMTR(), showWMTR() visualizes only vertices but, however, does it for the data of any dimension.

- Function loadWMTR(fname = "TRegion.dat", fdir = getwd()).

  Goal: Loads the calculated WMT region of $d =$ dim into a matrix object.

  Arguments:

  – fname: The name of the file that contains the calculated WMT region (the normal-intercept representation output file of WMTR()) in the directory fdir. The default name is "TRegion.dat".

  – fdir: A path to the directory where the file fname is located. The default value is the R working directory.

  Output:

– A matrix object containing the normal-intercept coordinates of the WMT region facets as its rows.

Description: This function loads the calculated WMT region of $d =$ `dim` into a matrix object in order to work with it as with a variable in R, for example, in using the function `pointinTR()`.

- Function `pointinTR(dpoint, tregion)`.

  Goal: Checks whether a point is in a specified trimmed region.

  Arguments:

  – `dpoint`: A vector containing the coordinates of the point to be checked.

  – `tregion`: A matrix object containing the WMT region in the normal-intercept representation.

  Output:

  – Boolean value, whether `dpoint` is contained by `tregion`.

  Details: `tregion` is normally produced by the `loadWMTR()` function basing on a calculated WMT region.

- Function `generTRsample(fname, fdir, dim, num, alpha, trtype)`.

  Goal: Generates sample data cloud file in a format appropriate for applying `WMTR()`.

  Arguments:

  – `fname`: The name of the output file.

  – `fdir`: A path to the directory where the file `fname` should be located.

  – `dim`: The dimension d of the data cloud.

  – `num`: The number of points in the data cloud.

  – `alpha`: The depth parameter.

  – `trtype`: The notion of the WMT region to be calculated.

  Output:

  – A file `fname` in the directory `fdir` with a data cloud of the specified parameters.

  Description: This function is an auxiliary one. It generates a random uniformly distributed data cloud of any size `num` and any dimension `dim` with a format of an input file described in Section 5.1.3. With the default values of its arguments it looks as follows: `generTRsample(fname = "Cloud.dat", fdir = getwd(), dim = 3, num = 20, alpha = 0.05, trtype = "zonoid")`.

Input and output

In this subsection we describe the format of the input and output information, which is represented by input and output files.

1. The input file. A data cloud is read from a text file of the following format (the sequence is fixed):

   - Type of the trimmed region (zonoid, ECH, ECH*, geometrically trimmed; given weight vector)

     Format: A text value from the following set: `"zonoid"`, `"ECH"`, `"ECH*"`, `"geometrical"`, `"general"`. `"general"` is used for the case when the weights are given manually instead of being automatically generated basing on the WMT region type and the depth parameter.
   - Depth parameter

     Format: A floating point number from the interval $[0, \ldots, 1)$.
   - Dimension

     Format: An integer number $d \geq 2$.
   - Number of points of the data cloud

     Format: An integer number $n > d$.
   - *(If the type* `"general"` *is selected)* The weight vector

     Format: n non-decreasing floating point numbers matching the requirements for the weight vector.
   - Coordinates of each point

     Format: n groups of d floating point numbers, each group representing the coordinates of a point from the data cloud.

     The points must be in the general position.

   For example, to calculate trimmed regions of a data cloud made of 7 points we have to input the following, where the left column refers to a zonoid region with depth parameter 0.05, and the right column to general WMT region defined by the weight vector `(0.02 0.02 0.03 0.15 0.15 0.26 0.37)`:

```
zonoid                                  general
0.05                                    0.00
3                                       3
7                                       7
3.433465 3.67261 2.985222               0.02 0.02 0.03 0.15 0.15 0.26 0.37
0.6119484 7.996853 6.70113              3.433465 3.67261 2.985222
6.429673 9.318805 5.684797              0.6119484 7.996853 6.70113
4.094673 3.255387 0.7768149             6.429673 9.318805 5.684797
4.764675 7.401488 1.766797              4.094673 3.255387 0.7768149
3.571828 4.102897 2.325751              4.764675 7.401488 1.766797
1.063743 0.7078045 8.968969             3.571828 4.102897 2.325751
                                        1.063743 0.7078045 8.968969
```

   A sample input file with any given parameters and random by generated coordinates is provided by the function `generTRsample()`.

2. Output files.

   The whole calculated WMT region is represented twofold:

- `"TRegion.dat"`.

  An output file `"TRegion.dat"` consists of lines, each representing a facet of the trimmed region in the normal-intercept format, namely by $d+1$ numbers giving the equation of the hyperplane containing the facet. The first d of these numbers are coordinates of a normal to the facet, which is directed outward the WMT region. The last number defines the intercept. For example,

  ```
  0.54301 0.43048 0.72100 -13.488
  ```

  corresponds to the hyperplane $\{x \in \mathbb{R}^3 : (0.54301\ 0.43048\ 0.72100)\cdot x - 13.488 = 0\}$.

- `"TRegion_vertices.dat"`.

  An output file `"TRegion_vertices.dat"` consists of lines, each representing a facet of the trimmed region by the coordinates of its vertices. The coordinates of vertices are given in parentheses, while the vertices of a facet are again collected in parentheses. For example, a single facet ($d = 3$) is given by:

  ```
  ( (1.290;9.249;2.059;) (0.995;9.108;1.729;) (1.099;9.129;1.662;)
  (1.978;9.391;1.613;) (2.030;9.416;1.671;) (1.445;9.296;2.050;)  ).
  ```

For some applications it makes sense to consider only the lower or upper boundary of the WMT region. This information is contained in two auxiliary files:

- `"TRegion_bound.dat"`. The same as `"TRegion.dat"` but containing facets only from the lower or upper boundary of the WMT region.

- `"TRegion_vertices_bound.dat"`. The same as `"TRegion_vertices.dat"` but containing facets only from the lower or upper boundary of the WMT region.

5.2. Examples

As an illustration how the algorithm works we present a comparative example of four different types of weighted-mean trimmed regions for the same data and depth parameter ($\alpha = 0.221$). Their visualization was done by a separately compiled C++ program and is exhibited in Figure 8.

In this subsection we give two examples of how to get such results by means of the installed package **WMTregions**.

Illustration with simulated data

As a first example, we show how to use the package on a randomly generated sample input file. Suppose we want to calculate a zonoid region of `100` data points in $\mathbb{R}^3$ with depth `0.117`. First, we load the package:

```
R> library("WMTregions")

Loading required package: rggobi
Loading required package: RGtk2
Loading required package: rgl
```

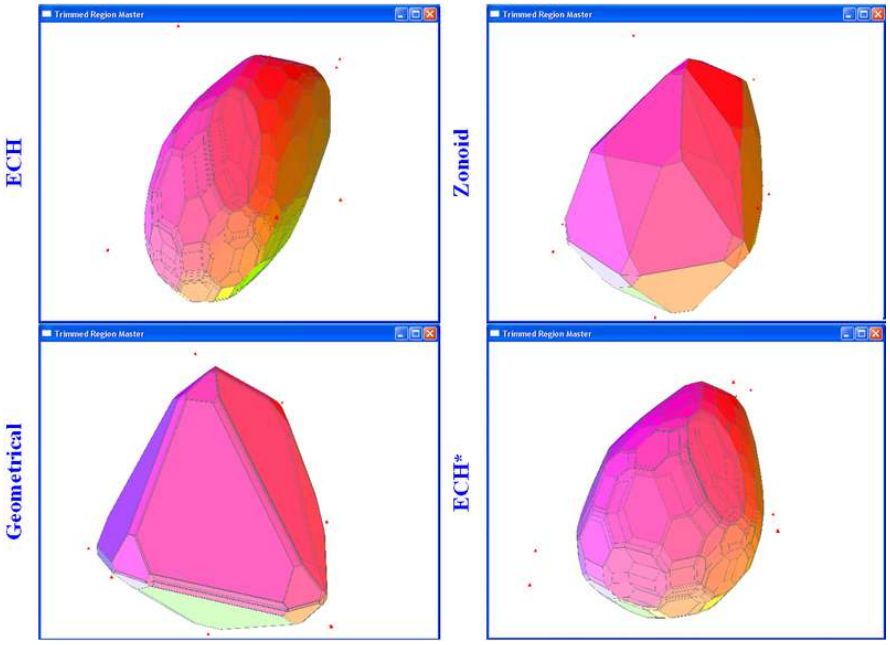

Figure 8: 3d visualization of various types of WMT regions.

Then we provide an input file with the data. The simplest way here is to use an embedded function `generTRsample()`. Having generated the file, we start the main procedure of calculating the WMT region:

```
R> generTRsample("Cloud.dat", dim = 3, num = 100, alpha = 0.117,
+    trtype = "zonoid")
R> WMTR("Cloud.dat")

[1] "The trimmed region was successfully calculated!"
```

Now we have two possibilities of visualizing the results: `showWMTR()` or `visualWMTR()`. As the data has dimension $d = 3$, the most appropriate choice is `visualWMTR()`:

```
R> visualWMTR()
```

You can see the 3d picture on the left side of Figure 9. On a color screen, the demonstrated facets are blue, while the ridges of the trimmed region are drawn in light green. Small red spheres represent the data cloud points. You can rotate or zoom the picture easily with the mouse.

Having obtained the result, we might want to check whether some point, say the origin $\mathbf{0}^\top$, lies inside the WMT region. The corresponding check is conducted as follows:

```
R> tregion <- loadWMTR("TRegion.dat")
R> point2check = c(0,0,0)
R> pointinTR(point2check, tregion)

[1] FALSE
```

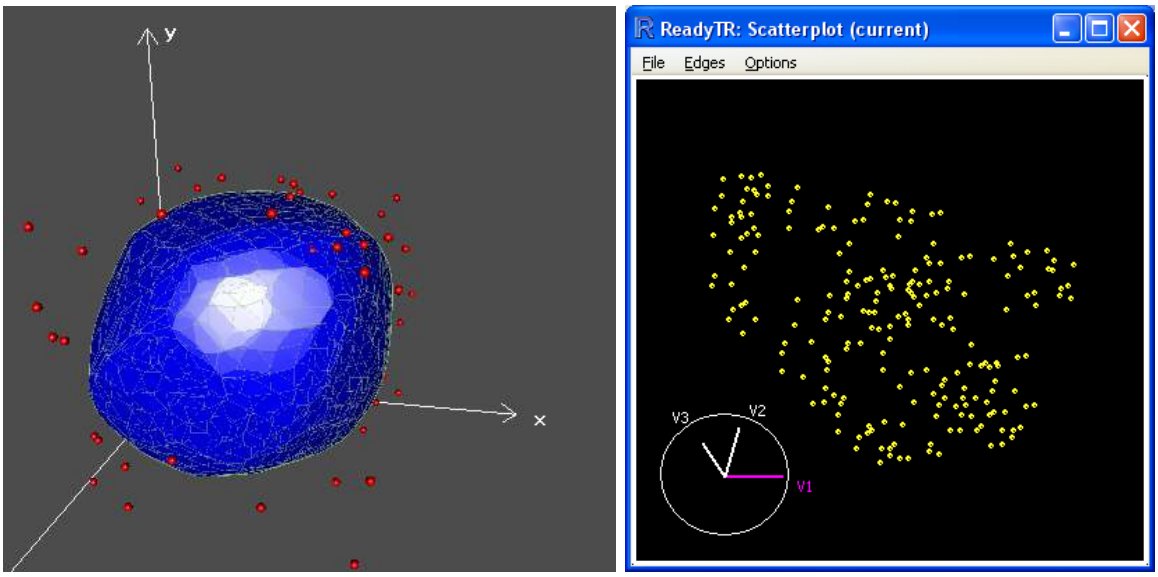

Figure 9: Visualization of the results in R.

Thus, the origin is outside the calculated WMT region. The right side of Figure 9 gives **rggobi** based visualization of the results for the data in dimension 4:

```
R> generTRsample("Cloud4.dat", dim = 4, num = 25)
R> WMTR("Cloud4.dat")

[1] "The trimmed region was successfully calculated!"

R> showWMTR()
```

Concerning available tools for manipulating the visualization in the window, we refer the reader to **rggobi** documentation.

Calculating multivariate set-valued risk measures

The second example represents an application of the WMT regions to the risk management. Our aim is to calculate the multivariate expected shortfall (Cascos and Molchanov 2007) set-valued risk measure. We have chosen this measure because it is the most important coherent risk measure. A zonoid region $ZD_\alpha(x_1, \ldots, x_n)$ determines the expected shortfall at the level α as $ES_\alpha(x_1, \ldots, x_n) = \mathbb{R}^d \setminus (ZD_\alpha(x_1, \ldots, x_n) \oplus \mathbb{R}^d_+)$. In other words, it is determined by the lower boundary of the zonoid region.

```
R> library("WMTregions")

Loading required package: rggobi
Loading required package: RGtk2
Loading required package: rgl
```

We have a file `"Indices_0809.dat"` (available in the supplements) with the real life data representing the relative losses in percent on DAX (x variable), Dow Jones (y variable) and

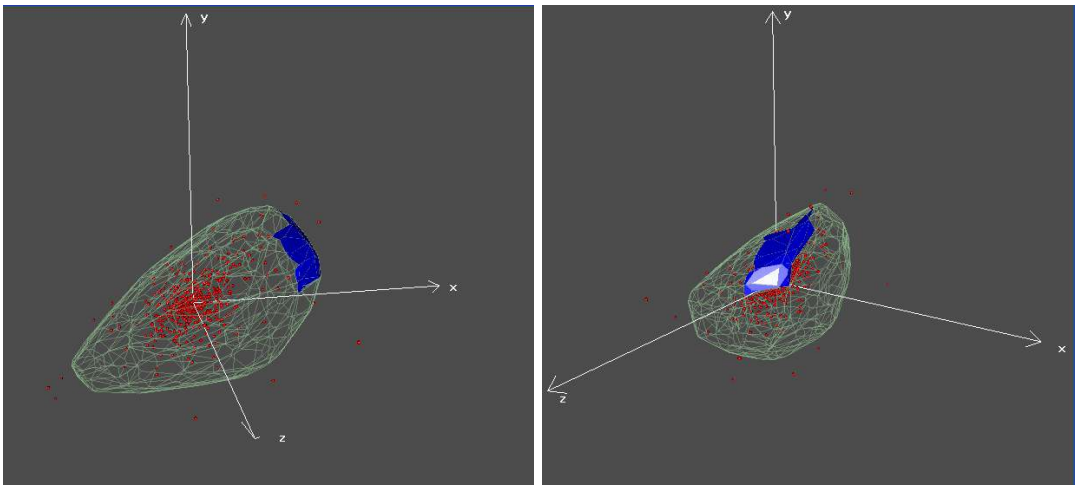

Figure 10: Visualization of the results in R.

Hang Seng (z variable) stock market indices in the years 2008 and 2009. Besides this, the data cloud points coordinates are also stored in the data set `"Indices_0809"` attached to the package. Using it as a historical information, we have to calculate the expected shortfall at the level 1% of the portfolio on these three indices. The data is represented by losses, therefore we are seeking for the reverse, that is, the upper boundary:

```
R> WMTR("Indices_0809.dat", bound = 1)

[1] "The trimmed region was successfully calculated!"

R> visualWMTR()
```

On Figure 10 we see the most critical part of the surface in blue. It is shown from two sides.

6. Conclusions

An exact algorithm has been constructed to compute the WMT regions of an empirical distribution in d-space for an arbitrarily given weight vector. It calculates all facets, edges, and vertices of a region at any given depth $\alpha \in]0, 1[$. (Recall that $\alpha = 0$ and $\alpha = 1$ are trivial cases.) In fact, the case $\alpha = 0$ can be also calculated, but then the WMT region degenerates into the convex hull of the data set.

The "double-hash" mechanism plays the prominent role by marking in a special way the ridges in the hash table, thus guaranteeing that each facet of the WMT region is generated only once and only these facets are calculated. It induces the unique order on the set of the facets, making the algorithm efficient. Really, the latter has as many loops as the WMT region has facets, which, obviously, is the minimum number of facet generating loops in this sort of algorithms. Moreover, in some significant applications of the WMT regions, such as the multivariate risk measurement, we can take advantage of the connectivity of the generated facets and calculate only, for instance, the lower boundary of the WMT region, which covers about $\frac{1}{2^d}$ of its surface.

To sum up, we would like to point out some perspectives of the future work on the algorithm and the R package. While the precise algorithm is efficient and has the optimal number of computational steps, its most important use is to employ it as a benchmark for computationally cheaper approximate procedures. As we have seen above, WMT regions have very large numbers of facets. Next steps of research target developing procedures of filtering them and replacing the "jumps" by "long jumps" across parallel ridges. Further, enhanced methods of more user-friendly visualizations in dimension $d \geq 4$ are under investigation.

Acknowledgments

We are thankful to Rainer Dyckerhoff for fruitful discussions and valuable advices. Also we would like to thank Marcus Kotsakechagias for preparing the data for the example concerning risk measurement and testing the R package on many financial datasets. And last but not least, we are grateful to two anonymous referees for their careful reading and their very valuable suggestions concerning the presentation of our results.

References

Adler D, Murdoch D (2012). ***rgl**: 3D Visualization Device System (**OpenGL**)*. R package version 0.92.880, URL http://CRAN.R-project.org/package=rgl.

Bazovkin P, Mosler K (2012). ***WMTregions**: Exact Calculation of Weighted-Mean Trimmed Regions*. R package version 3.2.5, URL http://CRAN.R-project.org/package=WMTregions.

Cascos I (2007). "The Expected Convex Hull Trimmed Regions of a Sample." *Computational Statistics*, **22**, 557–569.

Cascos I (2009). "Data Depth: Multivariate Statistics and Geometry." In WS Kendall, I Molchanov (eds.), *New Perspectives in Stochastic Geometry*. Clarendon Press, Oxford University Press, Oxford.

Cascos I, Molchanov I (2007). "Multivariate Risks and Depth-Trimmed Regions." *Finance and Stochastics*, **11**, 373–397.

Dyckerhoff R (2000). "Computing Zonoid Trimmed Regions of Bivariate Data Sets." In J Bethlehem, P van der Heijden (eds.), *COMPSTAT 2000 – Proceedings in Computational Statistics*, pp. 295–300. Physica-Verlag, Heidelberg.

Dyckerhoff R (2002). "Inference Based on Data Depth." In K Mosler (ed.), *Multivariate Dispersion, Central Regions and Depth: The Lift Zonoid Approach*, chapter 5. Springer-Verlag, New York.

Dyckerhoff R, Mosler K (2011). "Weighted-Mean Trimming of Multivariate Data." *Journal of Multivariate Analysis*, **102**, 405–421.

Edelsbrunner H (1987). *Algorithms in Combinatorial Geometry*. Springer-Verlag, Heidelberg.

GTK+ Development Team (2012). ***GTK+**: The Gimp Toolkit*. Version 3.4, URL http://www.gtk.org/.

Hallin M, Paindaveine D, Šiman M (2010). "Multivariate Quantiles and Multiple-Output Regression Quantiles: From L_1 Optimization to Halfspace Depth." *The Annals of Statistics*, **2**, 635–669.

Knuth D (1997). *The Art Of Computer Programming*, volume 1. 3rd edition. Addison-Wesley, Boston.

Koshevoy G, Mosler K (1997). "Zonoid Trimming for Multivariate Distributions." *The Annals of Statistics*, **25**(5), 1998–2017.

Lawrence M, Temple Lang D (2010). "**RGtk2**: A Graphical User Interface Toolkit for R." *Journal of Statistical Software*, **37**(8), 1–52. URL http://www.jstatsoft.org/v37/i08/.

Lawrence M, Temple Lang D (2012). ***RGtk2**: R Bindings for **GTK** 2.8.0 and Above*. R package version 2.20.24, URL http://CRAN.R-project.org/package=RGtk2.

Liu RY, Parelius JM, Singh K (1999). "Multivariate Analysis by Data Depth: Descriptive Statistics, Graphics and Inference." *The Annals of Statistics*, **27**(3), 783–858.

Mosler K (2002). *Multivariate Dispersion, Central Regions and Depth: The Lift Zonoid Approach*. Springer-Verlag, New York.

Mosler K, Hoberg R (2006). "Data Analysis and Classification with the Zonoid Depth." In RY Liu, R Serfling, D Souvaine (eds.), *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, pp. 49–59. American Mathematical Society.

Mosler K, Lange T, Bazovkin P (2009). "Computing Zonoid Trimmed Regions in Dimension $d > 2$." *Computational Statistics & Data Analysis*, **53**, 2500–2510.

R Development Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rockafellar RT (1970). *Convex Analysis*. John Wiley & Sons, New York.

Serfling R (2006). "Depth Functions in Nonparametric Multivariate Inference." In RY Liu, R Serfling, D Souvaine (eds.), *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, pp. 1–16. American Mathematical Society.

Swart G (1985). "Finding the Convex Hull Facet by Facet." *Journal of Algorithms*, **6**(1), 17–48.

Swayne D, Cook D, Temple Lang D, Buja A (2010). "**GGobi** Software, Version 2.1." URL http://www.ggobi.org/.

Temple Lang D, Swayne D, Wickham H, Lawrence M (2011). ***rggobi**: Interface Between R and **GGobi***. R package version 2.1.17, URL http://CRAN.R-project.org/package=rggobi.

Winder R (1966). "Partitions of N-Space by Hyperplanes." *SIAM Journal on Applied Mathematics*, **14**(4), 811–818.

A. Heuristics for determining all adjacent facets

Given a basis $\mathcal{V}_F$ of a facet F, let A be a nonsingular $d \times d$ matrix that contains the basis vectors as its first $d-1$ rows and an arbitrary last row that is linearly independent from the other rows. Consider the linear equation

$$Ar = e_d := \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{11}$$

The unique solution r to this equation is a scalar multiple of the normal vector p_F of F.

In search for a neighboring facet, the support vector has to be rotated in a plane of dimension two in $\mathbb{R}^d$ (Step C.(b.)II.v.). To reduce the algorithmic complexity of this step we compute the rotation plane in the following efficient way.

The transition from F to a neighboring facet, say via a ridge m, is done by a basis exchange. This means replacing some k rows of the matrix A (having indices $i \in \mathcal{I}$) by $k-1$ other data vectors and, as its last row, some vector that is linear independent from all previous rows and non-orthogonal to p, for example p itself. Let S_m denote the $d \times d$ matrix obtained from A by this exchange, and V_m the $d \times k$ matrix built from the k new vectors as columns. Thus, the solution $z = z_m$ of the linear equation

$$S_m z = e_d\,, \tag{12}$$

spans, together with p_F, a plane in which the support vector p may be rotated.

Note that (12) can be solved directly by calculating S_m^{-1}, which is the straightforward computation and has complexity $O(d^3)$. Instead, in our algorithm we decompose S_m in order to reduce the complexity of this step. It is easy to see that

$$S_m = K_m \cdot A\,,$$

where K_m is an identity matrix with substituted rows of indices $i \in \mathcal{I}$. Let these rows form a matrix C_m whose i-th row corresponds to the j_i-th row of K_m. Then it holds $A^\top C_m = V_m$ and, consequently,

$$C_m = \left(A^{-1}\right)^\top V_m\,.$$

Note that A^{-1} has to be computed only once at each facet. Given A^{-1}, calculating C_m has complexity $O(d^2)$.

Henceforth we denote the elements of A and C by a_{ij} and c_{ij} respectively. Consider rewriting (12) as

$$K_m A z = e_d\,,$$

where $K_m = \begin{pmatrix} 1 & 0 & & 0 \\ 0 & 1 & \ddots & 0 \\ \cdots & & & \\ c_{j_i 1} & c_{j_i 2} & c_{j_i i} & c_{j_i d} \\ \cdots & & \ddots & \\ 0 & 0 & & 1 \end{pmatrix}$ and $K_m^{-1} e_d = \begin{pmatrix} 0 \\ \vdots \\ -\frac{c_{j_i d}}{c_{j_i i}} \\ \vdots \\ 0 \\ 1 \end{pmatrix}$. Then it holds

$$z_m = A^{-1}K_m^{-1}e_d = A^{-1}\begin{pmatrix} 0 \\ \vdots \\ -\frac{c_{j_i d}}{c_{j_i i}} \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

For a single facet we have to compute (in $O(d^3)$)

$$A^{-1} = \begin{pmatrix} \alpha_1 & | & \dots & | & \alpha_i & | & \dots & | & \alpha_d \end{pmatrix}.$$

Thus, besides computing C_m, only the following computation is done to find a basis for the m-th "jump":

$$z_m = \alpha_d - \sum_{i \in \mathcal{I}} \frac{c_{j_i d}}{c_{j_i i}} \alpha_i, \quad O(d). \tag{13}$$

Recall that a basis for an m-th jump is given by $\{\bar{z}_m, \bar{p} = \bar{r}\}$. Let us denote the number of ridges for a facet F by $R(F)$. The complexity of finding bases for all jumps from the facet is

$$O(d^3 + (d^2 + d) \cdot R(F)) \cong O(d^2 \cdot R(F)).$$

It can be easily checked that, if we do not exploit the common information on A^{-1}, the complexity amounts to $O(d^3 \cdot R(F))$.

Affiliation:

Pavel Bazovkin
Graduate School of Risk Management
Universität zu Köln
Meister-Ekkehart-Str. 11
50923 Cologne, Germany
Telephone: +49/221/470-7705
E-mail: bazovkin@wiso.uni-koeln.de

Karl Mosler
Department of Economic and Social Statistics
Universität zu Köln
Albertus-Magnus-Platz
50923 Cologne, Germany
Telephone: +49/221/470-4266
E-mail: mosler@statistik.uni-koeln.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 47, Issue 13 *Submitted:* 2011-02-24
May 2012 *Accepted:* 2012-03-12