Computing zonoid trimmed regions in dimension $d > 2$

Karl Mosler, Köln*
Tatjana Lange, Merseburg
Pavel Bazovkin, Kiev

February 26, 2008

Abstract: A probability distribution on Euclidean $d$-space can be described by its zonoid regions. These regions form a nested family of convex sets around the expectation, each being closed and bounded. The zonoid regions of an empirical distribution introduce an ordering of the data that has many applications in multivariate statistical analysis, e.g. cluster analysis, tests for multivariate location and scale, and risk analysis. This paper provides an exact approach to construct the zonoid regions of an empirical distribution by their facets in dimension $d \geq 3$. It presents a characterization of vertices and their adjacency and a procedure by which the adjacent vertices of a zonoid region are found and its facets are built.

Keywords: Multivariate statistical analysis, central regions, zonoid data depth, convex polytope, algorithm.

1 Introduction

A probability distribution on Euclidean $d$-space can be described by its zonoid trimmed regions or, shortly called, zonoid regions. These regions

*Corresponding author: Karl Mosler, Department of Social and Economic Statistics, University of Cologne, Albertus-Magnus-Platz, 50923 Köln, Germany. mosler@statistik.uni-koeln.de.
form a nested family of convex sets – so called central regions – around the expectation, each being closed and bounded. The family is indexed by numbers that vary in the unit interval; it characterizes the underlying distribution in a unique way. Each zonoid region can be seen as a set valued parameter that reflects the location, scale, and shape of the distribution.

Zonoid regions have been introduced by Koshevoy and Mosler (1997) and have found many applications in multivariate statistical analysis. Regarding zonoid depth, their boundaries forms depth contours, which can be regarded as multivariate quantiles. Given a $d$-variate empirical distribution, zonoid regions are used as trimmed regions that exclude “outlying” data and include “inlying”, that is, central and relevant ones. Zonoid regions have been employed in cluster analysis (Hoberg, 2000), tests for multivariate location and scale (Dyckerhoff, 2002), and risk analysis (Molchanov and Cascos (2007); see also the monograph by Mosler(2002). Similar methodology has been based on alternative notions of data depth and trimmed regions, such as halfspace (= location) depth, simplicial depth, expected convex hull depth, and others. Zuo and Serfling (2000) provide some general theory on depth trimmed contour regions, while Liu, Parelius and Singh (1999) and Serfling (2006) broadly survey the theory and applications of various notions of depths.

In applying such methods to given multivariate data, the crucial point is the availability of efficient numerical procedures to compute the employed data depths and trimmed regions. To calculate the depth of a single point, algorithms have been provided by Rousseeuw and Ruts (1996) and Rousseeuw and Struyf (1997) for the halfspace depth, and by Dyckerhoff et al. (1996) for the zonoid depth. Aloupis (2006) gives a survey of algorithms for calculating different notions of medians and depths. But, calculating a depth trimmed region appears to be a much more demanding task. So far, algorithms have been constructed for the halfspace trimmed regions by Ruts and Rousseeuw (1996) and Miller et al. (2003) in dimension 2, and by Fukuda and Rosta (2004) in arbitrary dimension. For bivariate zonoid trimmed regions, Dyckerhoff (2000) provides an algorithm that employs a circular sequence; Cascos (2006) uses the same approach for bivariate regions that are trimmed by expected convex hull depth. In the sequel we will present an algorithm that efficiently calculates zonoid regions of any dimension.

Consider an empirical distribution that gives probability $\frac{1}{n}$ to each of the observations $x_1, \ldots, x_n \in \mathbb{R}^d$ and let $n \geq d$. The zonoid regions of the
empirical distribution are defined by

\[ D_\alpha(x_1, \ldots, x_n) = \left\{ \frac{1}{n\alpha} \sum_{i=1}^{n} \lambda_i x_i : \sum_{i=1}^{n} \lambda_i = n\alpha, 0 \leq \lambda \leq 1 \quad \forall i \right\}, \quad (1) \]

0 < \alpha < 1. It is immediately seen from the definition that \( D_{\frac{1}{n}}(x_1, \ldots, x_n) \) is the convex hull of the data \( x_1, \ldots, x_n \), while \( D_1(x_1, \ldots, x_n) \) is the set that contains the mean \( \frac{1}{n} \sum_{i=1}^{n} x_i \) as a single point. At 0 < \alpha < 1, \( D_\alpha(x_1, \ldots, x_n) \) is a convex polytope that lies between the convex hull and the expectation point and decreases strictly with \alpha. The border of such polytope consists of a finite number of facets. Each facet is part of a hyperplane in \( \mathbb{R}^d \) and can be described by the direction of its normal vector and its distance from the origin, that is, by some element \( p \) of the unit sphere \( S^{d-1} \) and some \( \lambda \in \mathbb{R}_+ \). The main task is to identify, among all directions \( p \in S^{d-1} \), those directions that determine the facets and compute them in an efficient way.

This paper provides an exact algorithm to construct the zonoid regions of an empirical distribution by their facets. That is, for any data \( x_1, \ldots, x_n \) and any \( \alpha \in [0,1] \), the facets and vertices of \( D_\alpha(x_1, \ldots, x_n) \) are calculated and their coordinates given. Our algorithm is efficient in that it computes the facets one after the other, proceeding from one facet to its neighbors.

In dimension \( d = 2 \), Dyckerhoff (2000) has developed an algorithm for constructing zonoid regions. His procedure is based on the idea of a circular sequence (cf. Edelsbrunner, 1987): A ray starting at the center is turned like a clock’s hand and the data points are projected on this ray. However, the method of circular sequence works only on bivariate data. There is no obvious generalization of such a sequence to higher dimensions.

Our first task is to characterize the vertices and facets of a given zonoid region, given the data points \( x_1, \ldots, x_n \) and \( \alpha \). For this, we introduce a global structure that partitions \( \mathbb{R}^d \) into direction cones that correspond one-to-one to the vertices of the zonoid region. In this cone structure, the adjacency of vertices is investigated and characterized. A linear program is constructed to decide, whether two vertices are neighbors. The resulting adjacency graph consists of elementary cycles that have either three or six nodes. Next, we show that each facet of the zonoid region corresponds to exactly \( d \) data points and provide a characterization of facets by a linear restriction on \( d \) data points.
As a second task, the facets of the zonoid region have to be put into an order by which they can efficiently calculated. For a given facet, a “jump-to-neighbor” procedure is introduced to transfer the calculation to the neighboring facets. Finally, a facet transversal graph is built, and a spanning tree order is realized to transverse this graph in an efficient way. This completes the algorithm.

Overview of the paper: Section 2 introduces the set of supporting vectors that belong to a given vertex of the zonoid. In Section 3 a global structure of direction cones is introduced, and the adjacency of vertices is described through conditions on these direction cones. Section 4 provides a linear program by which the adjacency of vertices can be checked. Section 5 presents the adjacency graph and a characterization of facets of the zonoid region. In Section 6 the “jump-to-neighbor” procedure and the spanning tree order are introduced, by which all facets are transversed. Section 7 concludes with a discussion of the complexity of our algorithm and its use in calculating zonoid regions for different $\alpha$. It also provides first numerical experience and remarks on possible modifications of the algorithm.

2 Vertices and direction domains of a zonoid region

Let us first recall some standard notions and facts about convex sets and polytopes in $\mathbb{R}^d$. A convex polytope is a finite intersection of closed halfspaces in $\mathbb{R}^d$. A nonempty intersection of its boundary with a hyperplane is called a facet if it has affine dimension $d - 1$, and a ridge if it has affine dimension $d - 2$. It is called an edge if it is a line segment, and a vertex if it is a single point. The boundary of a convex polytope is the union of its facets. A convex polytope has a finite number of facets, ridges, edges, and vertices. An edge is the intersection of (at least) two facets, and a vertex is the intersection of (at least) two edges, $d - 1$ ridges and $d$ facets.

A compact convex set is mentioned as a convex body. In particular, as a zonoid region is a bounded convex polytope, it forms a convex body in $\mathbb{R}^d$. The support function $h_C : S^{d-1} \to \mathbb{R}$ of a convex body $C \subset \mathbb{R}^d$ is defined by

\[ h_C (p) = \max\{ p'x : x \in C \}. \quad (2) \]
The support function of a convex body is closely related with its extreme points: A point $x_0$ is extreme in $C$ if and only if there exists some $p \in S^{d-1}$ such that

$$p'x = h_C(p) \implies x = x_0.$$ 

Now, for given data $x_1, \ldots, x_n$, denote

$$H = H(x_1, \ldots, x_n) = \{ p \in S^{d-1} : p'x_i = p'x_j \text{ for some } i \neq j \}.$$ 

Given a direction $p \in S^{d-1} \setminus H$, the inner product $p'x$ projects the data points $x_1, \ldots, x_n \in \mathbb{R}^d$ to numbers $p'x_1, \ldots, p'x_n \in \mathbb{R}$. While the data as points in $\mathbb{R}^d$ have no natural total order, their projection has. Thus, each $p \in S^{d-1} \setminus H$ induces a total ordering of the data, that is, a permutation $\pi_p$ of the index set $1, \ldots, n$ given by

$$p'x_{\pi_p(1)} < p'x_{\pi_p(2)} < \cdots < p'x_{\pi_p(n)}.$$ 

In the sequel we notate the $p$-ordered data by

$$X^p = (x_1^p, \ldots, x_n^p) \quad \text{with} \quad x_i^p = x_{\pi_p(i)}, \quad i = 1, \ldots, n.$$ 

There holds:

**Proposition 1. (Dyckerhoff, 2000)** Let $x_1, \ldots, x_n \in \mathbb{R}^d$ be pairwise distinct, $d \in \mathbb{N}$. For any $p \in S^{d-1} \setminus H$ define

$$x_{p,\alpha} = \frac{1}{n\alpha} \sum_{i=1}^{n} \lambda_i^p x_i,$$  

where

$$\lambda_i^p = \begin{cases} 
1 & \text{if } \pi_p(i) > n - \lfloor n\alpha \rfloor, \\
\frac{1}{n\alpha - \lfloor n\alpha \rfloor} & \text{if } \pi_p(i) = n - \lfloor n\alpha \rfloor, \\
0 & \text{if } \pi_p(i) < n - \lfloor n\alpha \rfloor.
\end{cases}$$

Then the set of vertices of the zonoid region $D_\alpha$ is given by

$$\mathcal{V}(D_\alpha) = \{ x_{p,\alpha} \in \mathbb{R}^d : p \in S^{d-1} \setminus H \}.$$ 

The set of all directions that yield vertices (= extreme points) of $D_\alpha$ is $S^{d-1} \setminus H$. Let $S(v) \subset S^{d-1} \setminus H$ denote the subset of those directions that belong to a given vertex $v \in S^{d-1} \setminus H$. $S(v)$ is named a direction domain.
According to Proposition 1, all directions that provide the same permutation of the data belong to the same direction domain, that is, to some common vertex $v$. The family of direction domains $S(v), v \in V(D_\alpha)$, forms a finite partition of $S^{d-1} \setminus H$.

Thus, the proposition gives us a discretization of the continuum of possible directions of the vector $p$, where the cardinality of the set of direction domains equals the number of vertices of the zonoid region. That is, a one-to-one relation between domains of directions and vertices has been established.

In the sequel we assume that the data are in general position, that is, every subset of $k + 1$ data points generates an affine space of dimension $k$, $k = 1, \ldots, d - 1$. (If the data are not in general position, the subsequent discussion and the algorithm need to be modified, e.g., by slightly perturbing the data.) Also, without loss of generality, we assume that the mean of the data is at the origin, $\frac{1}{n} \sum_{i=1}^{n} x_i = 0$.

### 3 Adjacent vertices

In this section we investigate the transition of one vertex to another, that is, of one direction domain to another. In our procedure we let a support vector $p$ – that represents direction – continuously move on the unit sphere $S^{d-1}$. We start with an arbitrary $p \in S^{d-1}$, which provides an initial permutation of the data points. Then we move $p$ continuously on great circles of the sphere. As it was mentioned, all $p$ that do not change the permutation of the points form a direction domain that belongs to a common vertex. When searching for all vertices, it obviously suffices to traverse each direction domain once. For this, we want to characterize and identify the possible transitions of one domain to a neighboring one. Our identification procedure is based on the following three lemmas.

**Lemma 1.** The vector $p$ hits the boundary of a direction domain only if, for some $i \neq j$, $p^t x_i = p^t x_j$, that is, $p$ is orthogonal to $x_i - x_j$.

**Proof.** Let us continuously turn the support vector $p$. Then values of the points’ projections (inner products) change also continuously. Thus the relative position of the points in the permutation does not change until two
such projections $p'x_i$ and $p'x_j$ become equal. If we continue moving the support vector further, the projections will interchange places. Coincidence of the projections corresponds to orthogonality of $p$ and the difference vector $p'x_i - p'x_j$. □

Note that the pair $(p'x_i, p'x_j)$ is not unique. However, at most $d - 1$ such pairs can arise since the space of all vectors that are orthogonal to $p$ has affine dimension $d - 1$ and the data are in general position.

**Lemma 2.** The vector $p$ crosses the boundary of a direction domain only if, for some $i \neq j$, $p'x_i$ and $p'x_j$ change their order. That is, any transition from one permutation to another is done by swapping some pair of data points.

**Proof.** Obvious.

The above considerations are summarized in the following theorem, which is a consequence of Proposition 1, Lemma 1, and Lemma 2.

**Theorem 1. (Identification of vertices)** The vector $p$ passes from one direction domain (and one vertex) to a neighboring one if and only if there are $i$ and $j$, $i \neq j$, such that $p$ is orthogonal to $x_i - x_j$, $\pi_p(i) = n - \lfloor n \cdot \alpha \rfloor$, and $|\pi_p(i) - \pi_p(j)| = 1$.

**Proof.** According to Lemma 2, the direction domain changes only if the order of some two points $x_i$ and $x_j$ is reverted that are neighbors under projections with $p$. Moreover, by Proposition 1, the vertex changes if and only if either $\pi_p(i) = n - \lfloor n \alpha \rfloor$ or $\pi_p(j) = n - \lfloor n \alpha \rfloor$ holds. □

Theorem 1 provides the basis for an algorithm that calculates all vertices of the zonoid region.

So far we have considered parts of the unit sphere, the direction domains, that correspond to the vertices of the zonoid region. For our subsequent discussion we will introduce the corresponding closed cones. For $v \in \mathcal{V}(D_\alpha)$ define the direction cone $C(v)$,

$$C(v) = \text{cl}\{\lambda y : \lambda \in \mathbb{R}_+, y \in S(v)\},$$

where $\text{cl}(U)$ means closure of a set $U \subseteq \mathbb{R}^d$. Each $C(v)$ is a closed convex cone in $\mathbb{R}^d$ with apex at the origin. It is finitely generated, having a maximum of $n - 1$ facets. The normals of its facets are described in Theorem 1.
The family of direction cones provides a global structure that divides the space $\mathbb{R}^d$ into sets corresponding to the vertices of $D_\alpha$. Next, we will clarify this structure and then, in the subsequent section, demonstrate how all vertices adjacent to a given vertex can be calculated.

**Lemma 3.** Consider three data points $x_i, x_j, x_k$ and the hyperplanes through the origin that are orthogonal to $x_i - x_j, x_i - x_k,$ and $x_j - x_k$, respectively. These hyperplanes intersect at a common hyperline that possibly contains a ridge of a direction cone. On the other hand, every ridge of a direction cone is contained in such a hyperline for some $i, j,$ and $k$.

**Proof.** Obvious.

**Corollary 1.** In the global structure a ridge of a direction cone belongs to another direction cone only if it is a ridge of the latter, too.

**Proof.** According to Lemma 3 each such ridge will be an intersection of (at least) three hyperplanes. For each direction cone that touches the ridge there are two bounding hyperplanes taking part in the intersection. This intersection of two hyperplanes carries a cone’s ridge. □

In the sequel we say that two direction cones are adjacent cones if they have a full facet in common. We conclude from Corollary 1 and Lemma 3 that a turning of the vector $p$ through a boundary facet implies a transition from a direction cone to an adjacent one. It means that, to leave a current cone and enter an adjacent one, we have to move the vector $p$ in an arbitrary way beyond one of the hyperplanes that carry the facets of the current cone as it is described in Lemma 3.

### 4 A linear program for constructing adjacent vertices

Our next task is to find an explicit way of constructing all vertices that are neighbors of a given vertex $v$. In other words, we will explicitly determine all direction cones that are adjacent to a given direction cone $C(v)$. These neighbors correspond to the facets of $C(v)$. Every facet of the cone is defined by a hyperplane. The set of hyperplanes that determine the cone’s boundary
is a subset of the hyperplanes described in Section 3. Each of these \( n - 1 \) hyperplanes is represented by its “inner” normal \( z_j \), i.e., the normal pointing inside the cone:

\[
    z_j = \begin{cases} 
        x^p_{n-[na]} - x^p_j & \text{if } j = 1, \ldots, n - [na] - 1, \\
        x^p_{n-[na]} - x^p_{j-[na]} & \text{if } j = n - [na] + 1, \ldots, n.
    \end{cases}
\]

The \( n - 1 \) normals are codirected with all directions \( p \) in the cone, that is, they have nonnegative inner products. In fact, \( C(v) \) is the intersection of all corresponding halfspaces. In other words, \( C(v) \) is the set of all vectors \( p \) that are codirected with the normals (4),

\[
    C(v) = \{ p \in \mathbb{R}^d : z'_j p \geq 0 \text{ for all } j = 1, \ldots, n - 1 \}.
\]

Our task is to identify those hyperplanes (or, equivalently, their normals) that belong to the boundary of the cone \( C(v) \). To determine, whether \( z_j \) is a boundary normal, we shall solve the following minimization problem,

\[
    \begin{align*}
    & \begin{array}{l}
        \text{min} \\
        \text{s.t.} \\
        \text{subject to}
    \end{array} \\
    & z'_j p \rightarrow \min \\
    & p \in C(v), \\
    & \sum_{i=1}^{d} |p_i| = 1.
    \end{align*}
\]

The solution of (4) tells us whether \( z_j \) is a boundary normal or not. If (4) has positive minimal value, there exists no support vector \( p \in C(v) \) that is orthogonal to \( z_j \). Hence the \( z_j \) is no boundary normal of \( C(v) \). If (4) is minimized with value \( z'_j p^* = 0 \), we conclude that \( p^* \) is a support vector that belongs to the boundary of \( C(v) \) and is element of the hyperplane that has normal \( z_j \). Consequently, \( p^* \) belongs also to the boundary of the direction cone \( C(\tilde{v}) \) of some vertex \( \tilde{v} \) that neighbors the current vertex \( v \).

(4) can be rewritten as a linear program (LPj),

\[
    \begin{align*}
    & \begin{array}{l}
        \text{min} \\
        \text{s.t.} \\
        \text{subject to}
    \end{array} \\
    & z'_j (p^+ - p^-) \rightarrow \min \\
    & z'_j (p^+ - p^-) \geq 0 \quad j = 1, \ldots, n - 1, \\
    & \sum_{i=1}^{d} (p^+_i + p^-_i) = 1, \\
    & p^+_i \geq 0, \quad p^-_i \geq 0, \quad i = 1, \ldots, d.
    \end{align*}
\]
Here we have inserted \( p = p^+ - p^- \), where \( p^+ = (p_1^+, \ldots, p_d^+) \) and \( p^- = (p_1^-, \ldots, p_d^-) \) are the positive and negative parts of \( p \), respectively. The linear program (5) is solved by the simplex method.

To find all vertices neighboring \( v \), we solve the linear programs (5) for \( j = 1, \ldots, n - 1 \). As all programs have the same set of feasible solutions, the calculations can be shortened by solving them simultaneously. In fact, the number of neighbors is small compared to \( n - 1 \). Therefore the number of basic feasible solutions in the simplex method will be relatively small too, which leads to a high average efficiency of the simplex method. In case \( n \) is large, the dual simplex method may outperform the primal approach.

Each ridge of a direction cone is an intersection of three hyperplanes. At the ridge either three or six direction cones touch each other. Let us consider these two cases in more detail:

Recall that, given a vertex \( v \), for all \( p \in C(v) \) the point \( x_{n-[na]}^p \) does not depend on \( p \). We call this point the main point of \( C(v) \).

Consider three direction cones \( C(v), C(w), \) and \( C(u) \) that have a common ridge and notate their main points by \( a, b, c \), respectively. For all \( p \in C(v) \) these three main points are \( p \)-ordered in the same way, either with \( bma \) in the middle or not. If \( a \) is in the middle, \( b \) and \( c \) cannot switch their positions. Hence, according to Lemma 2, \( C(w) \) and \( C(u) \) have no boundary hyperplane in common, and \( w \) and \( u \) are no adjacent vertices. In this case a total of six direction cones meet at the common ridge. If, on the other hand, \( b \) and \( c \) are on the same \( p \)-side of \( a \), their positions can switch and \( w \) and \( u \) are neighboring vertices. In this case, only three direction cones unite at the common ridge. The two cases are illustrated in Figure 1.

5 Edges and facets

Having an efficient procedure of finding extreme points we have to build an efficient one for constructing all facets. Our next lemma characterizes the edges of the zonoid region.

**Lemma 4 (Vertices and edges).** Let \( C(v) \) and \( C(w) \) be direction cones. The line connecting \( v \) and \( w \) is an edge if and only if \( C(v) \) and \( C(w) \) are adjacent cones.
**Proof.** Recall that the zonoid region \( D_\alpha \) is a convex polytope, and its extreme points form the vertices of this polytope. As \( C(v) \) and \( C(w) \) are direction cones of vertices \( v \) and \( w \), for all \( x \in D_\alpha \) it holds

\[
p'x \leq p'v \quad \text{if} \quad p \in C(v), \quad \text{and} \quad p'x \leq p'w \quad \text{if} \quad p \in C(w). \tag{6}
\]

Now assume that \( C(v) \) and \( C(w) \) are adjacent cones. Then for each \( p \) in their common boundary \( C(v) \cap C(w) \) it holds \( p'v = p'w \), hence, for all \( x \in D_\alpha \)

\[
p'x \leq p'(\lambda v + (1 - \lambda)w) \quad \text{if} \quad \lambda \in [0, 1]. \tag{7}
\]

It follows that the line connecting \( v \) and \( w \) is an edge of the polytope. On the other hand, assume that this line \( \overline{vw} \) is an edge. Then there exists some \( p \) that, for all \( x \in D_\alpha \), satisfies (7). For this \( p \), necessarily \( p'v = p'w \) holds. In view of (6), we conclude that (7) is true for all \( x \in D_\alpha \) if and only if

\[
p'(v - w) = 0, \quad p'v \geq 0, \quad \text{and} \quad p'w \geq 0,
\]

that is, \( p \) is in a \( d - 1 \)-dimensional cone which is subset of \( C(v) \) and of \( C(w) \). Consequently, \( C(v) \) and \( C(w) \) have a full facet in common, and thus are adjacent cones. \( \square \)
Lemma 4 provides a unique correspondence between the adjacency of direction cones in the global cone structure and the existence of edges of the zonoid region.

Recall that adjacent direction cones are such cones that have a common facet. The adjacency information of the zonoid region, which is a polytope, is represented by its adjacency graph, which consists of the polytope’s vertices and edges. Above we have demonstrated that either three or six direction cones touch each other at a ridge. Hence the adjacency graph is a concatenation of elementary cycles, each connecting either three or six vertices. This is illustrated in Figure 2:

![Figure 2: The structure of the adjacency graph](image)

Now, consider a facet of the zonoid region. Let the vector $\mathbf{p}$ be directed orthogonally to the facet. Then, if points from the data cloud are in general position, there exist exactly $d$ points $\mathbf{x}_{\pi_p(k)}, \mathbf{x}_{\pi_p(k+1)}, \ldots, \mathbf{x}_{\pi_p(k+d-1)}$ that

$$
\begin{align*}
\mathbf{p}' \mathbf{x}_{\pi_p(k)} &= \mathbf{p}' \mathbf{x}_{\pi_p(k+1)} = \cdots = \mathbf{p}' \mathbf{x}_{\pi_p(k+d-1)} \\
\text{with some } k, & \quad k \leq n - \lfloor n\alpha \rfloor \leq k + d - 1.
\end{align*}
$$

(8)

Obviously, the indices $k, \ldots, k + d - 1$ in (8) are not unique. However, any permutation of these $d$ points yields the same facet. We conclude the following theorem.

**Theorem 2. (Identification of facets)** Each facet can be identified by a set of exactly $d$ points from the data cloud and one of its vertices. Moreover, if there is a support vector that defines a permutation satisfying (8), then this permutation and these $d$ data points define a facet of the zonoid region.

It is easily seen from Theorem 2 that, if $k < n - \lfloor n\alpha \rfloor < k + d - 1$, the facet can have more than $d$ vertices. A facet will be mentioned as redundant if it
has more than \( d \) vertices, and as non-redundant if it has exactly \( d \) vertices. Anycase, there are only \( d \) different main points that belong to a facet. Let
\[
\ell = n - \lfloor n\alpha \rfloor - k.
\]
We obtain:

**Corollary 2.** The number of vertices of the facet equals \( d \cdot \binom{d-1}{\ell} \).

**Proof.** As stated above, the total number of possible relative positions of \( d \) points is \( d! \). But according to (3) the relative position of points in
\[
[k, (n - \lfloor n\alpha \rfloor)[ \text{ and } ](n - \lfloor n\alpha \rfloor), k + d - 1]
\]
is not significant, that is, there remain \( \binom{d-1}{\ell} \) different cases. This number is multiplied by \( d \), which is the number of possible main points. \( \square \)

**Corollary 3.** Each set of exactly \( d \) points from the data cloud defines at most one facet of a zonoid region unless:
\[
k \leq n - \lfloor n\alpha \rfloor \leq k + d - 1 \quad \text{and} \quad k \leq n - \lfloor n - n\alpha \rfloor \leq k + d - 1. \quad (9)
\]
Otherwise it defines exactly two parallel facets.

**Proof.** The first statement is clear from Theorem 2. The second is based on that if the condition (9) is met then the inverted support vector also defines a permutation satisfying (8). \( \square \)

**Corollary 4.** For each set of \( d \) data points there exists some \( \alpha \), such that the set defines a facet of the zonoid \( \alpha \)-region.

**Proof.** In fact, for a set of exactly \( d \) data points it is always possible to find an \( \alpha \) such that condition (8) is met. Then the statement follows from Theorem 2. \( \square \)

In case of a non-redundant facet we have \( \ell = 0 \). Then, according to Corollary 2 the facet is a \((d-1)\)-dimensional simplex having \( d \) vertices. The vertices identify the facet; they are pairwise adjacent and correspond to the pairwise adjacent direction cones. In turn every set of \( d \) vertices that correspond to pairwise neighboring cones defines either a facet or a cut of the zonoid region. Thus, also in this case, the identification of the facet is based on the adjacency graph.

Based on Corollary 3 we generate an arbitrary facet as follows:
1. Choose an arbitrary set of \( d \) points.

2. Check whether this set defines a facet. If not, go back.

3. Create the corresponding facet.

We will use this procedure also to initialize our algorithm by creating a first facet.

Thus Theorem 2 and its Corollaries provide a procedure for identifying each facet of the zonoid region.

6 Sequencing the facets

Now, to complete the algorithm, we have to create an efficient procedure that generates all facets in a sequential way.

The key point here is to specify an ordering relationship on the set of facets. This order has to be total, that is, all facets have to be generated by the algorithm. In case \( d = 2 \) such an order is easily created by a circular sequence; see Dyckerhoff (2000). But in higher dimensions there is no obvious solution.

We cope with this problem by introducing a spanning tree order (STO). For this, consider a ridge of the zonoid region. The ridge is part of a hyperline where two hyperplanes intersect. The hyperplanes define two facets that are incident with this ridge. Thus each ridge corresponds to exactly one neighboring facet and all neighboring facets can be found through ridges.

**Theorem 3. (Neighboring facets)**

(i) A facet has either \( 2d \) or \( d \) neighboring facets.

(ii) It has \( d \) neighbors if and only if it is non-redundant.

**Proof.** Let the support vector \( \mathbf{p} \) be orthogonal to the facet such that and condition (8) is met. A minimal violation of orthogonality is achieved by an infinitesimal move of \( \mathbf{p} \) to a direction that is perpendicular to a ridge of the facet and non-perpendicular to its other ridges.
This corresponds to the following change in the first equation of (8): Either 
\[ p'x_{\pi p(k)} < p'x_{\pi p(k+1)} = \cdots = p'x_{\pi p(k+d-2)} < p'x_{\pi p(k+d-1)} \]

or 
\[ p'x_{\pi p(k)} = \cdots = p'x_{\pi p(k+d-2)} < p'x_{\pi p(k+d-1)} \].

That is, a ridge is obtained by removing one of the points that define the facet according to the Theorem 2. It is clear from this that identification of a ridge is similar to that of a facet but based on a set of \( d - 1 \) points.

Hence, by removing one point to the higher and one to the lower part of the permutation, we obtain two ridges. Note that in a non-redundant case we can generate only one ridge in order not to violate the second equation of (8). The number of ridges is \( 2d \) (\( d \) in a non-redundant case), as there are \( d \) points to be removed.

Obviously, the number of facets that neighbor the current one equals the number of its ridges. \( \square \)

Thus, by removing one of the points that make a current facet according to the Theorem 2, we obtain a full characterization of either one or two new facets. In fact, for a redundant facet, removing a point yields two parallel ridges of the facet having different number \( \ell \) (see above). Each of these ridges is incident to the facet and to its neighbor. Thus we get two neighbors for a redundant, and one for a non-redundant facet.

Next we search a new facet incident to a given ridge. For this, we rotate the support vector \( p \) in the plane orthogonal to the ridge defined by the \( d - 1 \) points. Obviously, the rotation has to be made in the direction that corresponds to increasing \( p'x_i \) if \( x_i \) has been removed from the higher part of the permutation, and decreasing \( p'x_i \) in the opposite case. The rotation stops when the (8) first equation is met, that means, \( p \) has reached a normal of a new facet.

This procedure produces a “jump” from the current facet to each of its neighbors. We will mention it as the “jump-to-neighbor” procedure. By the procedure, sequentially generated facets are identified in very similar way, which allows for an efficient implementation of the procedure. Moreover, the traversal through all neighbors guarantees the absence of the “gaps”: no facet will be lost.

Based on Theorem 3 and the “jump-to-neighbor” procedure we shall build a special graph, the facet traversal graph (FTG). The vertices of the FTG correspond to the set of all facets of the zonoid region, and the edges of the
Figure 3: Examples of the facet traversal graph

FTG indicate the neighborhood of facets. As mentioned earlier, each vertex in this graph is incident with either \( d \) or \( 2d \) edges. In our procedure, the transition through an edge is performed by a “jump-to-neighbor”.

A proper sequential procedure for determining the zonoid region is obtained by finding a way to transverse all vertices of the corresponding FTG. This will be done for any dimension \( d \geq 0 \). Note, that in dimension \( d = 2 \) the FTG is an elementary cycle and its traversal is trivial and unique. In fact \( d = 2 \) is a degenerated case.

One way to solve the problem is building an Euler chain, that is a chain traversing each vertex of the FTG once. It can be visualized by “unfolding a cocoon”. An Euler chain is constructed by standard algorithms.

However, sequencing the vertices of the FTG by an Euler chain can be rather non-efficient. We construct an efficient algorithm that is based on the concept of spanning tree ordering. Note, that the Euler chain is a particular case of a spanning tree. In our approach, we create a spanning tree order (STO) of the FTG in a dynamical way. This STO orders the set of facets. For elementary steps the “jump-to-neighbor” procedure is used. It works as follows:

1. We organize a queue.
2. On each step pop from the queue a current facet, which corresponds to a vertex of the FTG. Add to the queue all adjacent vertices of the
current vertex that have not been processed so far. Mark the current vertex as processed.

3. Marking of the vertices is done through a hash table, where hash codes of all the processed vertices (i.e. facets of the zonoid region) are stored.

In the sequel, a hash table plays a significant role, as it enables us to store generated vertices in a special structure and realize a very fast search of whether the vertex has already been processed. According to the Corollary 3 the record for each facet in the hash table is fully described by $d$ integer numbers. These numbers are the labels of main elements of $d$ points that define the facet. If these points define 2 parallel facets, these facets can be easily generated on one step thus making possible to have one record for them in the hash table.

The linear order of the vertices (facets) is provided by its final positioning in the queue. Thus, the main problem of sequencing the calculation of facets is solved for any dimension $d \geq 3$.

The realization of the STO is illustrated by Figure 4:

![Figure 4: Realization of the STO](image)

Theorem 4. (Validity of the STO) The STO provides a sequential procedure by which all facets of the zonoid region are generated without repetition.
Proof. If two vertices are adjacent and one of them has been generated by
the procedure, the other will be generated too. As the FTG is connected, the
procedure yields each of its vertices. By definition, no vertices are repeated.

From Theorem 1 and Lemma 5 conclude that there are only $n$ possible dif-
ferent global cone structures. It follows from this fact that the global cone
structure is fully determined by the initial position $p_0$ of the support vector
and the corresponding main point, which can be any of the $n$ data points.

Now, given $p_0$, consider all $\alpha$ for which the main element has the same index
$k$, that is $n - [n\alpha] = \pi_{p_0}(k)$, equivalently

$$\frac{n - \pi_{p_0}(k)}{n} \leq \alpha < \frac{n - \pi_{p_0}(k) + 1}{n}.$$  

For all these $\alpha$, the global cone structure is the same. If we have calculated
the $\alpha$-region for one of these $\alpha$, the $\alpha$-regions for the other $\alpha$ are easily
determined by using the same global cone structure and only recalculating
the distances of facets from the data center.

7 Discussion

An algorithm has been constructed to calculate all facets of a zonoid region in
d-space. The algorithm is exact and works, in principle, for any number $n$ of
data points and any dimension $d$. The approach requires that the dimension
d$-2$ of ridges is not lower than the dimension $1$ of edges, that is, $d \geq 3$. This,
again, demonstrates that the bivariate case (Dyckerhoff, 2000) is special,
and the higher dimensional one asks for different methods, which have been
developed in this paper.

Each facet is generated only once. Thus the algorithm has as many loops as
the zonoid region has facets. Obviously, this is the minimum number of facet
generating loops in this sort of algorithm.

In a single facet generating loop, the most costly operations are the following:
Calculate the hyperplane equation of the current facet, calculate its distance
from the origin, and construct a basis of dimension $2$, following to obtaining
neighboring facets. This is done by solving linear equations and finding inner
products only. The complexity of the first operation is $O(d^3)$. Up to $d$ such operations are performed in each loop. The complexity of getting $n-1$ inner products is assessed $O(nd)$. That is, the complexity of one facet generating loop is described by $O(d^2(d^2 + n))$.

The number of computational loops of the algorithm is equal to the number of facets of the zonoid region. If the average number of facets is denoted $N(n, d)$, the average computational complexity of the algorithm amounts to $O(d^2(d^2 + n) \cdot N(n, d))$.

Notice that with increasing dimension $d$, complexity increases only moderately. For example, let us have two data clouds of dimensions $d_1$ and $d_2$ ($d_1 < d_2$) that contain the same number of points $n$. Then the second data cloud will form a polytope that has a more trivial structure in $\mathbb{R}^{d_2}$ than the first has in $\mathbb{R}^{d_1}$. Also it is easy to see, that there is no operation in the algorithm whose complexity grows exponentially with the growth of the dimension. Altogether the complexity is polynomial in $n$ and $d$. This confirms the efficiency of our algorithm.

Moreover, the structure of the algorithm lends itself well to being parallelized on a high-performance cluster system.

General memory resources are used, in the first place, for storing a hash table. The created facets then can be written into a secondary storage. A hash table consists of records, each having a size of at least $[d \cdot \log_2 n]$ bits. Therefore the use of general memory is has order $O(N(n, d) \cdot d \cdot \log_2 n)$.

We have implemented the algorithm on a PC. Figure 5 illustrates its application by exhibiting zonoid regions for a small data set of five points in dimension three and for several values of $\alpha$. Each zonoid region is depicted in three directions (by revolving it on a vertical axis). The data points are shown as little pyramids. (Note that these 3d-pictures employ a perspective view.)

From our implementation of the algorithm we have also obtained first quantitative results regarding the number of facets and total computation time in seconds. Table 1 exhibits, for different choices of $d$ and $n$, average values of total time, number of facets, and time per facet. The results in the table give an impression how the time for computing one facet grows with $d$ and $n$, and suggest that this growth is polynomial as well.
Figure 5: Graphic representation of the example in 3-d
Much computational load can be spared when we simultaneously calculate zonoid regions for several $\alpha$ that are sufficiently close to each other. If $[n \cdot \alpha_1] = \ldots = [n \cdot \alpha_k]$ holds, complete facets have to be computed for $\alpha_1$ only, while for $\alpha_2, \ldots, \alpha_k$ all facets are parallel to them; so, only their distances from the origin have to be calculated. Moreover, each facet that has been generated by the algorithm can be used in $d$ sequential global cone structures for different $\alpha$; see Corollary 4. In these structures, only one parameter of the facet has to be recalculated.

The algorithm as it is generates the facets one after the other in a deterministic way. It may be modified in order to gradually cover certain specified parts of the zonoid region that are of special interest. E.g., all facets belonging to the lower boundary may be constructed without generating the remaining facets of the zonoid region.

To speed up the procedure, our exact algorithm may also be modified by imposing heuristic rules on the choice of adjacent facets. Such a heuristic rule should choose those facets that determine the zonoid region “more strictly” than others. For instance, as a heuristic, we may prefer redundant facets over non-redundant ones, since a redundant facet cuts more other facets and, by this, can be regarded as a more determining facet. In any case, the exact algorithm serves as benchmark procedure by which any heuristic or approximate procedure, regarding precision as well as speed, can be compared.

<table>
<thead>
<tr>
<th>$d$</th>
<th>$n$</th>
<th>time per facet</th>
<th>number of facets</th>
<th>total time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>0.009989</td>
<td>106</td>
<td>1.062</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>0.012719</td>
<td>241</td>
<td>3.069</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.016651</td>
<td>476</td>
<td>7.929</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0.014992</td>
<td>257</td>
<td>3.862</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0.028511</td>
<td>948</td>
<td>27.044</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>0.053519</td>
<td>2840</td>
<td>152.177</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.017427</td>
<td>377</td>
<td>6.584</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>0.058464</td>
<td>2556</td>
<td>149.637</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>0.201225</td>
<td>13082</td>
<td>2648.587</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>0.018362</td>
<td>377</td>
<td>6.935</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>0.108239</td>
<td>5005</td>
<td>541.735</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>0.599587</td>
<td>38177</td>
<td>22890.445</td>
</tr>
</tbody>
</table>

Table 1: First computational results
8 References


