Computing Halfspace Depth Contours Based on the Idea of a Circular Sequence*

LIU Xiaohui · REN Haiping · WANG Guofu

DOI: 10.1007/s11424-015-3160-y
Received: 22 July 2013 / Revised: 29 January 2014

Abstract This paper presents a new efficient algorithm for exactly computing the halfspace depth contours based on the idea of a circular sequence. Unlike the existing methods, the proposed algorithm segments the unit sphere directly relying on the permutations that correspond to the projections of observations onto some unit directions, without having to use the technique of parametric programming. Some data examples are also provided to illustrate the performance of the proposed algorithm.

Keywords Circular sequence, exact algorithm, halfspace depth contour

1 Introduction

In order to generalize the concept of median to higher dimensions, Tukey heuristically proposed the halfspace depth in 1975. Given a set of data $\mathcal{X}^n = \{X_1, X_2, \cdots, X_n\}$ in the p-dimensional space R^p, he determined the depth value of an arbitrary point x with respect to $\mathcal{X}^n$ as the smallest portion of data carried by any closed halfspacse, as halfspace depth is called, containing x, that is[1],

$$d(x, F_n) = \inf_{u \in \mathcal{S}^{p-1}} \frac{1}{n} \#\{i : u^{\mathrm{T}} x \geq u^{\mathrm{T}} X_i, \ \ i \in \mathcal{N}\}, \tag{1}$$

where F_n denotes the empirical distribution corresponding to $\mathcal{X}^n$, $\mathcal{S}^{p-1} = \{v \in R^p : \|v\| = 1\}$, $\mathcal{N} = \{1, 2, \cdots, n\}$, and $\#\{\cdot\}$ is the number of the data points in set $\{\cdot\}$. For the sake of

LIU Xiaohui
School of Statistics, Jiangxi University of Finance and Economics, Nanchang 330013*, China; Research Center of Applied Statistics, Jiangxi University of Finance and Economics, Nanchang* 330013*, China.*
REN Haiping
School of Software, Jiangxi University of Science and Technology, Nanchang 330013*, China.*
WANG Guofu
School of Mathematics and Statistics, Central South University, Changsha 417100*, China.*
Email: csuliuxh912@gmail.com.

*This research was supported by the National Natural Science Foundation of China under Grant No. 11461029, the Natural Science Foundation of Jiangxi Province under Grant Nos. 20142BAB211014, 20132BAB211015, 20122BAB201023, 20133BCB23014, and the Youth Science Fund Project of Jiangxi provincial education department under Grant Nos. GJJ14350, GJJ14449, KJLD13033.
◇ *This paper was recommended for publication by Editor ZOU Guohua.*

convenience, in the sequel we will use $d_n(x)$ as the abbreviation of $d(x, F_n)$ if no confusion arises.

Since then, this depth notion has been intensively studied in the literature. The related researches include [1–3] among others. It is found that halfspace depth has many favorable properties. For example, it satisfies all the desirable properties of a general statistical depth function defined in [4], namely, affine invariance, maximality at center, monotonicity relative to deepest point, and vanishing at infinity. Furthermore, the concept of halfspace depth exhibits a very strong connection with multivariate quantiles[5]. Therefore, multiple-output quantile regression can possibly be generalized from the celebrated theory of quantile regression; see for example [6, 7] and references therein. Inspirited by [8], many other depth notions have also sprung up like mushrooms in the last decades. Simplicial depth[9], zonoid depth [10], regression depth[11] and projection depth[12] to name but a few. See [13] for a recent review on data depth.

In practice, halfspace depth is often used as a powerful multivariate ordering tool. Points near the boundary of the data cloud have lower depth values, and those deeper inside obtain higher depth values. This can be visualized by means of depth contours:

$$C_k = \{x \in R^p : d_n(x) = k/n\},$$

where $k = 1, 2, \cdots, \kappa^*$ with $\kappa^* = n \cdot \sup_{x \in R^p} d_n(x)$. Clearly, C_k is the boundary of the depth region D_k given by

$$D_k = \{x \in R^p : d_n(x) \geq k/n\}. \tag{2}$$

Many practical applications of halfspace depth have been developed. For example, [14] proposed a depth-based classification technique for unequal prior problems. [15] investigated a nonparametric method for constructing bootstrap confidence regions based on the depth-induced ordering. [16] developed several nonparametric statistics for testing the equality of two or more multivariate populations relying on the statistical depth functions such as the halfspace depth.

An important feature of halfspace depth is its close connection with the projection pursuit methodology[1], which plays a crucial role in the process of generalizing various univariate statistics to higher dimensions. However, a price having to pay is that the exact computation of halfspace depth, as well as its induced estimators, becomes very difficult due to the involvement of the infimum over an infinite number of direction vectors. A similar problem also exists in computing some other projection-based statistics. To facilitate the practical applications, much attention has been paid to this problem by many authors. For example, [17] and [18] considered the exact computation of halfspace depth when $p = 2, 3$, respectively. [19] constructed an exact algorithm to compute the halfspace depth contours for bivariate data.

Nevertheless, computing higher dimensional contours has not been solved until recently when [20, 21] developed two procedures for computing the multiple-output regression quantile regions by using linear programming techniques; see also [6, 7] and references therein. Both of their procedures include the computation of halfspace depth regions as a special case. In this paper, we find that it is still possible to simplify their procedures for this special case. That is, we can segment the search space $\mathcal{S}^{p-1}$ directly based on the permutations that correspond to the projections of observations onto some unit directions, without having to use the technique of parametric programming. A Matlab implementation of the proposed algorithm has also been written. The corresponding codes can now be obtained from the authors through email (csuliuxh912@gmail.com). Simulation studies indicate that, for the special case of computing halfspace depth contours, the proposed algorithm is faster than those of [20, 21] in all cases.

The rest of the paper is organized as follows. Section 2 presents the main idea of how to compute the halfspace depth contours relying on the idea of a circular sequence. Section 3 provides the corresponding algorithm. Several data examples are given in Section 4 to illustrate the performance of the proposed algorithm. Both real and simulated data are considered in this section. Finally, the proof of a main proposition is stated in the Appendix.

2 The Main Idea

In this section, we focus mainly on the discussion of how to obtain a halfspace depth contour based on the idea of a circular sequence[22–24] when $p \geq 2$ (the case of $p = 1$ is trivial). Throughout this paper, we assume that the data $\mathcal{X}^n$ are in general position, namely, every subset of $k+1$ data points generates an affine space of dimension k, $k = 1, 2, \cdots, p-1$. If the data are not in general position, the subsequent discussion and the algorithm need to be modified, e.g., by perturbing the data points by some random noise of a very small magnitude. This assumption is commonly supposed in many existing literature; see for example [25].

Let $Z_{(1)} \leq Z_{(2)} \leq \cdots \leq Z_{(n)}$ be the order statistics based on the univariate random variables $\mathcal{Z}^n = \{Z_1, Z_2, \cdots, Z_n\}$. For a given $1 \leq k \leq n$, denote $\eta_k(\mathcal{Z}^n) = Z_{(k)}$. By the affine equivalence and the fact that $\frac{1}{n}\#\{i : \mathbf{0}_p^{\mathrm{T}} X_i \leq \mathbf{0}_p^{\mathrm{T}} x,\ \ i \in \mathcal{N}\} = 1 \geq d_n(x)$, we replace the constraint $u \in \mathcal{S}^{p-1}$ by $u \in R^p$ in the definition of $d_n(x)$ hereafter, where $\mathbf{0}_m$ denotes the m-dimensional zero vector. In what follows, we will show that $g_k(t) = \eta_k(t^{\mathrm{T}}\mathcal{X}^n)$ is a piecewise linear function over R^p with respect to t, where $t^{\mathrm{T}}\mathcal{X}^n = \{t^{\mathrm{T}}X_1, t^{\mathrm{T}}X_2, \cdots, t^{\mathrm{T}}X_n\}$.

By the idea of a circular sequence, we have that, for any given $u_0\,(\neq \mathbf{0}_p)$, there must exist a permutation, write $(i_{l,1}, i_{l,2}, \cdots, i_{l,n})$, of $(1, 2, \cdots, n)$ such that

$$u_0^{\mathrm{T}} X_{i_{l,1}} \leq u_0^{\mathrm{T}} X_{i_{l,2}} \leq \cdots \leq u_0^{\mathrm{T}} X_{i_{l,n}},$$

and the set $\mathcal{C}_l = \{t \in R^p : \mathbb{A}_l^{\mathrm{T}} t \leq \mathbf{0}_{n-1}\}$ is non-coplanar, where

$$\begin{aligned}\mathbb{A}_l = (&X_{i_{l,1}} - X_{i_{l,k}}, X_{i_{l,2}} - X_{i_{l,k}}, \cdots, X_{i_{l,k-1}} - X_{i_{l,k}},\\ &X_{i_{l,k}} - X_{i_{l,k+1}}, \cdots, X_{i_{l,k}} - X_{i_{l,n}}).\end{aligned} \tag{3}$$

Then, for any $t \in \mathcal{C}_l$, some simple derivations can lead to

$$X_{i_{l,1}}^{\mathrm{T}} t,\ X_{i_{l,2}}^{\mathrm{T}} t,\ \cdots,\ X_{i_{l,k-1}}^{\mathrm{T}} t \leq X_{i_{l,k}}^{\mathrm{T}} t \leq X_{i_{l,k+1}}^{\mathrm{T}} t,\ \cdots,\ X_{i_{l,n}}^{\mathrm{T}} t.$$

That is, for any $t \in \mathcal{C}_l$, we have $g_k(t) = t^{\mathrm{T}} X_{i_k}$ with X_{i_k} being fixed and independent of t. Typically, $\mathcal{C}_l$ forms a polyhedral cone, the number of such cones is finite, and all of these cones together span the whole space R^p, namely, $R^p = \cup_{l=1}^{N_1} \mathcal{C}_l$, where N_1 denotes the number of the cones. This proves the piecewise-linear property of $g_k(t)$.

It is worth mentioning that, based on a permutation $(i_{l,1}, i_{l,2}, \cdots, i_{l,n})$, the practice of using $\mathcal{C}_l = \{t \in R^p : \mathbb{A}_l^{\mathrm{T}} t \le \mathbf{0}_{n-1}\}$ above enjoys many favorable properties. For example, (i) $g_k(t)$ has a simple form over $\mathcal{C}_l$, (ii) there is no need to take care of the ordering existing in both $t^{\mathrm{T}} X_{i_{l,1}}, t^{\mathrm{T}} X_{i_{l,2}}, \cdots, t^{\mathrm{T}} X_{i_{l,k-1}}$ and $t^{\mathrm{T}} X_{i_{l,k+1}}, t^{\mathrm{T}} X_{i_{l,k+2}}, \cdots, t^{\mathrm{T}} X_{i_{l,n}}$, since $Z_{i_{l,1}}, Z_{i_{l,2}}, \cdots, Z_{i_{l,k-1}} \le Z_{i_{l,k}} \le Z_{i_{l,k+1}}, \cdots, Z_{i_{l,n}}$ can also ensure that $Z_{(k)}$ equals $Z_{i_{l,k}}$, which, similar to [26], can be utilized to improve the efficiency of the proposed algorithm.

For a given k, denote

$$E_k = \left\{x \in R^p : u^{\mathrm{T}} x \ge g_k(u), \ \forall u \in R^p\right\}.$$

The following proposition states a strong connection between E_k and D_k (see (2)). The proof of this proposition depends heavily on the permutations corresponding to the direction vectors; see The proof of Proposition 1 in the Appendix. A similar result, but from a different view, can also be found in [7] (Theorem 4.2).

Proposition 1 *For E_k defined above, we have that, for any $1 \le k \le \kappa^*$, there exist a finite number of direction vectors such that*

$$D_k = E_k = \{x \in R^p : \mathbb{M}^{\mathrm{T}} x \ge \boldsymbol{b}\},$$

where $\mathbb{M} = (u_1, u_2, \cdots, u_{M_1})$, and $\boldsymbol{b} = (g_k(u_1), g_k(u_2), \cdots, g_k(u_{M_1}))^{\mathrm{T}}$, where $u_i \in \mathcal{V}$ $(j = 1, 2, \cdots, M_1)$, M_1 is the number of u_i's, and $\mathcal{V} = \{u \in R^p : \|u\| = 1, u$ lies in a vertex of $\mathcal{C}_l, \ 1 \le l \le N_1\}$. That is, $\mathcal{V}$ is the finite vertex set of $\mathcal{C}_l$'s.

Proposition 1 implies that, to compute a halfspace depth contour C_k, it is sufficient to: (S1) Find all the possible cones $\mathcal{C}_l$'s and obtain the finite vertex set $\mathcal{V} = \{u_i\}_{i=1}^{M_1}$, (S2) then compute the contour based on these unit direction vectors.

Note that: 1) Each polyhedral cone $\mathcal{C}_l$ can be identified uniquely by its facets, 2) each facet $\mathcal{F}$ of $\mathcal{C}_l$ can be identified uniquely by the mean of the vertices of $\mathcal{F} \cap [-1, 1]^p$, namely, instead of $\mathcal{C}_l$, we consider the polytopes formed by $\widetilde{\mathbb{A}}_l^{\mathrm{T}} t \le \boldsymbol{b}$, where

$$\widetilde{\mathbb{A}}_l = (\mathbb{A}_l, \mathbb{I}_p, -\mathbb{I}_p), \text{ and } \boldsymbol{b} = \left(\mathbf{0}_n^{\mathrm{T}}, \boldsymbol{I}_{2p}^{\mathrm{T}}\right)^{\mathrm{T}}$$

with $\mathbb{I}_p$ being the $p \times p$ identity matrix, and $\boldsymbol{I}_m$ the m-dimensional vector of ones; 3) each facet $\mathcal{F}$ of $\mathcal{C}_l$ corresponds to a non-redundant constraint condition in $\mathbb{A}_l^{\mathrm{T}} t \le \mathbf{0}_n$. Then, similar to [2], one can find all facets of such a polytope by program qhull[27], and all vertices by means of the dual relationship between vertex and facet enumeration[28]. In Matlab, a similar program, con2vert.m, has been developed by Michael Kleder, and can be downloaded from the Matlab Central File Exchange.

After obtaining all the non-redundant constraints of $\mathcal{C}_l$, it remains to clarify the process leading to the adjacent cone from one given facet $\mathcal{F}$ of $\mathcal{C}_l$. Note that there must be one and only one facet shared between two adjacent cones. That is why one may simply pass through all the cones counter-clockwise when $p = 2$. For $p \ge 2$, the situation is far more complicated.

However, it is still possible to utilize the breadth-first search algorithm to fulfill such tasks. See the Appendix of [20] for a more detailed discussion.

This actually completes the first step (S1). With a finite number of direction vectors $\mathcal{V}$ at hand, the computation of $\{x : \mathbb{M}^{\mathrm{T}}x \geq \boldsymbol{b}\}$ is trivial, and can be fulfilled by using qhull or con2vert.m very easily. Then step (S2) is completed.

3 Algorithm

Based on the discussion above, an algorithm for computing all the possible unit direction vectors $\mathcal{V} = \{u_i\}_{i=1}^{M_1}$ and the corresponding $\{g_k(u_i)\}_{i=1}^{M_1}$ can be given as follows, where $1 \leq k \leq n$.

3.1) Set $\mathcal{A} = \emptyset$, $\mathcal{T} = \emptyset$, and $\mathcal{V} = \emptyset$. Here $\mathcal{A}$ and $\mathcal{T}$ serve as the archives of facets having been and needing to be considered, respectively, and $\mathcal{V}$ the archive of direction vectors.

3.2) Generate a random unit vector u_0 repeatedly until it satisfies $u_0^{\mathrm{T}}X_{i_1} < u_0^{\mathrm{T}}X_{i_2} < \cdots < u_0^{\mathrm{T}}X_{i_n}$. Store the corresponding permutation $(i_1, i_2, \cdots, i_n)$.

3.3) Based on $(i_1, i_2, \cdots, i_n)$, compute the matrix

$$\mathbb{A} = (X_{i_1} - X_{i_k}, X_{i_2} - X_{i_k}, \cdots, X_{i_{k-1}} - X_{i_k}, X_{i_k} - X_{i_{k+1}}, \cdots, X_{i_k} - X_{i_n}).$$

Find all vertices and facets of $\mathcal{C} \cap [-1, 1]^p$, i.e., $\{t : \widetilde{\mathbb{A}}^{\mathrm{T}}t \leq \boldsymbol{b}\}$, where $\mathcal{C} = \{t \in R^p : \mathbb{A}^{\mathrm{T}}t \leq \mathbf{0}_{n-1}\}$, and $\widetilde{\mathbb{A}} = (\mathbb{A}, \mathbb{I}_p, \mathbb{I}_p)$. Drop the facets not being the constraints of $\mathcal{C}$. For each remaining facet $\mathcal{F}$, compute its $\mu_{\mathcal{F}}$ and $\theta_{\mathcal{F}}$. Assign all the $\mu_{\mathcal{F}}$, $(\mu_{\mathcal{F}}, \theta_{\mathcal{F}})$ and the unit direction vectors corresponding to the vertices of $\mathcal{C}$ to $\mathcal{A}_{\text{new}}$, $\mathcal{T}_{\text{new}}$ and $\mathcal{V}_{\text{new}}$, respectively. Here $\mu_{\mathcal{F}}$ is defined to be the mean of all the vertices of $\mathcal{F} \cap [-1, 1]^p$, and $\theta_{\mathcal{F}}$ denotes the outward unit normal vector of the hyperplane containing $\mathcal{F}$, for example, if $\mathcal{F}$ corresponds to the first constraint of $\mathbb{A}^{\mathrm{T}}t \leq \mathbf{0}_{n-1}$, then we have $\theta_{\mathcal{F}} = (X_{i_1} - X_{i_k})/\|X_{i_1} - X_{i_k}\|$.

3.4) Based on $\mathcal{A}_{\text{new}}$ and $\mathcal{T}_{\text{new}}$, update $\mathcal{A}$ and $\mathcal{T}$ by using the following procedure. Set $\mathcal{A}_{\text{temp}} = \emptyset$, $\mathcal{T}_{\text{temp}}^1 = \emptyset$, and $\mathcal{T}_{\text{temp}}^2 = \emptyset$. For every element $\mu_{\overline{\mathcal{F}}}$ of $\mathcal{A}_{\text{new}}$, check whether it exists in $\mathcal{A}$. If it does, add $(\mu_{\overline{\mathcal{F}}}, \theta_{\overline{\mathcal{F}}})$ into the set $\mathcal{T}_{\text{temp}}^1$, otherwise, add $\mu_{\overline{\mathcal{F}}}$ and $(\mu_{\overline{\mathcal{F}}}, \theta_{\overline{\mathcal{F}}})$ into the sets $\mathcal{A}_{\text{temp}}$ and $\mathcal{T}_{\text{temp}}^2$, respectively (in fact, here $\mathcal{T}_{\text{temp}}^1$ contains the facets that have been considered, while $\mathcal{T}_{\text{temp}}^2$ contains those unconsidered). Update $\mathcal{A}$ by adding all the elements $\mu_{\overline{\mathcal{F}}}$ of $\mathcal{A}_{\text{temp}}$ into $\mathcal{A}$. Update $\mathcal{T}$ by first eliminating the facets that exist in both $\mathcal{T}$ and $\mathcal{T}_{\text{temp}}^1$ from $\mathcal{T}$, and then adding the facets of $\mathcal{T}_{\text{temp}}^2$ into $\mathcal{T}$.

3.5) For each direction vector $\widetilde{u}$ of $\mathcal{V}_{\text{new}}$, check whether it exists in $\mathcal{V}$. If it does, do nothing, otherwise, add it into $\mathcal{V}$. Obtain the value of $g_k(\widetilde{u}) = \widetilde{u}^{\mathrm{T}}X_{i_k}$.

3.6) Check whether $\mathcal{T}$ is empty. If it is, terminate the algorithm successfully. If not, obtain a new inner point u_0 based on the first element $(\mu_{\mathcal{F}_1}, \theta_{\mathcal{F}_1})$ of $\mathcal{T}$. Compute u_0's permutation. Go back to Step 3.3. Here a good candidate of u_0 may be $\mu_{\overline{\mathcal{F}}} + \lambda_0\theta_{\mathcal{F}}$, where λ_0 denotes a very small positive magnitude such as 10^{-8}.

After obtaining $\mathcal{V}$ and $\{g_k(u_i)\}_{i=1}^{M_1}$, compute the corresponding matrix $\mathbb{M}$ and vector $\boldsymbol{b}$. Check whether the set $H_k = \{x : \mathbb{M}^{\mathrm{T}}x \geq \boldsymbol{b}\}$ is empty. If it is, this indicates that k is too large, namely, $k > \kappa^*$. If it's not, then we have $D_k = H_k$, and C_k is the boundary of H_k.

From the discussion above, we can see that the main difference between the proposed algorithm and those from [1, 2] is the way of segmenting the search space R^p (or $\mathcal{S}^{p-1}$). In the current algorithm, we segment R^p into a finite number of cones directly based on the permutations. This would be helpful in the computation of some other projection based statistics.

4 Data Examples

4.1 Illustrations

In this section, we present some examples to illustrate the performance of the proposed algorithm. Both real and simulated data are considered here. The depth contours are computed from a Matlab implementation of the algorithm developed above.

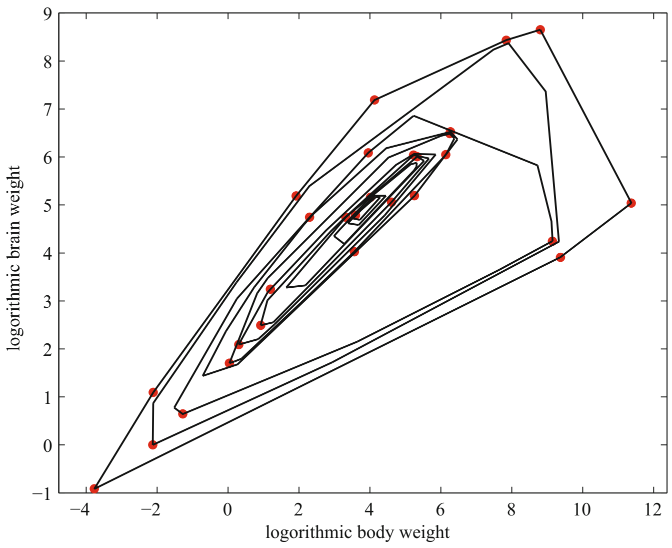
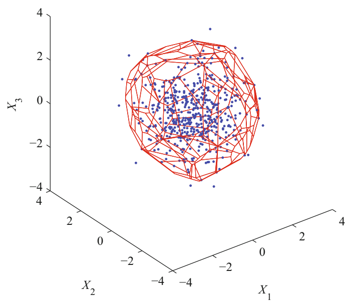


Figure 1 Shown are the logarithms of these data and the halfspace depth contours of depths 1/28, 2/28, $\cdots$, 10/28

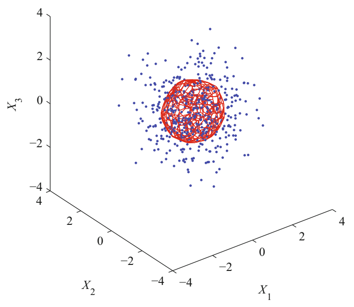
Bivariate data case. We start with a real data example. The data set is taken from Table 7 of [29] (p.57) (see also [21]). It consists of 28 animals' brain weight (in grams) and body weight (in kilograms). We plot the logarithms of these data, and the contours of depths 1/28, 2/28, $\cdots$, 10/28, from the periphery inwards in Figure 1, by using the new method described in the earlier paragraph. Figure 1 shows that the proposed algorithm gives the same result (viz. the same vertices or facets) as the function 'isodepth.r' contained in the R package 'depth' developed by Maxime Genest, Jean-Claude Masse and Jean-Francois Plante.

Furthermore, in order to gain more insight into the performance of the proposed algorithm, we provide a simulated example in the following.

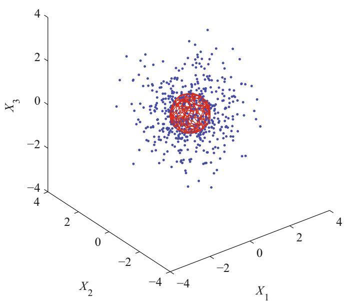
Trivariate data case. In this case, we consider a trivariate data set, which is generated from the trivariate normal distribution $N(\mathbf{0}_3, \mathbb{I}_3)$. The sample size is 500. Three depth contours, with depth values 2/500, 51/500, 101/500, are plotted in Figure 2. The time consumed are about 0.6, 176 and 480 seconds, respectively.



(a) 2/500-contour



(b) 51/500-contour



(c) 101/500-contour

Figure 2 Shown are the 3D halfspace depth contours of 2/500, 51/500 and 101/500. The data points are generated from the trivariate standard normal distribution

4.2 Speed Comparisons

In this subsection, some empirical results are provided to examine the speed of a Matlab implementation of the algorithm developed in this paper. All the results are obtained on a Dell inspiron 1525 laptop with Intel(R) Pentium(R) Dual 2.00GHz, RAM 2.00GB, Windows Vista$^{\mathrm{TM}}$ Home Basic and Matlab 7.8.

We consider three bivariate and a trivariate data sets. For the bivariate case, the data sets are generated from the bivariate standard normal distribution $N(\mathbf{0}_2, \mathbb{I}_2)$, the uniform distribution $U([-0.5, 0.5]^2)$ over the region $[-0.5, 0.5] \times [-0.5, 0.5]$ and the distribution of $X = (Z_1^2, Z_2^2)$, respectively, where (Z_1, Z_2) is bivariate normal distribution with mean $\mathbf{0}_2$ and $\sigma(Z_1) = 1$, $\sigma(Z_2) = 2$ and $\mathrm{corr}(Z_1, Z_2)$=0.4. The sample sizes are 50, 100, 150, 200, 300, 500, 1000, 2000, 5000, 10000, 20000. For the trivariate case, the data set is generated from the trivariate standard normal distribution $N(\mathbf{0}_3, \mathbb{I}_3)$ with sample sizes being 100, 200, 300, 400, 500. For each given n, we investigate the average execution time (in seconds) of computing the depth contours of $(\lfloor \tau n \rfloor + 1)/n$, where $\tau = 0.010, 0.025, 0.050, 0.100, 0.200, 0.400$ for the bivariate case, and $\tau = 0.010, 0.025, 0.050, 0.100, 0.150, 0.200$ for the trivariate case.

The results based on ten repeated computations are reported in Tables 1–4, respectively. Furthermore, for the sake of comparison, we also obtain the average execution time of an benchmark presented in [2]. The benchmark is also implemented in Matlab. Some redundant testing are suppressed by following strictly the recommendations of [2] (see the footnote 3 given in page 42). The times of the code developed in the current paper faster than the benchmark procedure are provided in the parentheses given in Tables 1–4. It is worth mentioning that, in the implementation of the algorithm in [2], there are a few checking statements, which have the potential to place an impact on the run speed. In spite of this limitation, we can see from these tables that the proposed algorithm seems desirable since the code corresponding to this paper is always observed faster than the benchmark in all of these cases (sometimes even more than 20 times), nevertheless.

Table 1 Average execution times (in seconds) of our Matlab code for the bivariate standard normal distribution $N(\mathbf{0}_2, \mathbb{I}_2)$

$n\backslash\tau$	0.010	0.025	0.050	0.100	0.200	0.400
50	0.0024 (13.55)	0.0027 (13.69)	0.0032 (12.40)	0.0040 (17.68)	0.0050 (15.74)	0.0056 (16.83)
100	0.0033 (20.47)	0.0040 (13.06)	0.0053 (16.98)	0.0071 (16.36)	0.0091 (15.87)	0.0105 (16.08)
150	0.0036 (14.98)	0.0057 (12.85)	0.0064 (13.58)	0.0105 (13.46)	0.0135 (13.78)	0.0161 (12.51)
200	0.0051 (20.22)	0.0067 (17.06)	0.0099 (14.47)	0.0130 (13.75)	0.0183 (12.45)	0.0234 (11.79)
300	0.0057 (16.15)	0.0086 (9.98)	0.0134 (10.71)	0.0208 (9.47)	0.0323 (8.79)	0.0415 (8.61)
500	0.0093 (12.74)	0.0163 (8.56)	0.0279 (8.04)	0.0443 (7.18)	0.0689 (6.89)	0.0940 (6.39)
1000	0.0251 (7.63)	0.0480 (6.35)	0.0835 (5.49)	0.1364 (4.89)	0.2247 (4.56)	0.3140 (4.43)
2000	0.0784 (5.89)	0.1670 (4.10)	0.2950 (3.83)	0.5003 (3.48)	0.7996 (3.33)	1.1768 (3.27)
5000	0.4337 (3.40)	0.9597 (2.87)	1.6477 (2.77)	2.7666 (2.73)	4.4812 (2.68)	6.3674 (2.62)
10000	1.5706 (2.57)	3.5009 (2.44)	6.1531 (2.37)	10.6133 (2.34)	17.3482 (2.31)	25.6702 (2.18)
20000	4.8887 (2.82)	10.8677 (2.68)	19.6046 (2.65)	34.0596 (2.56)	55.9406 (2.53)	81.3496 (2.49)

Table 2 Average execution times (in seconds) of our Matlab code for the uniform distribution $U([-0.5,\ 0.5]^2)$ over the region $[-0.5,\ 0.5] \times [-0.5,\ 0.5]$

$n\backslash\tau$	0.010	0.025	0.050	0.100	0.200	0.400
50	0.0034 (9.87)	0.0027 (13.26)	0.0032 (12.80)	0.0043 (17.52)	0.0051 (16.33)	0.0060 (14.74)
100	0.0035 (21.37)	0.0043 (11.38)	0.0047 (17.21)	0.0075 (16.47)	0.0097 (13.83)	0.0114 (14.71)
150	0.0034 (15.71)	0.0044 (14.04)	0.0069 (12.28)	0.0101 (13.08)	0.0138 (12.61)	0.0166 (13.04)
200	0.0051 (18.14)	0.0068 (17.79)	0.0094 (13.71)	0.0128 (13.22)	0.0190 (11.44)	0.0242 (11.63)
300	0.0059 (15.89)	0.0093 (10.55)	0.0136 (11.22)	0.0214 (9.64)	0.0326 (8.83)	0.0441 (8.79)
500	0.0095 (13.60)	0.0168 (8.41)	0.0278 (8.37)	0.0454 (7.24)	0.0710 (6.71)	0.0985 (6.72)
1000	0.0250 (7.84)	0.0514 (6.29)	0.0880 (5.47)	0.1471 (4.99)	0.2345 (4.67)	0.3201 (4.52)
2000	0.0832 (5.10)	0.1719 (4.33)	0.3160 (3.76)	0.5281 (3.50)	0.8474 (3.38)	1.1865 (3.30)
5000	0.4405 (3.58)	0.9934 (2.83)	1.7031 (2.71)	2.9691 (2.60)	4.8365 (2.56)	6.6519 (2.61)
10000	1.5670 (2.65)	3.4668 (2.45)	6.1407 (2.41)	10.6057 (2.36)	17.1853 (2.35)	24.6966 (2.26)
20000	4.8812 (2.77)	10.7299 (2.64)	19.4267 (2.59)	33.5474 (2.55)	55.9516 (2.55)	79.8616 (2.47)

Table 3 Average execution times (in seconds) of our Matlab code for the distribution of $X = (Z_1^2,\ Z_2^2)$, where (Z_1, Z_2) is bivariate normal distribution with mean $\mathbf{0}_2$ and $\sigma(Z_1) = 1$, $\sigma(Z_2) = 2$ and $\mathrm{corr}(Z_1,\ Z_2)=0.4$

$n\backslash\tau$	0.010	0.025	0.050	0.100	0.200	0.400
50	0.0040 (7.83)	0.0029 (13.53)	0.0033 (12.78)	0.0040 (18.07)	0.0047 (18.60)	0.0095 (10.26)
100	0.0030 (29.67)	0.0040 (14.59)	0.0056 (18.39)	0.0066 (16.96)	0.0084 (15.17)	0.0140 (11.71)
150	0.0037 (14.65)	0.0050 (13.89)	0.0072 (12.66)	0.0111 (14.09)	0.0131 (13.27)	0.0195 (10.65)
200	0.0054 (17.39)	0.0072 (15.28)	0.0098 (13.43)	0.0130 (13.58)	0.0188 (11.98)	0.0271 (10.04)
300	0.0069 (14.26)	0.0106 (9.34)	0.0163 (9.92)	0.0231 (10.05)	0.0333 (9.44)	0.0449 (7.94)
500	0.0118 (11.18)	0.0208 (7.46)	0.0317 (8.09)	0.0485 (7.44)	0.0720 (6.69)	0.0926 (6.51)
1000	0.0309 (6.71)	0.0584 (5.89)	0.0932 (5.48)	0.1507 (4.94)	0.2301 (4.68)	0.2893 (4.58)
2000	0.0998 (4.66)	0.1905 (3.99)	0.3172 (3.69)	0.5055 (3.67)	0.7844 (3.48)	1.0280 (3.45)
5000	0.5277 (3.22)	1.0778 (3.06)	1.7756 (3.03)	2.8538 (2.96)	4.4633 (2.87)	5.8639 (2.85)
10000	1.9865 (2.61)	4.0287 (2.50)	6.9797 (2.45)	11.1079 (2.45)	16.8268 (2.45)	22.5844 (2.41)
20000	6.3978 (2.69)	13.0222 (2.65)	22.2440 (2.63)	36.0918 (2.59)	55.6329 (2.56)	76.3986 (2.47)

Table 4 Average execution times (in seconds) of our Matlab code for the trivariate standard normal distribution $N(\mathbf{0}_3, \mathbb{I}_3)$

$n\backslash\tau$	0.010	0.025	0.050	0.100	0.150	0.200
100	0.3605 (2.23)	0.8726 (1.84)	2.5493 (1.50)	6.7343 (1.47)	10.2521 (1.68)	16.3636 (1.50)
200	1.0748 (1.98)	3.2783 (1.70)	9.1170 (1.15)	27.7463 (1.48)	49.8585 (1.36)	64.1989 (1.21)
300	1.8209 (1.80)	6.0221 (1.39)	20.9417 (1.32)	59.1933 (1.47)	103.3586 (1.28)	145.7080 (1.40)
400	3.1733 (2.10)	11.9082 (1.31)	37.9346 (1.14)	104.8924 (1.21)	184.5419 (1.22)	273.5947 (1.38)
500	4.9012 (1.46)	18.4903 (1.27)	62.3985 (1.07)	172.7891 (1.49)	300.7928 (1.11)	479.1082 (1.36)

Table 5 Average number of (not necessarily different) direction vectors considered by the Matlab implementations of [2] (PS2012b) and ours (New) for the trivariate standard normal distribution $N(\mathbf{0}_3, \mathbb{I}_3)$, where the quantities in the parentheses denote the times of the number of direction vectors obtained by the benchmark procedure more than that of ours

Method	$n\ \backslash\tau$	0.010	0.025	0.050	0.100	0.150	0.200
PS2012b	100	216	808	2528	7272	12776	18280
	200	1072	3608	7528	30912	50768	62176
	300	2088	6456	22056	68824	101392	154208
	400	4792	12088	34088	98944	172152	282008
	500	5352	18328	52696	196680	248712	440672
New	100	514 (0.42)	1015 (0.80)	2668 (0.95)	6414 (1.13)	9345 (1.37)	14498 (1.26)
	200	1203 (0.89)	3231 (1.12)	8328 (0.90)	23460 (1.32)	42833 (1.19)	54450 (1.14)
	300	1945 (1.07)	5601 (1.15)	18690 (1.18)	48781 (1.41)	84349 (1.20)	117328 (1.31)
	400	3058 (1.57)	10203 (1.18)	31902 (1.07)	82515 (1.20)	144101 (1.19)	210551 (1.34)
	500	4420 (1.21)	15365 (1.19)	48982 (1.08)	128639 (1.53)	221149 (1.12)	322599 (1.37)

Note that when calculating direction cones and their pertaining hyperplanes, some of them are redundant (containing no facet of the region), and some are not. Ideally, a good implementation is expected to be capable to eliminate as many as possible redundant direction cones from considerations. For a given data cloud $\mathcal{X}^n$ and a fixed τ, the more the redundant direction cones considered, the larger the number of direction vectors (or optimal bases) included in the final results. Therefore, we also investigate the number of direction vectors obtained by these two implementations. Since both of them obtain the same number of direction vectors for bivariate data, we only report the average number of (not necessarily different) direction vectors of ten repeated computations for the trivariate data. Table 5 shows that our new implementation yields a smaller number of direction vectors than that of [2] does in most combinations of n and τ, especially when n and/or τ are large.

5 Concluding Remarks

As a major depth notion, halfspace depth has emerged as a powerful tool in multivariate robust data analysis. The last decades have seen the extensive applications of such a depth notion in inducing many powerful nonparametric methods. Among others, halfspace depth contours are of great interest since they can serve as multivariate quantiles in spaces of high dimension. In fact, it is already seen that halfspace depth contours have a strong connection with the concept of multiple-output quantile regression[7,8]. Such a connection enables one to construct efficient algorithms for exactly computing halfspace depth contours even in dimensions with $p \geq 3$ by using parametric linear programming techniques; see [1, 2].

In this paper, we enriched the toolkits for computing halfspace depth contours by providing an another efficient algorithm. Unlike [1, 2], the new algorithm segments the unit sphere directly relying on the idea of a circular sequence. It does not have to employ the parametric linear programming techniques; see also [25] and [30] for similar discussions in the setting of other depth notions. We found in our simulations that its implementation can run faster than the existing methods in many cases. We wish that the developed algorithm has the potential to help practitioners in the data analysis.

Acknowledgements Thanks the author greatly appreciates his advisor Professor Yijun Zuo for stimulating discussions, thoughtful and constructive remarks and constant encouragement. Thanks also go to Dr. Miroslav Šiman for his great assistance during the preparation of this paper, and to the anonymous reviewers for their careful reading of the earlier manuscript and insightful comments, which led many improvements in this paper.

References

[1] Donoho D and Gasko M, Breakdown properties of location estimates based on halfspace depth and projected outlyingness, *Ann. Statist.*, 1992, **20**: 1808–1827.

[2] Tyler D, Finite sample breakdown points of projection based multivariate location and scatter statistics, *Ann. Statist.*, 1994, **22**: 1024–1044.

[3] Bai Z and He X, Asymptotic distributions of the maximal depth estimators for regression and multivariate location, *Ann. Statist.*, 1999, **27**: 1617–1637.

[4] Zuo Y and Serfling R, General notions of statistical depth function, *Ann. Statist.*, 2000, **28**: 461–482.

[5] Kong L and Mizera I, Quantile tomography: Using quantiles with multivariate data, *Statist. Sinica*, 2012, **22**: 1589–1610.

[6] Hallin M, Paindaveine D and Šiman M, Multivariate quantiles and multiple-output regression quantiles: From L_1 optimization to halfspace depth, *Ann. Statist.*, 2010, **38**: 635–669.

[7] Paindaveine D and Šiman M, On directional multiple-output quantile regression, *J. Multivariate Anal.*, 2011, **102**: 193–392.

[8] Tukey J, Mathematics and the picturing of data, *Proceedings of the International Congress of Mathematicians*, 523–531, Canada Mathematical Congress, Montreal, 1975.

[9] Liu R, On a notion of data depth based on random simplices, *Ann. Statist.*, 1990, **18**: 191–219.

[10] Koshevoy H and Mosler K, Zonoid trimming for multivariate distributions, *Ann. Statist.*, 1997, **25**: 1998–2017.

[11] Rousseeuw P and Hubert M, Regression depth (with discussion), *J. Amer. Statist. Assoc.*, 1999, **94**: 388–433.

[12] Zuo Y, Projection based depth functions and associated medians, *Ann. Statist.*, 2003, **31**: 1460–1490.

[13] Mosler K, Depth statistics, *Robustness and Complex Data Structures*, Springer Berlin Heidelberg, 2013, 17–34.

[14] Ghosh A and Chaudhuri P, On maximum depth and related classifiers, *Scand. J. Statist.*, 2005, **32**: 328–350.

[15] Yeh A and Singh K, Balanced confidence regions based on Tukey's depth and the bootstrap, *J. Roy. Statist. Soc. Ser. B*, 1997, **59**: 639–652.

[16] Chenouri S and Small C, A nonparametric multivariate multisample test based on data depth, *Electronic J. Statist.*, 2012, **6**: 760–782.

[17] Rousseeuw P and Ruts I, Algorithm AS 307: Bivariate location depth, *J. Roy. Statist. Soc. Ser. C*, 1996, **45**: 516–526.

[18] Rousseeuw P and Struyf A, Computing location depth and regression depth in higher dimensions, *Statist. Comput.*, 1998, **8**: 193–203.

[19] Ruts I and Rousseeuw P, Computing depth contours of bivariate point clouds, *Comput. Statist. Data Anal.*, 1996, **23**: 153–168.

[20] Paindaveine D and Šiman M, Computing multiple-output regression quantile regions, *Comput. Statist. Data Anal.*, 2012a, **56**: 840–853.

[21] Paindaveine D and Šiman M, Computing multiple-output regression quantile regions from projection quantiles, *Comput. Statist.*, 2012b, **27**: 29–49.

[22] Edelsbrunner H, *Algorithms in Combinatorial Geometry*, Springer, Heidelberg, 1987.

[23] Dyckerhoff R, Computing zonoid trimmed regions of bivariate data sets, *COMPSTAT* 2000*, Proceedings in Comput. Statist.*, Ed. by Bethlehem J and van der Heijden P, Physica, Heidelberg, 2000, 295–300.

[24] Cascos I, The expected convex hull trimmed regions of a sample, *Comput. Statist.*, 2007, **22**: 557–569.

[25] Mosler K, Lange T, and Bazovkin P, Computing zonoid trimmed regions of dimension $d > 2$, *Comput. Statist. Data Anal.*, 2009, **53**: 2500–2510.

[26] Floyd R and Rivest R, 'Algorithm 489: Select', *Communications of the ACM* 1975, **18**: 173–173.

[27] Barber C, Dobkin D, and Huhdanpaa H, The quickhull algorithm for convex hulls, *ACM Transactions Math. Software*, 1996, **22**: 469–483.

[28] Bremner D, Fukuda K, and Marzetta A, Primal-dual methods for vertex and facet enumeration, *Discrete Comput. Geometry*, 1998, **20**: 333–357.

[29] Rousseeuw P and Leroy A, *Robust Regression and Outlier Detection*, Wiley New York, 1987.

[30] Liu X, Zuo Y, and Wang Z, Exactly computing bivariate projection depth median and contours, *Comput. Statist. Data Anal.*, 2013, **60**: 1–11.

Appendix

Proof of Proposition 1 Let's first focus on the proof of $D_k = E_k$. Note that, for any $x \in E_k$ and $u \in \mathcal{R}^p$, it always holds $u^{\mathrm{T}}x \geq g_k(u) \geq u^{\mathrm{T}}X_{i_{k-1}} \geq \cdots \geq u^{\mathrm{T}}X_{i_1}$, where $(i_1, i_2, \cdots, i_n)$ denotes the permutation corresponding to u. Then we can easily obtain that $\#\{i : u^{\mathrm{T}}x \geq u^{\mathrm{T}}X_i,\ i \in \mathcal{N}\} \geq k$ for any given u, and therefore $d_n(x) \geq k/n$. This proves $E_k \subset D_k$. On the other hand, suppose that there is a $x_0 \in D_k$ such that $x_0 \notin E_k$, namely, there must exist at least one $\overline{u} \in \mathcal{R}^p$ such that $\overline{u}^{\mathrm{T}}x_0 < g_k(\overline{u})$. Note that $g_k(\overline{u}) \leq \overline{u}^{\mathrm{T}}X_{i_{k+1}} \leq \cdots \leq \overline{u}^{\mathrm{T}}X_{i_n}$, then we have $\#\{i : \overline{u}^{\mathrm{T}}x_0 \geq \overline{u}^{\mathrm{T}}X_i,\ i \in \mathcal{N}\} < k$ and therefore $d_n(x_0) < k/n$. This leads to a contradiction since $d_n(x) \geq k/n$ for all $x \in D_k$. This completes the proof of the first part.

For the second part, it is sufficient to prove that, for any given cone $\mathcal{C}_l = \{t \in R^p : \mathbb{A}_l^{\mathrm{T}}t \leq \mathbf{0}_{n-1}\}$, we have $\mathcal{G}_1 = \mathcal{G}_2$, where $\mathbb{A}_l$ is given in display (3), $\mathcal{G}_1 = \{x \in R^p : u^{\mathrm{T}}x \geq g_k(u),\ \forall u \in \mathcal{C}_l\}$, $\mathcal{G}_2 = \{x \in R^p : \widetilde{u}_j^{\mathrm{T}}x \geq g_k(\widetilde{u}_j),\ j = 1, 2, \cdots, m_0\}$, where $\widetilde{u}_1, \widetilde{u}_2, \cdots, \widetilde{u}_{m_0}$ denote the unit direction vectors corresponding to all the vertices of $\mathcal{C}_l$. Clearly, $\mathcal{G}_1 \subset \mathcal{G}_2$ since $\widetilde{u}_j \in \mathcal{C}_l$ for $j = 1, 2, \cdots, m_0$. On the other hand, it is ready to see that: (I) $\mathcal{C}_l$ is convex, and (II) for any $u \in \mathcal{C}_l$, we have $g_k(u) = u^{\mathrm{T}}X_{i_k}$. Then (I), (II) and the fact that $\widetilde{u}_1^{\mathrm{T}}x \geq \widetilde{u}_1^{\mathrm{T}}X_{i_k}$, $\cdots$, $\widetilde{u}_{m_0}^{\mathrm{T}}x \geq \widetilde{u}_{m_0}^{\mathrm{T}}X_{i_k}$ together lead to $(\sum_{j=1}^{m_0} \lambda_j \widetilde{u}_j)^{\mathrm{T}}x \geq (\sum_{j=1}^{m_0} \lambda_j \widetilde{u}_j)^{\mathrm{T}}X_{i_k}$, we can easily show $\mathcal{G}_2 \subset \mathcal{G}_1$, where $\lambda_j \geq 0$, $j = 1, 2, \cdots, m_0$.

This completes the proof of the proposition. ∎