Computation of projection regression depth and its induced median

YIJUN ZUO

Department of Statistics and Probability, Michigan State University
East Lansing, MI 48824, USA
zuo@msu.edu

September 11, 2020
Abstract

Notions of depth in regression have been introduced and studied in the literature. Regression depth (RD) of Rousseeuw and Hubert (1999) (RH99), the most famous, exemplifies a direct extension of Tukey location depth (Tukey (1975)) to regression. The extension of another prevailing location depth, the projection depth (Liu (1992), and Zuo and Serfling (2000)), to regression is called the projection regression depth (PRD) (Zuo (2018a)). The computation issues of RD of RH99 have been discussed in RH99, Rousseeuw and Struyf (1998), and Liu and Zuo (2014). Those of PRD have never been dealt with.

This article addresses the computation issues of PRD and its induced median (maximum depth estimator) in a regression setting, proposing for a given $\beta \in \mathbb{R}^p$ exact algorithms for PRD with cost $O(n^2 \log n)$ ($p = 2$) and $O(npN(n,p)(p^{2.5} + \log n))$ ($p > 2$) and approximate algorithms for PRD and its induced median with cost $O(N_\nu np)$ and $O(Rp(N_\nu N_\beta n + N_\beta + N_\nu N_{Iter} n))$, respectively, where $N(n,p)$ is a number defined based on the total number of $(p - 1)$ dimensional hyperplanes formed by points induced from sample points and the $\beta$, $N_\nu$ is the total number of unit directions $\nu$ tried, $N_\beta$ is the total number of candidate regression parameters $\beta$ tried, $N_{Iter}$ is the total number of iterations carried out in an optimization algorithm, and $R$ is the total number of replications.

Examples and a simulation study reveal that the maximum depth estimator induced from PRD is favorable in terms of robustness and efficiency, compared with the leading regression median, the maximum depth estimator induced from RD.


Key words and phrase: depth in regression, maximum depth estimator, computation, approximate and exact algorithms.

Running title: Computation of maximum regression depth estimator.
1 Introduction

Notions of location depth have been introduced and extensively studied in the literature in the last three decades. Depth notions have found applications in diversity fields and disciplines (see Zuo (2018a) for a review). Among others (Simplicial depth (Liu (1990)), Zonoid depth (Koshevoy and Mosler (1997), Mosler (2002, 2012)) and Spatial depth (Vardi and Zhang (2000), etc.), two prevailing location depth notions are the Tukey halfspace depth (HD) (Tukey (1975)) (popularized by Donoho and Gasko (1992)) and projection depth (PD) (Liu (1992), Zuo and Serfling (2000)) (thoroughly studied in Zuo (2003)), both of which are in the spirit of the projection-pursuit scheme.

One naturally wonders if the depth notion can be extended to a regression setting. Regression depth (RD) of Rousseeuw and Hubert (1999) (RH99), the most famous, exemplifies a direct extension of HD to regression, whereas projection regression depth (PRD), induced from Marrona and Yohai (1993) (MY93) and introduced in Zuo (2018a) (Z18a), is an extension of PD to regression.

Like their location counterparts, the most remarkable advantage of the notion of depth in regression is to introduce directly, the median-type estimator, the maximum (or deepest) regression depth estimator for regression parameters in a multi-dimensional setting. The maximum (deepest) regression depth estimators serve as robust alternatives to the classical least squares or least absolute deviations estimator of unknown parameters in a general linear regression model:

$$\begin{align*}
y &= (1, \mathbf{x}') \beta + e,
\end{align*}$$  \hspace{1cm} (1)

where ‘ denotes the transpose of a vector, and random vector \( \mathbf{x} = (x_1, \ldots, x_{p-1})' \in \mathbb{R}^{p-1} \) and parameter vector \( \beta = (\beta_0, \beta_1')' \in \mathbb{R}^p \) (\( p \geq 2 \)) and random variables \( y \) and \( e \) are in \( \mathbb{R}^1 \). Let \( \mathbf{w} = (1, \mathbf{x}')' \). Then \( y = \mathbf{w}' \beta + e \). We use this model or (1) interchangeably depending on the context.

The maximum depth estimator induced from RD, \( T^*_{RD} \), could resist, asymptotically, up to 33\% (Van Aelst and Rousseeuw (2000) (VAR00))(whereas the one from PRD, \( T^*_{PRD} \), could resist up to 50\% (Zuo (2019a) (Z19a)) contamination without breakdown, in contrast to the 0\% of the classical LS estimator. An illustration of these facts is given in Figure 1, where the data set is given in Table 9 of Chapter 2 of Rousseeuw and Leroy (1987) (RL87)). The original data set contains nine bivariate points.

For any \( \beta \in \mathbb{R}^p \) and joint distribution \( P \) of \( (\mathbf{x}', y) \) in \( \mathbb{R}^p \), RH99 defined the regression depth of \( \beta \), denoted here by \( RD_{RH}(\beta; P) \), to be the minimum probability mass that needs to be passed when tilting (the hyperplane induced from) \( \beta \) in any way until it is vertical. The maximum regression depth estimating functional \( T^*_{RD} \) is then defined as

$$T^*_{RD}(P) = \arg\max_{\beta \in \mathbb{R}^p} RD_{RH}(\beta; P).$$ \hspace{1cm} (2)

Various characterizations of \( RD_{RH}(\beta; P) \) have been given in the literature, e.g. Zuo (2018b).

By modifying the P-estimate of Marrona and Yohai (1993) (MY93) to achieve the scale invariance property, Z18a introduced projection regression depth (PRD), defined based on
Figure 1: Three regression lines for data with or without contamination (solid red for LS, dashed blue for $T_{RD}^*$ and dotted black for $T_{PRD}^*$). (a) Original nine-point data set, $T_{RD}^*$ and $T_{PRD}^*$ are identical. (b) Contaminated data set with one original point $(12, 1)'$ moved to $(12, 12)'$, leading to a drastic change in the LS line while both $T_{RD}^*$ and $T_{PRD}^*$ are unchanged and resist the contamination. (c) Contaminated data set with three original points moved to the points with 3 as their x-coordinates, $T_{RD}^*$ breaks down while both $T_{PRD}^*$ and LS lines are still informative.

The so-called “unfitness” (UF) for a given candidate regression parameter $\beta \in \mathbb{R}^p$:

$$UF(\beta; F(x', y)) = \sup_{v \in S^{p-1}} |R(F(w'v, y-w'\beta))/S(F_y)|,$$

$$PRD(\beta; F(x', y)) = 1/(1 + UF(\beta; F(x', y))),$$

where $F_Z$ stands for the distribution of the d-dimensional random vector $Z \in \mathbb{R}^d$, $w' = (1, x') \in \mathbb{R}^p$, $S^{p-1} = \{u \in \mathbb{R}^p : \|u\| = 1\}$, $R$ will be restricted to the univariate regression functional of the form $R(F(w'v, y-w'\beta)) = T(F_{y-w'\beta}/w'v)$ and it is regression, scale and affine equivariant (see page 116 of RL87 for definitions). $T$ could be a univariate location functional that is location, scale and affine equivariant; $S$ is a scale functional that is translation invariant and scale equivariant (see pages 158-159 of RL87 for definitions), and $S(F_y)$ does not depend on $v$ and $\beta$, see Z18a.

It is not difficult to see that $UF(\beta; F(x', y))$ and $PRD(\beta; F(x', y))$ are the regression counterparts of the outlyingness function $O(\beta; F_Z)$ and the projection depth function $PD(\beta; F_Z)$ in location (Zuo (2003)) respectively.

Examples of $T$ in (3) include mean, quantile, and median (Med), and location functionals in Wu and Zuo (2009). Examples of $S$ in (3) include standard deviation, median absolute deviations from the median (MAD), and scale functionals in Wu and Zuo (2008).

For robustness consideration, in the sequel, $(T, S)$ are fixed and it is the pair (Med, MAD), unless otherwise stated. Hereafter, we write Med($Z$) rather than Med($F_Z$). For this special choice of $T$ and $S$ such that

$$R(F(w'v, y-w'\beta)) = Med_{w'v\neq 0}(\frac{y-w'\beta}{w'v}),$$

$$S(F_y) = MAD(F_y).$$
We have
\[
UF(\beta; F(x', y)) = \sup_{v \in S^{p-1}} \left| \text{Med}_{\omega'v \neq 0} \left( \frac{y - w'\beta}{w'v} \right) \right| \text{MAD}(F_y),
\]
and
\[
\text{PRD} (\beta; F(x', y)) = \inf_{v \in S^{p-1}, w'v \neq 0} \frac{\text{MAD}(F_y)}{\text{MAD}(F_y) + \text{Med}\left( \frac{y - w'\beta}{w'v} \right)}.
\]

Applying the min-max (or max-min) scheme, we obtain the maximum (deepest) projection regression depth estimating functional (also denoted by \(T^*_\text{PRD}\)) w.r.t. the pair \((T, S)\)
\[
T^*_\text{PRD}(F(x', y)) = \arg \min_{\beta \in \mathbb{R}^p} UF(\beta; F(x', y))
\]
\[
= \arg \max_{\beta \in \mathbb{R}^p} \text{PRD} (\beta; F(x', y)).
\]

When a sample \(Z^n = \{(x'_i, y_i)\}', i = 1, \ldots, n\) of \(Z := (x', y)' \in \mathbb{R}^p\) is given, an empirical distribution \(F^n_Z\) based on \(Z^n\) is obtained. Replacing \(F(x', y)\) above by \(F^n_Z\) we obtain all empirical versions. In quest of the exact computation of \(\text{PRD}(\beta; F^n_Z)\) in the next section, we adopt a variant of Med, called \(\text{Med}^*\), which is the smaller of the middle two order statistics. The difference between \(\text{Med}^*\) and Med is bounded by \(O(n^{-3/4}\log n)\) by virtue of Bahadur’s representation for quantiles under some assumptions on the underlying distribution.

While both \(RD_{RH}\) and PRD enjoy desirable properties such as high breakdown robustness, these regression depth functions prove difficult to compute in practice since they involve the projection-pursuit scheme (see Z18a). The computation of \(RD_{RH}\) has been discussed in RH99, in Rousseeuw and Struyf (1998) (RS98), and in Liu and Zuo (2014) (LZ14). The computation issues of PRD and \(T^*_\text{PRD}\) have never been addressed. Presenting exact and approximate algorithms for PRD and discussing the algorithms for the computation of \(T^*_\text{PRD}\) are the main goals of this article.

The rest of the article is organized as follows. Section 2 presents the computation problem and addresses the exact and approximate computation algorithms for \(UF(\beta, F^n_Z)\), and equivalently for \(\text{PRD}(\beta, F^n_Z)\), along with some necessary theoretical preliminary results for the exact computation. Section 3 is devoted to the computation of \(T^*_\text{PRD}(\beta, F^n_Z)\). Section 4 investigates the efficiency of \(T^*_\text{PRD}\). Brief concluding remarks in section 5 end the article.

2 Computation of PRD

2.1 The computation problem

To compute the \(\text{PRD}(\beta; F^n_Z)\), it suffices to compute the \(UF(\beta; F^n_Z)\). Namely, to compute the following quantity:
\[
UF(\beta; F^n_Z) = \sup_{v \in S^{p-1}} \left| \text{Med}_{\omega'v \neq 0} \left( \frac{y - w'\beta}{w'v} \right) \right| / S_y,
\]
where \(w'_i = (1, x'_i)\) and \(S_y = \text{MAD}\{y_i\}\). Under (A1) below, \(P(w'v = 0) = 0, \forall v \in S^{p-1}\).
Denote by $g(v)$ the function inside the absolute operator $| \cdot |$ on the right hand side (RHS) of (8). By (A1), the continuity of $g(v)$ and the closedness of $S^p$ in $v$, there is a $v_0 \in S^p$ with probability one (w.p.1) such that $|g(v_0)|/S_y$ attains the supremum of the RHS of (8). If $g(v_0) \leq 0$, then the oddness of the function of $g(v)$ in $v$ yields $|g(v_0)| = g(-v_0) \geq 0$. Therefore one can drop the absolute operator in (8), but it is kept here purposely.

To facilitate the computation of the $UF(\beta; F^p_Z)$ or $PRD((\beta, F^p_Z))$, write $t'_i = w'_i/r_i(\beta)$ ($\beta$ is suppressed), where $r(\beta) = y - w'\beta$ and $r_i(\beta) = y_i - w'_i/\beta$. If $r_i(\beta) = 0$, replace $t_i$ by a vector with extreme large number (Inf) as its coordinates. Now the computation of $UF(\beta; F^p_Z)$ in (8) is equivalent to the computation of

$$UF(\beta; F^p_Z) = \sup_{v \in S^p} \left| \text{Med}_{t'_i v \neq 0} \left\{ \frac{1}{t'_i v} \right\} \right| / S_y. \tag{9}$$

It is obvious that $g(-v) = -g(v)$. This, in conjunction with the absolute operation on the RHS of (9), implies that it suffices to consider half-spheres on the RHS of (9), instead all of $S^p$. Write $(t_1 \cup \cdots \cup t_n) :\setminus = \{ v \in S^p : t'_i v = 0 \text{ for some } i \}$ and define

$$S^* = S^*(t_1, \cdots, t_n) := S^p :\setminus (t_1 \cup \cdots \cup t_n). \tag{10}$$

Remarks 2.1

(I) Note that the event that $t'_i v = 0$ for some $i$ occurs with probability zero (w.p.0) if (A1): $P(w'v = 0) = 0 \forall v \in S^p$ holds, i.e. the probability mass of any vertical hyperplane is zero. (A1) holds w.p.1 if $(x', y)$ has a density. The latter is also a sufficient condition for (A2): $P(r(\beta) = 0) = 0$ holds. (A2) implies that $t'_i v = \infty$ occurs w.p.0.

(II) Write $k^\gamma_y = 1/t'_i v$ if $v \in S^*$. Hereafter, assume that (i) (A1)-(A2) hold, which guarantees $t_i$ and $k^\gamma_y$ are well defined, (ii) $S_y = 1$ for convenience (the magnitude of $S_y$ does not affect the search of $v_0$, nor the deepest estimator $T_{PRD}$). The $UF(\beta; F^p_Z)$ in (9) is then

$$UF(\beta; F^p_Z) = \sup_{v \in S^*} \left| \text{Med}_i \{ k^\gamma_y \} \right|. \tag{11}$$

Exact computation of $UF(\beta; F^p_Z)$ in (11) is a very challenging task, if not impossible, whereas approximate computation is relatively straightforward (but it is still difficult to assess its accuracy without the benchmark of exact result). We shall address the two approaches separately in the sequel.

2.2 Exact Computation

2.2.1 Theoretical results

We first want to switch the ordering problem in (11) to the ordering problem of $u^\gamma_i := t'_i v = 1/k^\gamma_y$. Let $k^\gamma_{(1)} \leq k^\gamma_{(2)} \cdots \leq k^\gamma_{(n)}$ be ordered values of $k^\gamma_y$. Partition $S^*$ into two parts

$$S_1 = \{ v \in S^* : k^\gamma_{(1)} < 0 \text{ and } k^\gamma_{(n)} > 0 \}; \quad S_2 = \{ v \in S^* : k^\gamma_{(1)} > 0 \text{ or } k^\gamma_{(n)} < 0 \}. \tag{12}$$
Then the UF(β; F_Z^n) in (11) can be expressed as follows:

\[
UF(\beta; F_Z^n) = \max \left\{ \sup_{v \in S_1} |g(v)|, \sup_{v \in S_2} |g(v)| \right\}.
\]  

(13)

\(S_2\) is the set of all \(v\) such that all \(k^\gamma_i\) are positive or negative and is a symmetric (w.r.t. the origin) region of \(v \in S^*\). \(S_2\) is not empty if and only if (iff) the convex hull formed by points of \(t_i\) does not contain the origin. \(S_1\) is the set of all \(v \in S^*\) such that not all \(k^\gamma_i\) \((i = 1, \ldots, n)\) are positive or negative, also a symmetric (w.r.t. the origin) region of \(v \in S^*\). \(S_1\) is not empty if the convex hull formed by points of \(t_i\) contains the origin (in this case, \(S_1 = S^*\)). Fortunately, we do not have to henceforth identify the boundaries of \(S_1\) and \(S_2\).

**Proposition 2.1:** Assume (A1)-(A2) hold. Let \(u^{(1)}_1 \leq u^{(2)}_2 \leq \cdots \leq u^{(n)}_n\) be ordered values of \(u^\gamma_i\). Let \(N^- := \sum_{i=1}^n I(k^\gamma_i < 0)\). The unfitness function of \(\beta\) in (8) can be computed via (13). The latter can be computed as follows.

(i) For \(v \in S_2\)

\[
\sup_{v \in S_2} |g(v)| = \sup_{v \in S_2} \left| \frac{1}{u^{(n+1)/2})} + \frac{1}{u^{(n+2)/2})} \right|/2,
\]

(ii) For \(v \in S_1\), let \(m\) be a non-negative integer. Then

if \(n = 2m + 1\),

\[
\sup_{v \in S_1} |g(v)| = \begin{cases} 
\sup_{v \in S_1} \left| \frac{1}{u^{(N^- - m)}} \right| & \text{if } k^\gamma_{m+1} < 0, \\
\sup_{v \in S_1} \left| \frac{1}{u^{(N^- + m + 1)}} \right| & \text{if } k^\gamma_{m+1} > 0,
\end{cases}
\]

if \(n = 2m + 2\),

\[
\sup_{v \in S_1} |g(v)| = \begin{cases} 
\sup_{v \in S_1} \left| \left( \frac{1}{u^{(1)}} + \frac{1}{u^{(n)}} \right) \right|/2 & \text{if } k^\gamma_{m+1} < 0 \text{ and } k^\gamma_{m+2} > 0, \\
\sup_{v \in S_1} \left| \frac{1}{u^{(N^- + m + 1)}} + \frac{1}{u^{(N^- + m + 2)}} \right|/2 & \text{if } k^\gamma_{m+1} > 0, \\
\sup_{v \in S_1} \left| \frac{1}{u^{(N^- - m - 1)}} + \frac{1}{u^{(N^- - m)}} \right|/2 & \text{if } k^\gamma_{m+2} < 0.
\end{cases}
\]

**Proof:** These are straightforward to verify. Details thus are omitted.

Proposition 2.1 provides a way for the computation of the unfitness via the ordered values of the \(u^\gamma_i\). It also clearly indicates the possibility of the exact computation of \(UF(\beta; F_Z^n)\) or \(PRD(\beta; F_Z^n)\) when \(n\) is odd. The latter is due to the fact that all the suprema on the RHS in Proposition 2.1 (supremum of a single reciprocal of a projected value) can be computed exactly (see the discussion in the sequel).
projected values change their orders

(a) circular-sequence demonstration

an angular region within unit circle

(b) angular region demonstration

Figure 2: (a) \( u \) (or \( ij \)) is perpendicular to the line segment connecting the points \( X_i, X_j \) and between \( u_1 \) and \( u_2 \). When the two points are projected to \( u_1, u_2 \) and \( u \), \( X_i \) precedes \( X_j \) on \( u_1 \) whereas on \( u_2 \) it is reversed. On \( u \) they overlap. (b) a unit circle is cut into pieces (angular regions) by the median sequence. Over each piece, the median of the projected values is the average of the middle two (or one) of the projected values of the same two (or one) fixed points (see Figure 3).

The observation above serves as a strong impetus for the modification of the definition of Med operator in (8), (9), (11), (13) and in Proposition 2.1 as:

\[
\text{Med}^*\{z_1, \ldots, z_n\} := z_{\lfloor (n+1)/2 \rfloor},
\]

where \( z_i \in \mathbb{R}^1 \) and \( z_{(i)} \) are the ordered \( i \)th value. \( \text{Med}^* \) is the same as the original \( \text{Med} \) for odd \( n \). It differs from Med slightly for even \( n \) (smaller of the middle two instead of the average). The difference is bounded by \( O(n^{-3/4} \log n) \) by virtue of Bahadur’s representation. With \( \text{Med}^* \), all functions \( f \) using \( \text{Med} \) in their definition above will be called \( f^* \) hereafter.

This simple modification allows the exact computation of \( \text{UF}^*(\beta; F_n^u) \) when \( n \) is even. The computation essentially becomes the computation of \( \sup_{v \in S_1} |1/u_v^{(j_k)}| \), for some \( j_k \).

**Corollary 2.1** Assume (A1)-(A2) hold. Let \( u_v^{(1)} \leq u_v^{(2)} \leq \cdots \leq u_v^{(n)} \) be ordered values and \( N^- := \sum_{i=1}^n I(k_i^v < 0) \). Replacing the Med in Proposition 2.2 with the Med* in (14), then

(i) For \( v \in S_2 \)

\[
\sup_{v \in S_2} \left| g^*(v) \right| = \begin{cases} 
1/\min_{v \in S_2} u_v^{((n+1)/2)} & \text{if } N^- = 0 \text{ and } n \text{ odd}, \\
1/\max_{v \in S_2} u_v^{((n+1)/2)} & \text{if } N^- = n \text{ and } n \text{ odd}, \\
1/\min_{v \in S_2} u_v^{((n+1)/2)+1} & \text{if } N^- = 0 \text{ and } n \text{ even}, \\
1/\max_{v \in S_2} u_v^{((n+1)/2)+1} & \text{if } N^- = n \text{ and } n \text{ even}, 
\end{cases}
\]

(ii) For \( v \in S_1 \), let \( m \) be a non-negative integer. Then
if $n = 2m + 1$,

$$\sup_{\mathbf{v} \in S_1} |g^*(\mathbf{v})| = \begin{cases} 1/\max_{\mathbf{v} \in S_1} u^v_{(N^v_m - m)} & \text{if } k^v_{(m+1)} < 0, \\ 1/\min_{\mathbf{v} \in S_1} u^v_{(N^v_m + m+1)} & \text{if } k^v_{(m+1)} > 0, \\ \end{cases}$$

if $n = 2m + 2$,

$$\sup_{\mathbf{v} \in S_1} |g^*(\mathbf{v})| = \begin{cases} 1/\max_{\mathbf{v} \in S_1} u^v_{(1)} & \text{if } k^v_{(m+1)} < 0 \text{ and } k^v_{(m+2)} > 0, \\ 1/\min_{\mathbf{v} \in S_1} u^v_{(N^v_m - m+1)} & \text{if } k^v_{(m+1)} > 0, \\ 1/\max_{\mathbf{v} \in S_1} u^v_{(N^v_m - m-1)} & \text{if } k^v_{(m+2)} < 0. \\ \end{cases}$$

**Proof:** It is obvious that if $g^*(\mathbf{v}) > 0$, $\forall \mathbf{v} \in S$, then $\sup_{\mathbf{v} \in S} |1/g^*(\mathbf{v})| = 1/\inf_{\mathbf{v} \in S} g^*(\mathbf{v})$. Likewise, if $g^*(\mathbf{v}) < 0$, $\forall \mathbf{v} \in S$, then $\sup_{\mathbf{v} \in S} |1/g^*(\mathbf{v})| = 1/\sup_{\mathbf{v} \in S} g^*(\mathbf{v})$. For a continuous function over a compact set, we use sup (or inf) and max (or min) interchangeably. These, in conjunction with Proposition 2.1, lead to the desired result.

To facilitate the elaboration of the basic idea to achieve the exact computation via Corollary 2.1, we first invoke the concept of “circular sequence” (see, e.g. Edelsbrunner (1987)).

Given $n$ general points, $t_1, t_2, \ldots, t_n$ (obtained from $Z^{(n)}$ and a $\beta$) in $\mathbb{R}^p$, and any unit vector $\mathbf{v}$, assume that $u_{i_1}^v \leq u_{i_2}^v \leq \cdots \leq u_{i_n}^v$ (recall $u_j^v = t'_j \mathbf{v}$). Then $\{i_1, i_2, \ldots, i_n\}$ forms a permutation of $\{1, 2, \ldots, n\}$ (e.g., see (b) of Fig. 3, where “4321” represents a permutation from the projection of 4 points (labeled as 1, ..., 4) to the direction labeled as “34”).

If one rotates $\mathbf{v}$ counter-clockwise (in $\mathbb{R}^2$), then one will get a sequence of permutations. This periodic sequence of permutations is called a *circular sequence* (see the ones in Fig. 3). In $\mathbb{R}^p$ ($p > 2$), when the unit vector $\mathbf{v}$ rotates on the unit sphere, we again get a sequence of permutations from the subscripts of ordered projected values, a circular/spherical sequence.

**Some observations on circular/spherical sequences**

**O1** The permutation obtained from the projection of the $n$ points on $\mathbf{v}$ is exactly the reverse of the permutation obtained from the projection of them on $-\mathbf{v}$.

**O2** Two successive permutations of a circular/spherical sequence differ only by switching $p$ integers in the sequence (see (a) of Fig. 2).

**O3** The permutation changes only whenever the rotation of $\mathbf{v}$ passes through a direction perpendicular to a $(p-1)$-dimensional subspace formed by $p$ data points in a given data set that is in general position (defined later) (see Fig. 3 and (a) of Fig. 2).

**Proposition 2.2:** Assume (A1)-(A2) hold. Let $V \subset S^*$ be a piece of a unit circle/sphere such that $\forall \mathbf{v} \in V$, $u_{j_1}^v \leq u_{j_2}^v \leq \cdots \leq u_{j_n}^v$. That is, over $V$, $j_1, j_2, \ldots, j_n$ is a fixed permutation of $\{1, 2, \ldots, n\}$. Then (i) $N^v$ is a constant over $V$; (ii) there are no $\mathbf{v}_i \in V$ ($i = 1, 2$) such that $\mathbf{v}_1 \neq \mathbf{v}_2$ and $\mathbf{v}_i \in S_i$. 


Proof:

(i) When $v$ moves over $V$, in order for $N_{v}^{-}$ to change its value, it is obvious that at least one $k_{i}^{x}$ changes from less than zero to greater or equal to zero. That is, $v$ must cross a $v_{0}$ such that $k_{i}^{x} = 0$. The latter happens with probability zero under (A2).

(ii) Assume that there is a $v \in S_{2} \cap V$, then $N_{v}^{-}$ is either 0 or $n$. By (i) there exists no $v_{1} \in V$ such that $v_{1} \in S_{1}$, since the latter means $0 < N_{v_{1}}^{-} < n$, a contradiction. That is, $V \subset S_{2}$. Similarly, if there is a $v \in S_{1} \cap V$, one can conclude that $V \subset S_{1}$.

Now we are in the position to explain the possibility of the exact computation via Corollary 2.1. Essentially, if we can compute the supremum of the $|1/\min_{i \in t_{i}} u_{(j)}^{y}|$ for some $j$ $(1 \leq j \leq n)$ and $i = 1$ or 2, where the min could also be the max, then the exact computation is achieved.

$X^{(n)} := \{x_{i} \in \mathbb{R}^{p}, i = 1, \cdots, n\}$ is said to be in general position (IGP) if any $(p - 1)$ dimensional subspace in $\mathbb{R}^{p}$ contains at most $p$ observations of $X^{(n)}$. It implies that any $p$ points from $X^{(n)}$ will uniquely determine a $(p - 1)$-dimensional hyperplane.

Let $N_{n}^{p}(X^{(n)})$ be the total number of the $(p - 1)$-dimensional hyperplanes formed by (usually $p$) points from $X^{(n)}$. Then $N_{n}^{p}(X^{(n)}) > 1$ if not all points in $X^{(n)}$ lie on a $(p - 1)$-dimensional hyperplane and it is at most $\binom{n}{p}$ if $X^{(n)}$ is IGP. When $N_{n}^{p}(X^{(n)}) = 1$, after some transformations it is greater than 1, one could work with the dimension-reduced data set.

For every $(p - 1)$-dimensional hyperplane $P$ determined by points from $\{t_{i}\}$ (assume that $N_{n}^{p}(\{t_{i}\}) > 1$), there exists a hyperplane $H_{0}$ going through the origin that is perpendicular to the $P$ and cuts the surface of unit sphere (or circle in $\mathbb{R}^{2}$) into pieces. There are possibly at most $N(n, p)$ pieces $P_{k}$, where $N(n, p) = 2\sum_{i=0}^{p-1}(q-1)$ and $q := N_{n}^{p}(\{t_{i}\})$ (see Winder(1966)). The permutation, say $\{j_{1}, \cdots, j_{n}\}$, of the spherical sequence is fixed within each piece $P_{k}$, in light of $O_{2}$ and $O_{3}$. By Proposition 2.2, $P_{k} \subset S_{1}$ or $P_{k} \subset S_{2}$.

That is, $\forall v \in P_{k}$ ($k = 1, \cdots, N(n, p)$), $v_{j_{1}}^{y} \leq v_{j_{2}}^{y} \leq \cdots \leq v_{j_{n}}^{y}$, and $v_{(i)}^{y}$ in Corollary 2.1 is the projected value of the fixed point $t_{j_{i}}$. Namely, $v_{(i)}^{y} = v_{j_{i}}^{y} = t_{j_{i}}^{y} v = \|t_{j_{i}}\| \cos(\alpha)$, where $\alpha$ is the angle between vectors $t_{j_{i}}$ and $v$. In light of the fundamental theorem of linear programming, the solution in Corollary 2.1 must be attained at the boundary of $P_{k}$.

Indeed, over $P_{k}$, we have a perfect linear programming problem by Corollary 2.1: we want $\sup_{p_{j} \in \mathbb{Z}} \{1/u_{(j)}^{y}\}$ $j = 1, 2$, which is equivalent to minimizing (or maximizing in the negative case) $u_{(i)}^{y} = t_{j_{i}}^{y} v$, subject to the constraints: $A_{p \times (n-1)}^{\prime} v \leq 0_{(n-1) \times 1}$ and $\|v\| = 1$, where we write $C_{(n-1) \times p} x = ((C_{1} x), \cdots, (C_{(n-1)} x))^{\prime} \leq b = (b_{1}, \cdots, b_{n-1})^{\prime}$ means $(C_{i} x) \leq b_{i}$ for all $i$, and $A = (A_{1}, \cdots, A_{n-1})_{p \times (n-1)}$ and $A_{i} = t_{j_{i}} - t_{j_{i+1}}$ $(1 \leq i \leq n - 1)$. That is, over the piece $P_{k}$ our supremum problem could be solved by linear programming with the worst case cost $O(n^{3.5} n)$ (see Gonzaga (1995)). So overall, the problem in Corollary 2.1 can be solved in $O((p^{2.5} + \log(n)) np N(n, p))$. (we implicitly assume that $n > p$ throughout).

**Theorem 2.1** For a given $\beta \in \mathbb{R}^{p}$ and $\{t_{i}\} \in \mathbb{R}^{p}$, UF($\beta; F_{n}^{p}$) or PRD($\beta; F_{n}^{p}$) can be computed exactly in (i) $O(n^{2} \log(n))$ for $p = 2$; (ii) $O((p^{2.5} + \log(n)) np N(n, p))$ for $p > 2$.

**Proof:** For the simplicity of description, we assume that $N_{n}^{p}(\{t_{i}\}) = \binom{n}{p}$. The general case
Figure 3: Median-sequence demonstration. (a) Five sample points labeled as “1”,...,“5”. Line “14” cuts the space into two halfspaces. Focusing on the upper right one suffices. Label “ij” means that the labeled ray is perpendicular to the line segment connecting i and j. When v rotates within the angular region formed by “ij” and “ik” (or “kj”, or “jk”), the median of the projected values is the projected value of the repeated label (point) i (or j, or k). The median sequence is “14”, “13”, “35”, “25”, “24” (and “14”). (b) Four sample points labeled as “1”,..., “4”. Line “34” cuts the space into two halfspaces, focusing on the lower right one suffices. Along each ray, there are two permutations listed (as in (a)), due to the overlaps of the projected values of some two points. The labels of the common middle two points in the permutations help to identify the median sequence “34”, “23”, “14”, “12” (and “34”) which form 4 regions corresponding to two middle point pairs “4-2” (formed by “34” (upward), “23” and O), “4-3”, “1-3”, and “3-2”.

could be treated similarly.

(i) Consider the case $p = 2$. That is, the $t_i$ are bivariate points. We show that we can divide the entire circle $||v|| = 1$ into $O(n)$ pieces (arcs) using the so-called median sequence (Zuo and Lai (2011)). These $O(n)$ pieces of arcs further help to divide the entire unit disk into $O(n)$ pieces (each formed by the origin, two radii and a piece of arc) (see (b) of Fig. 2). Over each piece, the middle two numbers (see (b) of Fig. 3) (or one in the odd n case, see (a) of Fig. 3) of the projected values $t'_i v$ are the projected values of some two (or one) fixed points (or point) from $\{t_i\}$. (If we employ Med* in (14), the discussion is similar and easier).

In (a) of Fig. 3, when v rotates over the angular region formed by O, if we consider rays labeled as “ij” and “ik” (or “kj”, or “jk”) then the point labeled as “i” (or “j”) (i.e. the common label) is the single point whose projected value will always be the median of the projected values. The median sequence is the rays “14” (up), “13”, “35”, “25”, “24”, “14” (down) which form 5 angular regions corresponding to point “1” (formed by “14” (up), “13”, and O) “3”, “5”, “2”, and “4”; whereas in (b) of Fig. 3, along various rays labeled as “ij”, there are permutations listed (also in (a)). Each ray corresponds to two equivalent permutations, because along each direction (ray), the projection of some two points overlaps. These permutations help to identify the middle two points and the median sequence. The median sequence is the rays “34” (up), “23”, “14”, “12”, “34” (down) which form 4 regions corresponding to two-point pairs “4-2” (formed by “34” (upward) “23” and O), “4-3”, “1-
3”, “3-2”. When \( v \) rotates over the angular region formed by \( O, \) “23”, and “34” (up), the points “4” and “2” are the two points whose projected values are the middle two of all projected values (they appear in the middle of the permutations along the rays “34” (up), “24” “23”).

Figure 3 just illustrates a general phenomenon in concrete examples. We have generally

**Lemma 2.1:** (i) For \( p = 2 \), there are \( O(n) \) rays that divide the unit disk into \( O(n) \) pieces (cones, or angular regions) \( A_j \), each with the origin as its vertex. Over \( A_j \), the median of the projected values \( \{ t_i'v \} \) is the projected values of some two (or one in the odd \( n \) case) fixed points \( t_{j_1} \) and \( t_{j_2} \). (ii) \( UF^*(\beta; F^p_Z) \) and \( PRD^*(\beta; F^p_Z) \) can be computed exactly in \( O(n^2 \log n) \).

**Proof:** We prove the first part of the Lemma with the traditional Med. It certainly holds if Med’ is employed. The latter is employed in the second part of the Lemma.

For simplicity, label sample points as \( 1, 2, \ldots, n \). For \( i \) there are \( j_1, \ldots, j_k \) labels (or points), such that the line connecting \( i \) to \( j_m \) \( (1 \leq m \leq k) \), labeled as “\( ij_m \)”, cuts the plane into two closed halfplanes so that each contains no less than \( \lfloor (n+1)/2 \rfloor \) points. In Fig. 3, \( i_k \) is 2 (for odd \( n \)) and 3 (for even \( n \)). But they could be larger in other cases.

Identify the unit vector over the unit circle that is perpendicular to the line \( ij_m \) by its polar coordinate angle \( \theta_{ij_m} \) \( (0 \leq \theta_{ij_m} \leq \pi) \) (only halfplane suffices). For each \( i \), keep the two unit vectors that have the minimum and maximum polar angle, respectively. Totally, there are \( O(n) \) such unit vectors. These \( O(n) \) vectors cut the unit disk into \( O(n) \) angular regions each formed by the origin and two unit vectors. By the construction (also see Fig. 3), it is readily seen that over each angular region \( A_j \), the middle two (or one in odd \( n \) case, skip mentioning this case hereafter) integers of the permutations are the same. When \( v \) rotates over each region \( A_j \) the middle two of the projected values \( u_i' \) are the projected values of some two fixed points (say, \( t_{j_1}, t_{j_2} \)). This completes the proof of first part of the Lemma.

Over each piece \( A_j \) (totally \( O(n) \) pieces), invoking Corollary 2.1 and linear programming (considering the boundary directions sufficient), the job can be done in \( O(n^2 \log n) \). we have the second part of the Lemma.

(ii) Consider the cases \( p > 2 \). Before proving this case, we introduce some basic concepts about a convex body. For more details, refer to Fukuda (2004).

A hyperplane \( H = \{ x \in \mathbb{R}^p \mid \mathbf{a}'x = c \}, \mathbf{a} \in \mathbb{R}^p \setminus \{0\}, c \in \mathbb{R} \}, \) A closed halfspace \( H = \{ x \in \mathbb{R}^p \mid \mathbf{a}'x \leq c \}, A \text{ polyhedron } P = \{ x \in \mathbb{R}^p \mid A \mathbf{x} \leq \mathbf{b} \}, A \in \mathbb{R}^{m \times p}, \mathbf{b} \in \mathbb{R}^m, \) A Polytope \( P = \{ x \in \mathbb{R}^p \mid A \mathbf{x} \leq \mathbf{b}, \mathbf{I} \leq \mathbf{x} \leq \mathbf{u} \}, \mathbf{I}, \mathbf{u} \in \mathbb{R}^p, \) A polyhedral cone \( P = \{ x \in \mathbb{R}^p \mid A \mathbf{x} \leq 0 \}, A \text{ hyperplane } H \) of \( \mathbb{R}^p \) is supporting \( P \) (a p-polyhedron or p-polytope) if one of the two closed halfspaces of \( H \) contains \( P \). A subset \( F \) of \( P \) is called a face of \( P \) if it is either \( \emptyset, P \) itself, or the intersection of \( P \) with a supporting hyperplane. The faces of dimension \( 0, 1, \dim(P) - 2 \) and \( \dim(P) - 1 \) are called the vertices, edges, ridges and facets, respectively.
Obviously, exact computation is achieved if we can obtain the RHS of display (13). For the latter, we appeal to Corollary 2.1. To implement the corollary, essentially, we need to compute the supremum of the $|1/\min_{v \in S_i} u_{(j)}^v|$ for a $j$ ($1 \leq j \leq n$) and a $i \in \{1, 2\}$ in Corollary 2.1, where the min could also be the max, then the exact computation is achieved.

Let $t_{ij}, \ldots, t_{in}$ be the permutation induced by projecting data points to the vector $v$ above, that is, $u_{(j)}^v \leq \cdots \leq u_{in}^v$, or equivalently, $t_{ij}^v \leq \cdots \leq t_{in}^v$ (recall $u_j^v = t_{ij}^v$). So for $u_{(j)}^v$ above, we have $u_{(j)}^v = u_{ij}^v = t_{ij}^v v$.

If $t_{ij}^v$ above is a fixed constant vector, then the optimization in Corollary 2.1 can be solved by linear programming. Over $P_k$ introduced below, the permutation is fixed, then $t_{ij}^v$ is indeed a fixed (constant vector). Then the optimization solution in Corollary 2.1 can be obtained. Consequently, the exact computation of UF or PRD is achieved.

In light of the (O3), when a vector $v$ moves on the surface of unit sphere, its induced permutation changes only when it crosses a hyperplane ($H_0$) that goes through the origin and is perpendicular to another hyperplane ($H_1$) that is formed by sample points from $\{t_i\}$.

For every $(p-1)$-dimensional hyperplane $H_1$ determined by points from $\{t_i\}$ (assume that $N_n^p(\{t_i\}) > 1$), there is a hyperplane $H_0$ that contains the origin and is perpendicular to the $H_1$ and cuts the surface of unit sphere (or circle in $\mathbb{R}^2$) into pieces. There are possibly at most $N(n, p)$ pieces $P_k$, where $N(n, p) = 2\sum_{i=0}^{p-1} \binom{q-1}{i}$ and $q := N_n^p(\{t_i\})$ (see Winder (1966)).

The permutation, say $\{i_1, \ldots, i_n\}$, of the spherical sequence is fixed over each piece $P_k$, in light of (O3). By Proposition 2.2, either $P_k \subset S_1$ or $P_k \subset S_2$.

To identify all $N(n, p)$ disjoint pieces of $P_k$, it suffices, for the task we have, to identify all $N(n, p)$ distinct permutations. For the latter, it suffices to identify a unit vector that belongs to the interior of each $P_k$, $k = 1, \ldots, N(n, p)$.

For the last step above, one can first identify a vector on the boundary of $P_k$, then by small perturbation of the vector, one gets a unit vector inside of $P_k$ which then will induce projected values without ties (previous boundary vector will lead to $p$ (when $\{t_i\}$ is IGP case) or more than $p$ (not IGP case) identical projected values).

To identify a vector on the boundary of $P_k$, it suffices to identify a vector on one of facets of the polyhedral cone which has $P_k$ as its top part. Notice that each $P_k$ identifies a polyhedral cone which is bounded by hyperplanes $H_0$'s each of which goes through the origin and part of each hyperplane forms one of facets of the cone.

The direction from the origin to any other point on the intersection hyperline of two hyperplanes $H_0$'s is the solution of the vector sought above. Denote the direction by $u$ ($u$ could also be obtained more costly via the origin and any vertex of the cone through vertex enumeration (see Bremner et al., 1998, Paindaveine and Šiman (2012) and Liu and Zuo (2014)).

The $u$ above not only lies in the facet of one cone but also lies in that of an adjacent cone which shares the common intersection hyperline (edge or ridge) with the former cone. Tiny perturbation of of $u$ in opposite directions will lead $u$ entering the interiors of the two adjacent
cones. There might be more than two cones that are adjacent. Thus, every \( u \) might yield two or more new permutations (the scheme in the algorithm yields up to \( 8^*(p-2) \) distinct ones).

Keep a counter of the total number of distinct permutations obtained. For each permutation, find out a solution \( v \) by linear programming using the linear constraints matrix \( A \) given in the paragraph immediately before the theorem 2.1, using the \( v \) and Corollary 2.1 to update \( |g^*(v)| \). Keep constructing \( u \)'s by constructing more \( H_1 \)'s and \( H_0 \)'s until we have a total \( N(n, p) \) distinct permutations or until \( UF^* \) can not be improved.

For a given data set (or \( \{t_i\} \)), the total number of \( H_1 \)'s is fixed, but for each \( H_1 \), there are infinitely many \( H_0 \)'s (\( p > 2 \)) which go through the origin and are perpendicular to \( H_1 \). So by utilizing different \( H_0 \)'s one can always obtain all distinct permutations in theory. If one obtains \( N(n, p) \) distinct permutations which means that each piece of \( P_k \) has been visited (or all relevant directions \( u \)'s have been obtained), the resulting \( UF^* \) (or \( PRD^* \)) is exact in theory. In practice, however, not every distinct permutation makes real contribution for updating \( UF^* \). In the latter case, the stopping rule “until \( UF^* \) can not be improved” becomes handy.

We have the exact \( UF^* \) eventually and the final total cost is at most \( O(N(n, p)(p^{3.5}n + np\log(n))) \) or \( O((p^{2.5} + \log(n))npN(n, p)) \). This completes the proof of the theorem. 

**Remark 2.2:** Note that the theorem holds w.r.t. the original \( UF(\beta; F^n_Z) \) and \( PRD(\beta; F^n_Z) \) when \( n \) is odd. That is, there is no need to invoke Med* for odd \( n \). 

### 2.2.2 Exact computation algorithms

**(I) Algorithm for the exact computation of \( UF^*(\beta; F^n_Z) \) and \( PRD^*(\beta; F^n_Z) \) in \( \mathbb{R}^2 \)**

Before listing the key steps of the algorithm, we make some comments.

(i) Directions that are perpendicular to the line segment connecting \( t_i \) and \( t_j \) do not belong to the set \( \{t_1 \cup \cdots \cup t_n\}^\perp \). The directions in the latter set are excluded from \( \mathbb{S}^{p-1} \). However, they could be the boundary of angular regions and/or of \( S^* \), so we will have to include them in our calculation.

(ii) \( \sup_{v \in S_i} |g^*(v)| \) (\( i = 1, 2 \)) can be obtained along the median sequence and the directions given in (i) above.

**Exact Algorithm EA-UF2D**

**Input** a \( \beta \) and \( n \) data points \( Z^{(n)} = \{(x_i, y_i)\} \) in \( \mathbb{R}^2 \); **Output** \( UF^*(\beta; F^n_Z) \) and \( PRD^*(\beta; F^n_Z) \).

**Initial Step:** (i) Obtain \( N := N_0^n(\{t_m\}) \) unit vectors \( u_k(i, j) \) that are perpendicular to the all possible \( N \) hyperplanes formed by \( t_i \) and \( t_j \) from \( \{t_m\}, k = 1, \cdots, N \). (ii) Sort \( u_k(i, j) \) according their polar angles such that \( \alpha_{i_1} \leq \alpha_{i_2} \leq \cdots \leq \alpha_{i_N} \). (iii) Record the pair \( (i, j) \) associated with \( \alpha_k \) as the pair \( (i^k, j^k) \).

Set a nine-component matrix \( I_0 \) with extremely large (or small) values corresponding to min (or max) in Corollary 2.1 nine cases. Let \( k = 0, M_k = I_0, u_k = (1, 0)' \) and \( i_1, \cdots i_n \)
be a permutation induced by \( u_k \). If \( i \neq j \) and \( t'_i u_k = t'_j u_k = t'_i u_k \), set \( m^1_k = i^k = i \) and \( m^2_k = j^k = j \); otherwise \( m^1_k = m^2_k = i^k = j^k = i_{kk} \), where \( kk = [(n+1)/2] \).

**Loop step:** While \( (k <= N+1) \) \{ Let \( \mathbf{v} = u_k \). Update \( M_k \) according to Corollary 2.1. Let \( k = k + 1 \), \( \mathbf{v} = (\cos(\alpha_k), \sin(\alpha_k)) \). If the set \( S^m_{k-1} := \{i^k, j^k\} \) intersects with the set \( S_k := \{i^*, j^*\} \), then let \( m^1_k = i^k, m^2_k = j^k \); otherwise, let \( k = k + 1 \), \( m^1_k = m^2_k = i_{kk} \), \( i \in \{1, 2\} \). \}

For \( (i \in 1:n) \) \{ get \( \mathbf{v}_i \) that is perpendicular to \( t_i \), using \( \mathbf{v}_i \) to update \( M_{N+i-1} \) according to Corollary 2.1 and to obtain \( M_{N+i} \).

**Final step:** Set the maximum no-zero element of \( M_{N+n} = I_0 \) be the UF*\((\beta; F^n_Z)\).

(II) Algorithm for the exact computation of UF*\((\beta; F^n_Z)\) and PRD*\((\beta; F^n_Z)\) in \( \mathbb{R}^p \), \( p > 2 \)

**Exact Algorithm (EA-UFHD)**

**Input** a \( \beta \) and \( n \) data points \( \mathbf{Z}^{(n)} = \{(x^i, y^i)'\} \) in \( \mathbb{R}^p \); **Output** UF*\((\beta; F^n_Z)\) and PRD*\((\beta; F^n_Z)\).

(a) Compute \( N(n, p) \) and call it by \( N \), let \( k_{pm} = 0 \). \( (O(p), \text{assume that } q = \left(\binom{n}{p}\right)) \).

(b) Construct two non-parallel hyperplanes \( H_i \) \( (i = 1, 2) \) (each of which is formed by \( p \) points from \( \{t_i\} \)) with normal vectors \( \mathbf{v}_i \), \( i = 1, 2 \), respectively. \( (O(np + p^2)) \)

Find two hyperplanes \( H^{ot}_i \) that are through the origin and perpendicular to \( H_i \) and with normal vectors \( \mathbf{u}_i \), \( i = 1, 2 \), respectively. \( (O(p^3)) \)

Let \( \mathbf{U} \) be the unit vectors matrix each of its \( (p-2) \) columns is perpendicular to both of \( \mathbf{u}_1 \) and \( \mathbf{u}_2 \). \( O(p) \)

(c) For each column vector of the \( \mathbf{U} \) above, call it by \( \mathbf{u} \), introduce eight vectors \( \pm \mathbf{u} \pm \mathbf{v}_1 \pm \mathbf{v}_2 \). For each of eight vectors, obtain its induced permutation, and store it in a matrix if it is a new one. Update the total number of distinct permutations \( k_{pm} \). \( O(pn \log n + nk_{pm}) \)

For each distinct new permutation, find a solution \( \mathbf{v} \) via linear programming, using it to update \( |g^*(\mathbf{v})| \) via Corollary 2.1. Also use \( \mathbf{v}_1 \) and \( \mathbf{v}_2 \) to update \( |g^*(\mathbf{v})| \) via Corollary 2.1. \( (O(n(p^{2.5} + p \log n))) \)

(d) while \( (k_{pm} < N) \) \{ do (b), (c) above until either \( k_{pm} = N \) or UF cannot be improved. \} \( (O(nN(p^{2.5} + p \log n))) \)

(e) Output the final UF*\((\beta; F^n_Z)\) via Corollary 2.1 and (13). \( (O(1)) \)

Overall cost in the worst case is \( O(Nnp(p^{2.5} + \log n)) \).

EA-UFHD exactly follows the idea given in the proof of Theorem 2.1. Since there are infinitely many \( H^{ot}_i \)'s for each \( H_i \), in theory one can get all distinct permutations, equivalently all desirable unit directions \( \mathbf{u} \), the final UF* \( \text{or PRD}^* \) obtained is in theory exact.
Although UF\textsuperscript{*} or RPD\textsuperscript{*} can be computed exactly by the algorithms above, they are not feasible in practice for \( p > 3 \), with the worst case time complexity no better than \( O(npN(n, p) \log n) \). In the following, we discuss some more practically feasible approximate algorithms.

### 2.3 Approximate computation

Approximate computation of statistical depth functions is common and has been carried out in Rousseeuw and Struyf (1998), Dyckerhoff (2004), Cuesta-Albertos and Nieto-Reyes (2008), Chen et al. (2013), and Zuo (2018) and in the references cited therein.

In this section, we stick to the standard Med and do not need to utilize the modified version Med\textsuperscript{*} any more. The latter is for the exact computation for even \( n \) only.

Here we present three approximate algorithms. The first one is a straightforward naive one. It randomly selects a fixed number \( N \) directions from a distribution (e.g. uniform on the hypersphere), and decorrelating the data before calculation the UF(\( \beta; F^n \)) defined in (9) along those directions.

**Approximate algorithm AA-UF-1**

**Input** a \( \beta \) and \( n \) data points \( Z^{(n)} = \{ (x_i', y_i') \}' \) in \( \mathbb{R}^p \); **Output** UF(\( \beta; F^n_Z \)) and PRD(\( \beta; F^n_Z \)).

(a) Randomly select \( N \) unit directions \( v \in S^{p-1} \) according to a uniform distribution on the hypersphere, use the formula given in (9) or (8) to calculate/update \( \sup_{v \in S^{p-1}} |g(v)| \).

(Overall cost \( O(npN) \), the cost to find median can be as low as \( O(n) \)).

The second approximate algorithm below employees the idea in EA-UFHD. It considers the directions that represent the edges of the convex cones, where the cones are stemming from the origin and partitioning the entire sphere \( S^{p-1} \) into disjoint (convex) pieces.

When \( v \) moves over each piece, the permutation induced is fixed. By the fundamental theorem of linear programming, the solution of the maxima or minima of a linear function over a convex polygonal region occurs at the region’s corners. (Note that we no longer have linear functions without using Med\textsuperscript{*}, but \( |\text{Med} - \text{Med}^*| \) is extremely small for large \( n \)).

**Approximate algorithm AA-UF-2**

**Input** a \( \beta \) and \( n \) data points \( Z^{(n)} = \{ (x_i', y_i') \}' \) in \( \mathbb{R}^p \); **Output** UF(\( \beta; F^n_Z \)) and PRD(\( \beta; F^n_Z \)).

(a) Compute the \( t_i, i = 1, \cdots, n \). (total cost \( O(np) \))

(b) Sample two sets \( P_i \), each with \( p \) points, from \{\( t_i \}\}. Construct two hyperplanes \( H_i \) with normal vectors \( v_i \), uniquely determined by \( P_i \), respectively. Try different \( P_2 \) until \( v_2 \) is not parallel to \( v_1 \). (total cost \( O(p^3) \))

(c) Construct two hyperplanes \( H_i^\perp \) (with normal vectors \( u_i \)) that is through the origin and perpendicular to \( H_i \), respectively. (total cost \( O(p^3) \))
(d) Obtain \( \mathbf{v} = \mathbf{u}_1 \times \mathbf{u}_2 \) and \( \mathbf{v}_0 = \mathbf{v}/\|\mathbf{v}\| \); use \( \mathbf{v}_0 \) and the formula given in (9) to calculate/update \( \sup_{\mathbf{v} \in \mathbb{S}^{p-1}} |g(\mathbf{v})| \). (total cost \( O(np) \))

(e) Repeat (b)-(d) \( N \) times. (total cost \( O(N(np+2p^3)) \)). Overall cost \( O(N(np+2p^3)+np) \). \( \blacksquare \)

The one below uses \( N \) normal vectors of the hyperplanes determined by \( p \) points from \( \{t_i\} \).

Approximate algorithm AA-UF-3

Input a \( \beta \) and \( n \) data points \( Z^{(n)} = \{(x'_i, y_i)'\} \) in \( \mathbb{R}^p \); Output \( \text{UF}(\beta; F_{Z}^{n}) \) and \( \text{PRD}(\beta; F_{Z}^{n}) \).

(a) Compute the \( t_i, i = 1, \cdots, n \). (total cost \( O(np) \))

(b) Sample \( p \) points from \( \{t_i\} \), find the normal vector \( \mathbf{v} \) of the hyperplane determined by them. Along \( \mathbf{v} \), use the formula (9) to calculate/update \( \sup_{\mathbf{v} \in \mathbb{S}^{p-1}} |g(\mathbf{v})| \). (total cost \( O(p^3 + np) \))

(c) Repeat (b) \( N \) times. (total cost \( O(N(p^3 + np)) \)). Overall cost \( O(N(np+p^3)+np) \). \( \blacksquare \)

2.4 Examples

To better understand the algorithms in the last two subsections, we present some examples. Algorithms for the deepest regression lines discussed in Section 3 are also employed below.

Example 2.4.1 Performance of exact and approximate algorithm. Here we examine the performance of the exact versus the approximate algorithm (EA-UF2D v.s. AA-UF-1) for computing the UF, w.r.t. their accuracy, speed, and estimated mean squared errors.

(a) three lines and six points

(b) three lines and five points

Figure 4: (a) Solid red: the deepest regression line induced from PRD. Dashed blue: the least squares line based on five points without the horizontal outlier. Dotted green: the least squares line based on all six points. (b) Solid red: the deepest regression line induced from PRD w.r.t five points. Dashed blue: the least squares line based on five points without the horizontal outlier. Dotted green: the deepest regression line induced from RD of RH99 w.r.t. five points.
For illustration purpose, we utilize the data set given in Huber and Ronchetti (2009) Exhibit 7.1. Three regression lines obtained w.r.t. the data. The first line $T_{PRD}^*$ is the maximum PRD line ($\beta_1 = (-1.7317456, -0.8184845)^\prime$); the second one (LS without outlier) is the least squares line without the horizontal outlier ($\beta_2 = (-1.87, -0.977)^\prime$); the third one (LS) is the least squares line ($\beta_3 = (0.07, -0.08)^\prime$) w.r.t. all points. See (a) of Figure 4.

Next, we calculate the unfitness of the three lines ($\beta$’s). First, the unfitness, reported in Table 1, is calculated without the horizontal outlier for the fairness of the comparison of exact and approximate algorithms. That is, they are calculated w.r.t. just five points ($n$ is odd, one obtains $UF$ not $UF^*$ from both EA and AA).

Second, using all six points ($n$ is even), the results are very similar to those in Table 1 and details are omitted. As an example, the $UF^*$ from EA are 0.8340, 1.2113, and 2.3367 and $UF$ from AA (mean of 1000 replications) are 0.5246, 0.6855 and 2.1846, respectively.

Consistent with expectations, $T_{PRD}^*$ has the lowest $UF$ (or $UF^*$), $\beta_2$ has the second lowest $UF$ (or $UF^*$), and the $\beta_3$ has the highest $UF$ (or $UF^*$). That is, in terms of the $UF$ (or $UF^*$) ordering, $T_{PRD}^*$ is the best choice among the three while $\beta_3$ is the worst, fitting with the intuitive comprehension of (a) of Figure 4.

At the same time, it is not difficult to determine the regression depths (RD) of RH99 of the three lines, they are $2/6$, $1/6$, $1/6$, respectively. (For simple methods of calculation of RD, see RH99 or RS98). That is, the least square line $\beta_3$ is as deep (or good) as the line $\beta_2$, while both are less deep than PRD line $T_{PRD}^*$ in terms of $RD_{RH}$ ordering, which is somewhat inconsistent with the intuitive comprehension of (a) of Figure 4. Of course, the comparison here is not very fair since the different methods (PRD vs RD) based on different objective criteria and $\beta_2$ and $\beta_3$ use a different number of total points.

In (b) of Figure 4, all three lines are calculated w.r.t. just five points without the outlier. The solid red line is the deepest line from PRD with $\beta_1 = (-2.083114, -1.009444)^\prime$, the dashed blue line is the same as in (a), the dotted green line is the deepest line from RD of RH99 with $\beta_3 = (-1.58, -0.77)^\prime$. The RDs of the three lines are $2/5$, $1/5$, $3/5$, respectively. This time, $T_{RD}^*$, the line induced from RD, as expected, becomes the deepest one.

Table entries (a,b,c,d) are a:= mean of $UF$, b:=standard deviation of $UF$, c:=time consumed (in seconds), d:=number of unit vectors used.

<table>
<thead>
<tr>
<th></th>
<th>$\beta_1$ (line $L_1$)</th>
<th>$\beta_2$ (line $L_2$)</th>
<th>$\beta_3$ (line $L_3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>(0.59, 0, 1.11e-3, 13)</td>
<td>(0.87, 0, 1.28e-3, 15)</td>
<td>(2.88, 0, 1.30e-3, 15)</td>
</tr>
<tr>
<td>AA</td>
<td>(0.58, 4.1e-3, 1.657e-2, e+3)</td>
<td>(0.86, 4.8e-3, 1.662e-2, e+3)</td>
<td>(2.85, 3.4e-2, 1.664e-2, e+3)</td>
</tr>
</tbody>
</table>

Table 1: Performance of exact and approximate algorithms w.r.t. different $\beta$’s (lines).

In Table 1, the calculation of approximate algorithm (AA) is repeated 1000 times to mitigate the randomness; the mean and the standard deviation of 1000 UF’s are calculated (note that it is not $UF^*$). 1000 unit vectors are used in the calculation per replication, and the time consumed per replication is reported in the table.
The table reveals that the exact algorithm (EA) is much faster than the AA. The former used no more than 15 unit vectors whereas the latter, using 1000 vectors, still returned a smaller (under-approximated) UF than the exact one. This is exact the beauty of EA. If the number of unit vectors used is increased to $10^4$, then the UF from AA is still smaller than the one from EA which just employed 13 unit vectors, in the $\beta_1$ case. Since there is no fluctuation in EA, all the s.d.’s are zero. The time reported is the average of per replication from 1000 replications. Observations above hold only in this special ($n = 5$ and $p = 2$) example. Note that when $n$ and/or $p$ increase, EA is no longer feasible in practice.

Results above and below in this section are all based on matlab codes on a desktop: Intel(R)Core(TM) i7-2600 CPU @ 3.40GHz 3.40GHz. All matlab as well as R codes in this and next sections are downloadable via https://www.stt.msu.edu/users/zuo/Codes/readme.txt.

**Example 2.4.2 Performance of exact algorithm versus approximate algorithms**

In the last example, the dimension $p = 2$ and $n = 5$ are too limited in practice. In this example, we consider standard normal random points with $p = 3$ and $n = 10$. For this fixed small data set, we employ the EA and AA’s to compare their performance. Here we seek the best output from them, that is the largest UF (note that UF is defined based on the supremum of univariate unfitness. So generally speaking, the larger the better).

For the EA, the stopping rule of the algorithm is the total number of distinct permutations ($N_{perm}$) used by it whereas for the AA’s, the stopping rule is the total number of unit directions ($N_v$) employed by them. The outputs from different algorithms are listed in table below. Column one is the number of distinct permutations used by the EA-UFHD, the second column is the number of the while loops that need to be executed to get the desired total distinct permutations. Four column is the total number of directions used by the AA’s.

<table>
<thead>
<tr>
<th>$N_{perm}$ used</th>
<th>$N_{while-loop}$ executed</th>
<th>UF</th>
<th>$N_v$ used</th>
<th>AA-UF-1 UF</th>
<th>AA-UF-2 UF</th>
<th>AA-UF-3 UF</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>18</td>
<td>4.3505</td>
<td>120</td>
<td>4.0414</td>
<td>4.3862</td>
<td>5.4190</td>
</tr>
<tr>
<td>200</td>
<td>35</td>
<td>5.1988</td>
<td>1,000</td>
<td>5.8134</td>
<td>5.2446</td>
<td>5.4190</td>
</tr>
<tr>
<td>300</td>
<td>61</td>
<td>6.3708</td>
<td>10,000</td>
<td>6.6369</td>
<td>6.4582</td>
<td>5.4190</td>
</tr>
<tr>
<td>400</td>
<td>92</td>
<td>8.2669</td>
<td>100,000</td>
<td>6.7699</td>
<td>6.6985</td>
<td>5.4190</td>
</tr>
<tr>
<td>500</td>
<td>136</td>
<td>8.2669</td>
<td>1,000,000</td>
<td>6.7924</td>
<td>6.8182</td>
<td>5.4190</td>
</tr>
<tr>
<td>600</td>
<td>187</td>
<td>8.2669</td>
<td>10,000,000</td>
<td>6.8930</td>
<td>6.8675</td>
<td>5.4190</td>
</tr>
</tbody>
</table>

Table 2: Performance comparison of exact versus approximate algorithms on a fixed standard normal data set with the size 3 by 10.
Inspecting the table immediately reveals that (i) in order to use 120 distinct permutations, one needs to run the while loop 18 times in the EA and can get $4.3505$ for UF. Note that each single while loop yields a $p$ by $(p-2)$ unit vector matrix $u$ (in this case a single column vector) and each column of $u$ will induce eight unit vectors (see (c) of the EA-UFHD), these eight vectors will lead to up to eight distinct permutations in each while loop. This explains why only 18 while loops needed. The number 120 equal $\binom{10}{3}$ in this case is exactly the total possible planes formed by any three sample points from a data set (IGP) of size 10 in a three dimensional space; (ii) if one uses 400 distinct permutations, then 92 while loops needed and one can get the $8.2669$ for UF, which is the final answer for exact UF; (iii) for the AA’s, the AA-UF-3 yields the same UF for all numbers of directions considered, as long as the latter is no less than 120. The meaning of the latter number is explained above. This phenomena is no surprising and it is due to the design of the algorithm which only utilizes the normal vectors of all possible planes formed by three points, the latter is fixed number 120 in this case; (iv) the AA-UF-1 yields larger UF than that of AA-UF-2 in most of cases; (v) most important, the AA’s even after exhausting 10 millions directions still can not get a comparable size of UF that the exact algorithm produces just using 400 distinct permutations.

Notice that in this example, the number of total possible distinct permutations is $N(p, q) = 2 \ast (\binom{10}{0} + \binom{10}{1} + \binom{10}{2}) = 14282$. However, 400 distinct permutations are enough for the exact UF. The superiority of EA over AA’s is clearly demonstrated. In summary, AA’s can run much faster than the EA, but it is very difficult for them to get the results the EA provides (in fact, AA-UF-1 yields UF 6.8930 even after depleting 10 millions directions).

On the other hand, the EA is not feasible in practice for larger $n$ and $p$, I would not recommend it for $p > 3$. Recall, the possible distinct permutations in $p = 3$ and $n = 10$ case above is 14282. In table 2, we just used 400 of them to get the exact result of UF. One issue immediately arises. That is, in practice what should be the cut off number $N$ for the distinct permutations one should employ. One suggestion is $N = \min\{\binom{n}{p}, 500, 1000\}$ (obviously, there is an alternative automatic stopping rule based on the UF) and the while loop should be executed at most 1000 times.

However, there are at least two drawbacks in the comparison above, (i) the data set is still small, (ii) all results are random (since all use random sampling) (which means perhaps 200 permutations are enough for the exact UF if the EA runs many times. Similarly, AA using 1000 directions might get much better results if it runs many times). Each of results in table 2 is a single run result. Therefore the results and conclusions above have their limitations.

Example 2.4.3 Performance of exact algorithm versus approximate algorithms

In this example, we will consider the cases $p = 3, n = 50$ and $p = 4, n = 20$. Furthermore, we will run each of the algorithms 100 times (ideally it should be 1000 or even 10,000 times, but due to the time consumed by the EA-UFHD, they are not affordable options). $q$ for the two cases are 19600 and 4845 and $N(n, p)$ are 384140402 and 37887089270, respectively.

Furthermore, we try to set up a fair comparison base for the two types of methods. We will use the numbers of directions employed by them as the criterion for performance assessment. Note that for the EA, we count the number of directions it utilized as follows. In each while
We (i) compute the mean, minimum (min) and maximum (max) of 100 UF’s for each of the algorithms, and (ii) compute $D_{max} :=$ the difference in max UF (the max of UF of any algorithm subtracted by that of EA) and (iii) count the times that the UF of any algorithm is less than that of the EA in the 100 trials (denoted it by $N_{nega} :=$ the count of the negative $D_{uf}$’s, $D_{uf}$ is defined as the UF of any algorithm subtracted by that of the EA for each of 100 trials).

Results are listed in the table 3. Note that in $p = 3$ case, 400 and 800 directions used by the AA’s amounts to 40 and 80 while loops executed in the EA-UFHD (we convert them to the number of while loops in EA to terminate the algorithm). In the $p = 4$ case, 360 and 540 directions for the AA’s amounts to 20 and 30 while loops executed in the EA.

Examining the table reveals that (i) in all cases considered, the EA produces the largest UF among the 100 trials, this is demonstrated by the max or $D_{max}$ column in the table; (ii)
AA3 is the second best method among all four in the sense that (a) it yields the largest UF among the three AA methods in mean, minimum, and maximum of 100 UF’s in all cases (b) its mean and minimum of 100 UF’s are even larger than those of EA in all cases considered (except the most important maximum of 100 UF’s); (iii) the last columns tell the percentage that the EA yields a larger UF than that of other AA’s. For example, in the \( p = 4, n = 20, \) and \( N_v = 540 \) case, UF of EA is larger than that of AA2 in every trial.

**Performance of approximate algorithms w.r.t. efficiency and accuracy**

<table>
<thead>
<tr>
<th>( p )</th>
<th>methods</th>
<th>mean UF</th>
<th>standard deviation</th>
<th>time consumed</th>
<th>number of ( v ) used</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>AA-UF-1</td>
<td>0.3035</td>
<td>0.1053</td>
<td>26.4750</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-2</td>
<td>0.3041</td>
<td>0.1054</td>
<td>116.0503</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-3</td>
<td>0.3042</td>
<td>0.1057</td>
<td>34.7244</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td>5</td>
<td>AA-UF-1</td>
<td>0.4815</td>
<td>0.0994</td>
<td>31.1470</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-2</td>
<td>0.4447</td>
<td>0.0995</td>
<td>227.3421</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-3</td>
<td>0.5472</td>
<td>0.1095</td>
<td>80.3334</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td>10</td>
<td>AA-UF-1</td>
<td>0.5198</td>
<td>0.0928</td>
<td>33.4609</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-2</td>
<td>0.4467</td>
<td>0.1049</td>
<td>278.6988</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-3</td>
<td>0.7385</td>
<td>0.1156</td>
<td>100.4025</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td>20</td>
<td>AA-UF-1</td>
<td>0.5253</td>
<td>0.0877</td>
<td>40.5291</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-2</td>
<td>0.4152</td>
<td>0.1054</td>
<td>570.2156</td>
<td>( 10^3 )</td>
</tr>
<tr>
<td></td>
<td>AA-UF-3</td>
<td>1.1656</td>
<td>0.1626</td>
<td>236.5488</td>
<td>( 10^3 )</td>
</tr>
</tbody>
</table>

Table 4: Performance comparison of three approximate algorithms.

Above examples clearly demonstrate the advantage of the EA over the AA’s. But as said before, EA is not very feasible in practice due to its limitation on \( p \) and \( n \). For example, it is not capable of handling the case \( p = 20 \) and \( n = 100 \) and running 1000 times with while loop executed 20 times in each trial. But AA’s is more feasible and can easily handle the situation. Furthermore, the depth induced median has to be computed eventually by approximate methods. Following example is devoted to the comparison of the three AA’s.

**Example 2.4.4 Performance comparison between three approximate algorithms**

Here we generate \( m = 1,000 \) samples from the model: \( y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_{p-1} x_{ip-1} + e_i, \ i = 1, 2, \ldots, n, \) with sample sizes \( n = 100, \) where \( e_i \sim N(0, 1). \) In light of the regression equivariance of deepest projection depth estimator and the invariance of PRD, we can assume without loss of generality (w.l.o.g.) that \( \beta_0 = (\beta_0, \beta_1, \ldots, \beta_{p-1})' = (0, 0, \cdots, 0)' \). We generate \( Z_i = (x_{i1}, \cdots, x_{ip-1}, y_i) \) from \( p \)-dimensional standard normal distribution.

Three AA’s compute the unfitness of \( \beta_0 = (0, 0, \cdots, 0)' \) with results (mean of 1000 UF’s and deviations, **total time** consumed (in seconds) for 1000 samples, and unit directions used) are listed in Table 4 which features the attributes of three AA’s.
The table reveals that (i) AA-UF-1 is fastest and AA-UF-2 is slowest in all cases, confirming the theoretical time complexity results; (ii) AA-UF-3 is the most accurate in all cases (with the largest mean UF), AA-UF-2 is superior over AA-UF-1 only for the case $p = 2$ in terms of accuracy (mean is slightly larger); (iii) AA-UF-1 is most efficient (smallest s.d. and fastest), AA-UF-3 has the largest s.d., but this could be reversed by increasing the number of directions $v$ used to $(n_p)$; (vi) $p$ has the least (no greater than linear) effect on the time consumed by AA-UF-1 and it reduces the s.d. of AA-UF-1 when it increases; (v) overall, AA-UF-3 (or AA-UF-1) should be recommended.

3 Computation of the maximum PRD

3.1 Algorithms

The last section addresses the computation of the unfitness (UF), and thus equivalently that of the projection regression depth (PRD). In order to take the advantage of the regression depth notion to achieve better robustness, our ultimate goal is to seek the deepest regression lines (or hyperplanes). Now we focus on the computation of the maximum PRD ($T^*_\text{PRD}$).

The rough idea is as follows. Randomly select $N_\beta$ of $\beta$ over a very wide range in parameter space $\mathbb{R}^p$, calculate all $\text{UF}(\beta, F_Z)$ w.r.t. the sample distribution $F_Z$ of $F_Z$. Sort the latter and select $p + 1$ $\beta$'s with smallest unfitness. Over the simplex formed by these $p + 1$ $\beta$ points (in parameter space), search the point ($\beta$) with the smallest unfitness (equivalent the deepest regression line or hyperplane). Denote the latter by $T^*_n$, the sample version of $T^*_\text{PRD}$.

In the above process, we have implicitly taken advantage of the property of $\text{PRD}(\beta; F_Z)$ or $\text{UF}(\beta; F_Z)$. That is, $\text{PRD}(\beta; F_Z)$ satisfies the property (P3) of Z18a (monotonicity relative to the deepest point). Therefore the depth region of $\beta$ (the set of all $\beta$’s with depth no less than a fixed value) is convex and nested. Hence, the deepest point(s) must lie over the convex simplex formed by the $p + 1$ $\beta$ points. The deepest PRD point is unique (see Zuo (2019b)).

The following is an approximate algorithm for the computation of $T^*_n$.

(A) Randomly select a set of points $\beta_j \in \mathbb{R}^p$, $j = 1, \cdots, N_\beta$, where $N_\beta$ is a tuning parameter for the total number of random points.

(B) For each $\beta_j$, compute, over a set of randomly selected unit directions $v_k \in S^{p-1}$, $k = 1, \cdots, N_v$, an approximate unfitness of $\beta_j$ w.r.t. $\{Z_{ik}^j = (y_i - w_i'\beta_j)/(w_i'v_k)\}$, $i = 1, \cdots, n$, $k = 1, \cdots, N_v$, where $N_v$ is another tuning parameter.

(C) Select the deepest $p + 1$ $\beta_j$’s (points with smallest unfitness). Search over the closed convex hull formed by these $p + 1$ points via a common nonlinear optimization algorithm (e.g. the downhill simplex method (Nelder-Mead), or the MCMC technique) to get the final deepest $\beta$ or our approximate $T^*_n$.

(D) To mitigate the effect of randomness, repeat the steps above (many times) so that the one $T^*_n$ with the maximum updated regression depth is adopted.

Remarks 3.1:
(I) The candidate (random point) \( \mathbf{\beta} \) can be produced by randomly selecting \( p \) points from \( \mathbf{Z}^{(n)} = \{ (\mathbf{x}_i, y_i), i = 1, \cdots, n \} \) which determine a \( (\mathbf{\beta}) \) or hyperplane \( y = \mathbf{w}' \mathbf{\beta} \) containing all \( p \) points. Let \( S_{\mathbf{\beta}} := \{ \mathbf{\beta}_1, \cdots, \mathbf{\beta}_{N_{\mathbf{\beta}}} \} \) be all \( \mathbf{\beta}'s\).

(II) The random directions could be selected among those which are normal vector of the hyperplanes formed by \( p \) points from \( \mathbf{Z}^{(n)} \). Furthermore, for each \( \mathbf{\beta}_j \in S_{\mathbf{\beta}} \), one can consider all \( \mathbf{v}_i^j = (\mathbf{\beta}_i - \mathbf{\beta}_j)/\|\mathbf{\beta}_i - \mathbf{\beta}_j\|, \forall \mathbf{\beta}_i \neq \mathbf{\beta}_j\). Let \( S_{\mathbf{v}} := \{ \mathbf{v}_1, \cdots, \mathbf{v}_{N_{\mathbf{v}}} \} \) be all \( \mathbf{v}'s\).

(III) For a better approximation of depth (unfitness) of \( \mathbf{\beta}_j \), tune \( N_{\mathbf{v}} \). For a better approximation of \( \mathbf{T}_{n}^{*} \), tune \( N_{\mathbf{\beta}} \). Continue iterating until it satisfies a stopping rule (e.g. the difference between consecutive depths is less than a cutoff value).

(IV) The overall worst case time complexity of the algorithm is: step (A)+(B): \( O(N_{\mathbf{v}}N_{\mathbf{\beta}}pm) \), step (C): \( O(pN_{\mathbf{\beta}} + N_{\mathbf{v}}N_{\text{Iter}}pm) \), where \( N_{\text{Iter}} \) is the total number of iterations in the optimization algorithm; step(D) \( O(R(N_{\mathbf{v}}N_{\mathbf{\beta}}pm + pN_{\mathbf{\beta}} + N_{\mathbf{v}}N_{\text{Iter}}pm)) \), where \( R \) is the number of replications. The overall cost of the algorithm is \( O(Rp(N_{\mathbf{v}}N_{\mathbf{\beta}} + N_{\mathbf{\beta}} + N_{\mathbf{v}}N_{\text{Iter}}n)) \), which could be reduced to \( O(N_{\mathbf{v}}N_{\mathbf{\beta}}pm) \) by skipping steps (C) and (D).

(V) After obtaining the approximate UF of the first \( (p+1) \mathbf{\beta}_j \)'s, record UF\( _{\text{min}} \), the minimum of all \( (p + 1) \) UF's. For the calculation of UF for any future \( \mathbf{\beta}_k \), if along any direction \( \mathbf{v} \), the directional UF\( _{\mathbf{v}}(\mathbf{\beta}_k, F_{\mathbf{v}}^p) \geq UF_{\text{min}} \), then stop the computation for \( \mathbf{\beta}_k \) and move to \( \mathbf{\beta}_{k+1} \). Update UF\( _{\text{min}} \) if a new UF is obtained. In this way, the overall cost of the algorithm will be drastically reduced.

(VI) **Alternative algorithms.**

(i) After (A), compute the coordinate-wise median of the \( \mathbf{\beta}'s\) and use it as an initial point for a nonlinear optimization algorithm (e.g. optimx or DEoptim in R) along with other arguments (e.g. a function compute-UF) to find the \( \mathbf{T}_{n}^{*} \).

(ii) Increasing \( N_{\mathbf{v}} \) and \( N_{\mathbf{\beta}} \) and skipping steps (C) and (D), just employ steps (A)+(B).

(iii) Seeking an algorithm similar to the one MEDSWEEP in Van Aelst et al (2002) for \( \mathbf{T}_{RD}^{*} \) since when \( p = 1 \) and regression through the origin, \( \mathbf{T}_{PRD}^{*} \) recover the median of \( \{y_i/x_i\} \) (see Section 3.3 of RH99 for a related discussion).

\[ \blacksquare \]

### 3.2 Computation times

Under the demand of a referee, we now turn to the comparison of the computation times for \( \mathbf{T}_{PRD}^{*} \), \( \mathbf{T}_{RD}^{*} \) and the least trimmed squares (Rousseeuw (1984)) regression (ltsReg) estimators.

We generate 1000 samples \( \mathbf{Z}^{(n)} = \{ (\mathbf{x}_i^j, y_i^j), i = 1, \cdots, n, \mathbf{x}_i \in \mathbb{R}^{p-1} \} \) from the Gaussian distribution with zero mean vector and 1 to \( p \) as its diagonal entries of the diagonal covariance matrix for various \( n \) and \( p \). They are contaminated by 5% i.i.d. normal \( p \)-dimensional points with individual mean 10 and variance 0.1. Thus, we no longer have a symmetric errors and homoscedastic variance model (skewness and heteroscedasticity are allowed for RD of RH99).
Table entries (a, b) are: $a$: empirical mean squared error, $b$: average time per calculation (seconds).

<table>
<thead>
<tr>
<th>$n$</th>
<th>method</th>
<th>$p = 2$</th>
<th>$p = 3$</th>
<th>$p = 4$</th>
<th>$p = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T^*_P RD$</td>
<td>(0.427, 5.239)</td>
<td>(1.079, 7.960)</td>
<td>(1.958, 3.909)</td>
<td>(7.842, 5.093)</td>
</tr>
<tr>
<td></td>
<td>$T^*_RD$</td>
<td>(0.461, 0.018)</td>
<td>(0.966, 0.028)</td>
<td>(7.5e+06, 0.777)</td>
<td>(3.942, 8.760)</td>
</tr>
<tr>
<td></td>
<td>ltsReg</td>
<td>(0.616, 0.003)</td>
<td>(1.030, 0.011)</td>
<td>(1.762, 0.008)</td>
<td>(3.445, 0.011)</td>
</tr>
<tr>
<td>$n = 20$</td>
<td>$T^*_P RD$</td>
<td>(0.223, 6.889)</td>
<td>(0.535, 16.94)</td>
<td>(0.847, 12.46)</td>
<td>(2.155, 5.253)</td>
</tr>
<tr>
<td></td>
<td>$T^*_RD$</td>
<td>(0.239, 0.034)</td>
<td>(0.502, 0.121)</td>
<td>(1.1e+04, 7.052)</td>
<td>(1.623, 10.26)</td>
</tr>
<tr>
<td></td>
<td>ltsReg</td>
<td>(0.379, 0.007)</td>
<td>(0.578, 0.010)</td>
<td>(0.782, 0.012)</td>
<td>(1.307, 0.016)</td>
</tr>
<tr>
<td>$n = 40$</td>
<td>$T^*_P RD$</td>
<td>(0.147, 7.795)</td>
<td>(0.347, 24.93)</td>
<td>(0.593, 21.24)</td>
<td>(1.390, 9.993)</td>
</tr>
<tr>
<td></td>
<td>$T^*_RD$</td>
<td>(0.154, 0.039)</td>
<td>(0.350, 0.283)</td>
<td>(2.5e+07, 25.41)</td>
<td>(1.209, 11.10)</td>
</tr>
<tr>
<td></td>
<td>ltsReg</td>
<td>(0.308, 0.006)</td>
<td>(0.458, 0.012)</td>
<td>(0.595, 0.015)</td>
<td>(0.860, 0.020)</td>
</tr>
<tr>
<td>$n = 60$</td>
<td>$T^*_P RD$</td>
<td>(0.103, 21.50)</td>
<td>(0.223, 30.56)</td>
<td>(0.385, 31.21)</td>
<td>(0.898, 24.62)</td>
</tr>
<tr>
<td></td>
<td>$T^*_RD$</td>
<td>(0.109, 0.043)</td>
<td>(0.238, 0.870)</td>
<td>(2.6e+04, 132.53)</td>
<td>(0.995, 15.92)</td>
</tr>
<tr>
<td></td>
<td>ltsReg</td>
<td>(0.268, 0.009)</td>
<td>(0.389, 0.017)</td>
<td>(0.456, 0.019)</td>
<td>(0.595, 0.028)</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>$T^*_P RD$</td>
<td>(0.147, 7.795)</td>
<td>(0.347, 24.93)</td>
<td>(0.593, 21.24)</td>
<td>(1.390, 9.993)</td>
</tr>
<tr>
<td></td>
<td>$T^*_RD$</td>
<td>(0.154, 0.039)</td>
<td>(0.350, 0.283)</td>
<td>(2.5e+07, 25.41)</td>
<td>(1.209, 11.10)</td>
</tr>
<tr>
<td></td>
<td>ltsReg</td>
<td>(0.308, 0.006)</td>
<td>(0.458, 0.012)</td>
<td>(0.595, 0.015)</td>
<td>(0.860, 0.020)</td>
</tr>
</tbody>
</table>

Table 5: Computation times (seconds) of different regression methods for various $n$ and $p$.

For a general estimator $T$, if it is regression equivariant, then we can assume (w.l.o.g.) that the true parameter $\beta_0 = 0 \in \mathbb{R}^p$. We calculate $\text{EMSE} := \frac{1}{R} \sum_{i=1}^{R} ||T_i - \beta_0||^2$, the empirical mean squared error (EMSE) for $T$, where $R = 1000$, $\beta_0 = (0, \cdots, 0)' \in \mathbb{R}^p$, and $T_i$ is the realization of $T$ obtained from the $i$th sample with size $n$. The EMSE and the average computation time (in seconds) per sample by different estimators are listed in Table 5.

Inspecting the table reveals that the naive algorithm above for $T^*_P RD$ is, as expected, the slowest (with a few unexpected exceptions) whereas the one for ltsReg is the fastest. In terms of EMSE, $T^*_P RD$ is the smallest when $p = 2$ whereas $T^*_RD$ becomes the smallest one when $p = 3$ and $n \leq 40$, when $p > 4$ ltsReg takes its turn (with few exceptions). Furthermore, when $p = 4$, $T^*_RD$ has extremely large EMSE, which can be slightly improved by taking the average of all deepest lines/hyperplanes, but the resulting EMSE are still the largest.

Table 3 calls for a much faster algorithm for $T^*_P RD$, somewhat similar to the MEDSWEET in Van Aelst et al (2002) for $T^*_RD$. Note that, we have employed the R function rdepth in mrfdDepth package for $T^*_RD$ computation. We selected a deepest hyperplane among many (or all in the small $n$ and lower $p$ cases) hyperplanes that formed by $p$ points from the sample. This approach guarantees the resulting $T^*_RD$ possessing the maximum RD among all considered but it might loss the equivariance. Another approach is calling the function
Figure 5: Four regression lines for a dataset with a single outlier (Solid red for LS, dashed blue for $T^*_{RD}$, dotted black for $T^*_{PRD}$ and dotdash green for lstReg). Left: Original eleven countries, lines from $T^*_{RD}$, $T^*_{PRD}$ and lstReg are similar while the LS line is attracted by single country, the USA. Right: The outlier, USA, is removed from the original data, all four lines are very similar and catch the overall pattern.

rdepthmedian in package mrfDepth. But it does not always yield to a $T^*_{RD}$ which still has the maximum RD (see Van Aelst et al (2002) and Mizera and Volauf (2002)), albeit it boosts efficiency (or lowers EMSE) of $T^*_{RD}$. For example, if we use the rdepthmedian, then the $T^*_{RD}$ will be $(-1.1276501, 0.3010287)$ in panel (a) of Figure 1, but in this case there is a unique deepest line (0, 0) which was obtained (and plotted in (a) of Figure 1) by utilizing our approach above with the function rdepth.

Overall, Table 3 suggests that for high $p$ Rousseeuw’s lstReg should be recommended among the three estimators whereas $T^*_{PRD}$ is a promising alternative to $T^*_{RD}$ in regression.

All R codes ran on a desktop Intel(R)Core(TM) i7-2600 CPU @ 3.40GHz 3.40GHz.

3.3 Examples

Example 3.1 Deepest regression line $T^*_{PRD}$ for a real data set

Lung Cancer and Smoking Data set is composed of per capita consumption of cigarettes in eleven countries in 1930 and the death rates (number of deaths per million people) from lung cancer in 1950 (see Table 3-3 of Tufte (1974), source: Doll (1955)).

To find out the relationship between death rate and the cigarettes consumed, we first regress the data with deepest line $T^*_{PRD}$, for benchmark and comparison purpose, lines from $T^*_{RD}$ (another line induced from depth), LS (classical one) and lstReg are also given. In terms of (intercept, slope) form, they are $(65.7488570, 0.2291153)$, $(-14.1666667, 0.4166667)$,
Figure 6: Four regression lines for data with or without contamination (Solid red for LS, dashed blue for $T^*_{RD}$, dotted black for $T^*_{PRD}$, and dotdash green for ltsReg). Left: Original 100 normal points, lines from LS, $T^*_{RD}$, $T^*_{PRD}$ and ltsReg are similar and catch the overall linear pattern. Right: 34% contaminated data set, both LS and $T^*_{RD}$ “break down” while $T^*_{PRD}$ and ltsReg resist the contamination and still track the major pattern.

(-14.9401198, 0.4191617 ), and (13.553435, 0.357668) for LS, $T^*_{RD}$, $T^*_{PRD}$ and ltsReg, respectively. The two depth lines are almost identical whereas LS line is attracted by a single country USA downward, see the left panel of Figure 5.

Next we remove the single outlier and repeat the steps above, we have this time, (13.553435, 0.357668), (-14.6184633, 0.4182743), (-14.1666667, 0.4166667) and (13.553435 0.357668) for LS, $T^*_{RD}$, $T^*_{PRD}$ and ltsReg, respectively. ltsReg and LS are the same as previous ltsReg with USA included and $T^*_{RD}$ and $T^*_{PRD}$ are almost the same with $T^*_{PRD}$ is the exact previous $T^*_{RD}$ with USA included, see the right panel of Figure 5.

Overall, this example indicates that a single outlier can drastically affect the LS line and distinguish the LS line from other robust lines. But it can not differentiate the other three.

**Example 3.2 Performance of the deepest line of $T^*_{PRD}$ versus $T^*_{RD}$**

Here we first generate 100 points $Z_i = (x, y)'$ from the bivariate normal distribution $N(\mu, \Sigma)$, where

$$\mu = \begin{pmatrix} 8 \\ 0 \end{pmatrix}, \quad \Sigma = \begin{pmatrix} 9 & 0.9 \\ 0.9 & 1 \end{pmatrix}.$$  

Among the 100 points, we randomly select 34 points and replace them by other 34 points from another bivariate normal distribution $N(\mu_c, \Sigma_c)$ with

$$\mu_c = \begin{pmatrix} 1 \\ 11 \end{pmatrix}, \quad \Sigma_c = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}.$$  

25
Thus we have a 34% replacement-contamination data set.

First: w.r.t. the un-contaminated data set, we compute the deepest regression line induced from PRD and then the competitor line induced from RD of RH99. Again we also calculated the LS and ltsReg lines. The four lines in (intercept, slope) form are (-0.2533241, 0.0431503), (-0.25902968, 0.04446287), (-0.51728622, 0.08431267) and (-0.42734074, 0.06248309) for LS, \( T_{RD}^*, T_{PRD}^* \) and ltsReg, respectively. They are almost identical as shown in the left panel of Figure 6. All four seem to be useful, catching the overall linear pattern.

Second: w.r.t. the replacement-contaminated data set, we also compute the four lines. They are (9.643466, -1.085616), (11.407796, -1.487017), (0.71612780, -0.03124584) and (-0.42734074, 0.06248309) for LS, \( T_{RD}^*, T_{PRD}^* \) and ltsReg, respectively. They differ very much as shown in the right panel of Figure (6). Both LS and \( T_{RD}^* \) lines break down (attracted by the cloud of contamination) whereas \( T_{PRD}^* \) and ltsReg can resist the 34% contamination (in fact up to 50%) and catch the major pattern and continue to provide a useful regression line.

The computations in the example above (and below) are carried out with the R programming language for two reasons: (i) available codes (package: mrfDepth) for the RD of RH99 are in R and (ii) fair comparisons. R codes are available (see the link posted above).

Remarks 3.2: 

(I) Example 3.2 confirms the theoretical results in Z18b. That is, the deepest regression lines/hyperplanes induced from RPD is a robust alternative to the traditional LS lines/hyperplanes and has a higher asymptotical breakdown point (ABP) (50%) than the leading depth median (33%), the deepest regression estimator induced from RD of RH99. Note that with an appropriate trimming rate the least trimmed squares line possesses the best possible ABP as well whereas if the rate tends to 0% it leads to the LS lines/hyperplane having 0% ABP since just one outlier can ruin them.

(II) Robustness does not work in tandem with efficiency. So the key question is: Are the deepest projection regression lines/hyperplanes \( T_{PRD}^* \) efficient? In the following, ltsReg is excluded for a pure apple vs apple comparison (depth median vs depth median).

4 Efficiency of deepest projection regression lines/hyperplanes

Example 3.2 above confirms that \( T_{PRD}^* \) (or \( T_n^* \) in the empirical case) has a higher ABP than that of the leading regression depth induced median \( T_{RD}^* \). Robustness, however, is just one performance criterion for an estimator. Efficiency is another major performance measure. One naturally wonders whether \( T_{PRD}^* \) is inferior to \( T_{RD}^* \) w.r.t. the efficiency criterion.

In the following we investigate via simulation the finite-sample relative efficiency of the deepest lines \( T_{RD}^* \) and \( T_{PRD}^* \) w.r.t. the benchmark, the classical least squares line. The latter is optimal for the normal models by the Gauss-Markov theorem. We generate \( R = 1,000 \) samples from the simple linear regression model: 

\[
y_i = \beta_0 + \beta_1 x_i + e_i, i = 1, 2, \cdots, n,
\]

with different sample sizes \( n \) (see Tables 3 and 4), where \( e_i \sim N(0, \sigma^2) \).
Empirical mean squared error and relative efficiency of $T^*_{RD}$ and $T^*_{PRD}$ w.r.t. LS estimator

<table>
<thead>
<tr>
<th>n</th>
<th>measures</th>
<th>$T^*_{RD}$</th>
<th>$T^*_{PRD}$</th>
<th>LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>EMSE</td>
<td>0.5987264</td>
<td>0.3723071</td>
<td>0.2653862</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>44%</td>
<td>71%</td>
<td>100%</td>
</tr>
<tr>
<td>20</td>
<td>EMSE</td>
<td>0.2358544</td>
<td>0.1571197</td>
<td>0.1104146</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>47%</td>
<td>70%</td>
<td>100%</td>
</tr>
<tr>
<td>40</td>
<td>EMSE</td>
<td>0.10163933</td>
<td>0.07492950</td>
<td>0.05287073</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>52%</td>
<td>71%</td>
<td>100%</td>
</tr>
<tr>
<td>80</td>
<td>EMSE</td>
<td>0.04893200</td>
<td>0.04060671</td>
<td>0.02597673</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>53%</td>
<td>64%</td>
<td>100%</td>
</tr>
<tr>
<td>100</td>
<td>EMSE</td>
<td>0.03196556</td>
<td>0.02535978</td>
<td>0.01679633</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>53%</td>
<td>66%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6: Relative efficiency of $T^*_{RD}$ and $T^*_{PRD}$ for a normal model with 0% contamination.

<table>
<thead>
<tr>
<th>n</th>
<th>measures</th>
<th>$T^*_{RD}$</th>
<th>$T^*_{PRD}$</th>
<th>LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>EMSE</td>
<td>0.6612575</td>
<td>0.6181737</td>
<td>0.6373658</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>96%</td>
<td>103%</td>
<td>100%</td>
</tr>
<tr>
<td>20</td>
<td>EMSE</td>
<td>0.3396453</td>
<td>0.3247345</td>
<td>0.5179225</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>152%</td>
<td>159%</td>
<td>100%</td>
</tr>
<tr>
<td>40</td>
<td>EMSE</td>
<td>0.1613517</td>
<td>0.1525281</td>
<td>0.4475525</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>277%</td>
<td>293%</td>
<td>100%</td>
</tr>
<tr>
<td>80</td>
<td>EMSE</td>
<td>0.10348167</td>
<td>0.09775415</td>
<td>0.43277938</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>418%</td>
<td>442%</td>
<td>100%</td>
</tr>
<tr>
<td>100</td>
<td>EMSE</td>
<td>0.09702797</td>
<td>0.08947668</td>
<td>0.42298543</td>
</tr>
<tr>
<td></td>
<td>RE</td>
<td>436%</td>
<td>473%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 7: Relative efficiency of $T^*_{RD}$ and $T^*_{PRD}$ for a normal model with 10% contamination.

In light of the regression equivariance of the deepest regression estimators (see Z18a), we
can assume w.l.o.g. that true parameter $\beta_0 = (\beta_0, \beta_1)' = (0, 0)' = (0, 0)'.$ We generate $(x_i, y_i)'$ from an $\epsilon\%$ contaminated normal model $(1 - \epsilon)N((0, 0)', I_{2\times2}) + \epsilon\delta_{(0, 0)'}$ with $\epsilon = 0$ (a pure normal model, no contamination) and $\epsilon = 0.1$ (a 10% contaminated normal model), where $\delta_Z$ is a point mass contaminating distribution at point $Z \in \mathbb{R}^2$.

For a general estimator $T$, the relative efficiency (RE) of $T$ is obtained by dividing the EMSE of the LS estimator by that of $T$. Tables 6 (pure normal model case) and 7 (normal model with 10% contamination) demonstrate the results with various $n$’s.

Table 6 reveals that (i) $T_{\*PRD}$ is uniformly more efficient than $T_{\*RD}$ for all $n$; (ii) the limited numbers in Table 6 give a false impression that the efficiency of $T_{\*RD}$ increases forever as $n$ increases. This is not true since when $n = 200$ the efficiency of $T_{\*RD}$ is still just 52%; (iii) as expected, the EMSE’s of all lines decrease when $n$ increases.

On the other hand, at the 10% contaminated normal model, Table 7 shows that (i) when $n = 10$, there is just one point that is contaminated. The classical least squares line as well as the line $T_{\*RD}$ are drastically affected by just one contaminated point, nevertheless. They are less efficient than deepest projection regression depth line. It is surprising that the line produced by $T_{\*RD}$ is sensitive to just one point contamination and is even less efficient than the LS line; (ii) when $n$ increases, the efficiency of both deepest depth lines increases and are much higher than that of the LS line; (iii) $T_{\*PRD}$ is more efficient than $T_{\*RD}$ uniformly for all $n$; (iv) the EMSE’s of all lines decrease when $n$ increases; (v) the efficiency of the deepest lines increases as $n$ increases, for example, when $n = 200$, the efficiency of $T_{\*RD}$ will be 525%.

5 Concluding remarks

The maximum projection regression depth estimator is a robust alternative to the classical least squares estimator. It possesses the best asymptotic breakdown point, a bounded influence function, and a very high finite sample replacement breakdown point (see Zuo (2019a)).

This article addresses the computation issues of unfitness (UF), or equivalently the projection regression depth (PRD), and of the PRD induced regression median, the maximum projection depth estimator. Exact and approximate algorithms are proposed and investigated. Compared with the leading regression depth notion RD and its induced median $T_{\*RD}$ (RH99), $T_{\*PRD}$ is more computationally intensive ($O(Rp(NvN\beta + N\beta + NvN_{Iter}n))$ versus $O(p(n + N_{Iter}n + n \log n))$ of $T_{\*RD}$ (Van Aelst et al (2002)) or see Table 3). It, however, is not only more robust but also slightly more efficient (at least in $p = 2$ case).

The advantage (or disadvantage) of PRD comes from its definition, it is defined based on the magnitude of residuals whereas RD is defined purely on the sign of residuals. The latter results in a low breakdown point and efficiency, and non-uniqueness of $T_{\*RD}(F^n_Z)$ (the average might not have the maximum depth, see Van Aelst et al (2002) and Mizera and Volauf (2002)) whereas $T_{\*PRD}(F^n_Z)$ is unique (see Zuo (2019b)).

However, RD also gains some unique features. For example, among others, RD has the unique invariance property under the monotone transformations (see RH99); it is less difficult to compute and its definition does not require symmetry or homoscedasticity (see RH99).
Acknowledgments

The author thanks Hanshi Zuo and Professor Emeritus James Stapleton for their careful proofreading. He thanks Dr. Wei Shao for his proofreading and help in R coding. Special thanks go to two anonymous referees for their insightful comments and constructive suggestions, all of which have significantly improved the manuscript.

References


30