Computational Statistics and Data Analysis 56 (2012) 840–853

Contents lists available at SciVerse ScienceDirect

Computational Statistics and Data Analysis

journal homepage: www.elsevier.com/locate/csda

Computing multiple-output regression quantile regions

Davy Paindaveine [a,*], Miroslav Šiman [b]

[a] Université Libre de Bruxelles, Belgium

[b] Institute of Information Theory and Automation of the ASCR, Czech Republic

ARTICLE INFO

Article history:
Available online 24 November 2010

Keywords:
Halfspace depth
Multiple-output regression
Parametric linear programming
Quantile regression

ABSTRACT

A procedure relying on linear programming techniques is developed to compute (regression) quantile regions that have been defined recently. In the location case, this procedure allows for computing halfspace depth regions even beyond dimension two. The corresponding algorithm is described in detail, and illustrations are provided both for simulated and real data. The efficiency of a MATLAB implementation of the algorithm[1] is also investigated through extensive simulations.

1. Introduction

Due to the lack of a satisfactory concept of multivariate quantile, Koenker and Bassett (*Econometrica* 1978)'s celebrated theory of quantile regression has for long been restricted to *single-output* regression problems, which constitutes a severe limitation. Various works tried to extend quantile regression to the multiple-output context; see, e.g., Chaudhuri (1996), Koltchinskii (1997), Chakraborty (2003), Wei (2008), or Kong and Mizera (2008). Here, the focus is on the quantiles that were proposed in Hallin et al. (2010) — hereafter referred to as HPŠ10 — and that can be described as follows.

Consider a multiple-output regression problem where the m-variate response $\boldsymbol{Y}$ is to be regressed on the p-variate vector of regressors $\boldsymbol{X} = (1, \boldsymbol{W}')'$—so that $\{(\boldsymbol{w}', \boldsymbol{y}')' : \boldsymbol{w} \in \mathbb{R}^{p-1}, \boldsymbol{y} \in \mathbb{R}^m\} = \mathbb{R}^{p-1} \times \mathbb{R}^m$ is the natural space for considering fitted regression "objects". Assume that corresponding data points $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^p \times \mathbb{R}^m$, $i = 1, \ldots, n$, are given. For any $\tau \in (0, 1)$ and $\boldsymbol{u} \in \mathcal{S}^{m-1} := \{\boldsymbol{y} \in \mathbb{R}^m : \|\boldsymbol{y}\| = 1\}$, the sample HPŠ10 $(\tau\boldsymbol{u})$-quantile is defined as any element of the collection $\Pi_{\tau\boldsymbol{u}}^{(n)}$ of hyperplanes $\pi_{\tau\boldsymbol{u}}^{(n)} := \{(\boldsymbol{w}', \boldsymbol{y}')' \in \mathbb{R}^{p-1} \times \mathbb{R}^m : \widehat{\boldsymbol{b}}'_{\tau\boldsymbol{u}}\boldsymbol{y} - \widehat{\boldsymbol{a}}'_{\tau\boldsymbol{u}}(1, \boldsymbol{w}')' = 0\}$, with

$$(\widehat{\boldsymbol{a}}'_{\tau\boldsymbol{u}}, \widehat{\boldsymbol{b}}'_{\tau\boldsymbol{u}})' \in \arg\min \sum_{i=1}^{n} \rho_\tau(\widetilde{\boldsymbol{b}}'\boldsymbol{y}_i - \widetilde{\boldsymbol{a}}'\boldsymbol{x}_i) \quad \text{subject to } \boldsymbol{u}'\widetilde{\boldsymbol{b}} = 1, \tag{1}$$

where $\rho_\tau(x) = x(\tau - \mathrm{I}(x < 0))$ is the well-known τ-quantile check function. In other words, this regression $(\tau\boldsymbol{u})$-quantile simply is the traditional (single-output) Koenker and Bassett (1978) regression quantile of order τ obtained when considering, in $\mathbb{R}^{m+p-1}$, the oriented vectorial line bearing $(\boldsymbol{0}'_{p-1}, \boldsymbol{u}')'$ as the "vertical" axis (that is, as the axis of the univariate response).

Each optimal solution $(\widehat{\boldsymbol{a}}'_{\tau\boldsymbol{u}}, \widehat{\boldsymbol{b}}'_{\tau\boldsymbol{u}})'$ to (1) can be associated with the upper $(\tau\boldsymbol{u})$-quantile halfspace $H_{\tau\boldsymbol{u}}^{(n)+} = \{(\boldsymbol{w}', \boldsymbol{y}')' \in \mathbb{R}^{p-1} \times \mathbb{R}^m : \widehat{\boldsymbol{b}}'_{\tau\boldsymbol{u}}\boldsymbol{y} - \widehat{\boldsymbol{a}}'_{\tau\boldsymbol{u}}(1, \boldsymbol{w}')' \geq 0\}$. For $\tau \in (0, 1)$, HPŠ10 then defines (sample) τ-quantile regions as

$$R^{(n)}(\tau) := \bigcap_{\boldsymbol{u} \in \mathcal{S}^{m-1}} \bigcap \{H_{\tau\boldsymbol{u}}^{(n)+}\}, \tag{2}$$

* Corresponding address: E.C.A.R.E.S. and Département de Mathématique, Université Libre de Bruxelles, Av. F.D. Roosevelt, 50, CP114, B-1050 Brussels, Belgium. Tel.: +32 2 650 3845; fax: +32 2 650 4475.

E-mail addresses: dpaindav@ulb.ac.be (D. Paindaveine), siman@utia.cas.cz (M. Šiman).

[1] The code can be downloaded from http://homepages.ulb.ac.be/~dpaindav.

0167-9473/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.csda.2010.11.014

where $\bigcap\{H^{(n)+}_{\tau\boldsymbol{u}}\}$ stands for the intersection over all optimal solutions corresponding to the fixed τ and $\boldsymbol{u}$. In the location case $p = 1$, these regions were shown to coincide with the Tukey (1975) halfspace depth regions; see Theorem 4.2 of HPŠ10. In the general regression case $p > 1$, they form a family of nested polyhedral regions wrapping, up to the classical quantile crossings, a median or deepest regression hypertube. As shown in Section 7 of HPŠ10, these quantile regions allow for a much richer regression analysis than any traditional multiple-output regression method can provide.

Computing $R^{(n)}(\tau)$ (for some fixed τ), however, is a very challenging problem even when all the regression $(\tau\boldsymbol{u})$-quantiles are uniquely defined. In this case, each such regression quantile hyperplane (as any standard single-output Koenker and Bassett (1978) regression quantile hyperplane in $\mathbb{R}^{m+p-1}$) must contain $m+p-1$ data points almost surely if the underlying distribution of $(\boldsymbol{W}', \boldsymbol{Y}')'$ is absolutely continuous with respect to the Lebesgue measure. This implies that the collection of all such regression quantiles is finite, and that $R^{(n)}(\tau)$ could then be in principle computed *exactly*. Nevertheless, considering all $(m + p - 1)$-tuples of data points is unfeasible for practical datasets, so that computing the quantile regions remains a very difficult issue. Even in the location case where the problem reduces to computing halfspace depth regions, there is no exact *implementable* (non-trivial) solution beyond dimension two, at least to the best of the authors' knowledge.

The main objective of this paper is to provide a solution to this problem in the general regression case ($p \geq 1$), by showing how to compute efficiently, for any fixed $\tau \in (0, 1)$, the finite collection of *all* regression $(\tau\boldsymbol{u})$-quantiles. Since computing a single upper quantile halfspace $H^{(n)+}_{\tau\boldsymbol{u}}$ can be done in a straightforward way (by using single-output quantile regression techniques), the challenge here is to efficiently aggregate the information associated with the various fixed-τ directional quantile halfspaces.

The present work has much in common with parametric programming and sensitivity analysis and is particularly close to Shi and Lukas (2005) and Lukas and Shi (2006) that deal with sensitivity of constrained linear L_1 regression. Perturbations in rows and columns of the constraints matrix have been widely discussed in the general linear programming context as well; see Kon-Popovska (2003), and references therein. Interestingly, the very special form of the problem considered here and the row and column permutations employed lead to surprisingly simple and neat results, which makes it possible to solve the problem for all $\boldsymbol{u}$'s efficiently. This contribution therefore confirms the trend that applications of parametric programming in computational geometry still grow in number; see Raković et al. (2004) for another paper on this topic.

The outline of the paper is as follows. Section 2 derives a procedure that solves problem (1) with a given fixed $\tau \in (0, 1)$ for *all* $\boldsymbol{u} \in \mathcal{S}^{m-1}$ by means of parametric linear programming. In Section 3, the corresponding algorithm is described in detail (a MATLAB implementation of this algorithm can be downloaded from http://homepages.ulb.ac.be/~dpaindav). Section 4 provides some illustrations of quantile regions, both on simulated data (Section 4.1) and real data (Section 4.2). Extensive simulations are conducted in Section 5 to evaluate the efficiency of the MATLAB implementation of the algorithm. Finally, some technical matters related to the algorithm are discussed in Appendix.

2. Description of the procedure

This section describes a solution to the problem from the Introduction by means of parametric linear programming, and provides the theoretical background for the algorithm presented in Section 3. The structure of this section closely follows the accompanying MATLAB code and is split into three subsections. Section 2.1 rewrites the problem (1) as a linear program in a convenient way and shows that the assumption $\boldsymbol{u} \in \mathcal{S}^{m-1}$ can be relaxed without any harm into $\boldsymbol{u} \in \mathbb{R}^m$ (or more precisely, into $\mathbb{R}^m \setminus \{\boldsymbol{0}\}$). Section 2.2 then demonstrates that the resulting space $\mathbb{R}^m$ of the $\boldsymbol{u}$'s can be segmented into (a finite collection of) polyhedral cones, each corresponding to a *single* $(\tau\boldsymbol{u})$-quantile halfspace $H^{(n)+}_{\tau\boldsymbol{u}}$. Finally, Section 2.3 explains how to find all neighboring cones adjacent to a given one by means of simplex post-optimization, which paves the way for finding the whole conic segmentation and thus solves the problem completely.

2.1. Simplification of the linear problem

The following notation will be used throughout. The vectors $\boldsymbol{0}_\ell$ and $\boldsymbol{1}_\ell$ are defined as the ℓ-dimensional zero vector and the ℓ-dimensional vector of ones, respectively. The symbols $\mathbb{I}_{\ell\times\ell}$ and $\mathbb{O}_{r\times s}$ respectively refer to the ℓ-dimensional identity matrix and the zero $r \times s$ matrix. The positive and negative parts of an ℓ-vector $\boldsymbol{v} = (v_1, \ldots, v_\ell)'$ are defined as $\boldsymbol{v}_+ := (\max(v_1, 0), \ldots, \max(v_\ell, 0))'$ and $\boldsymbol{v}_- := (\max(-v_1, 0), \ldots, \max(-v_\ell, 0))'$, respectively, which yields $\boldsymbol{v} = \boldsymbol{v}_+ - \boldsymbol{v}_-$. The vector of residuals $r_i = r_i(\widetilde{\boldsymbol{a}}, \widetilde{\boldsymbol{b}}) := \widetilde{\boldsymbol{b}}'\boldsymbol{y}_i - \widetilde{\boldsymbol{a}}'\boldsymbol{x}_i$, $i = 1, \ldots, n$, will be denoted as $\boldsymbol{r} = (r_1, \ldots, r_n)'$. From the $n \times m$ (response) matrix

$$\mathbb{Y} := (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)' =: (\boldsymbol{y}^c_1, \ldots, \boldsymbol{y}^c_m)$$

and the $n \times p$ (design) matrix

$$\mathbb{X} := (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)' =: (\boldsymbol{x}^c_1, \ldots, \boldsymbol{x}^c_p),$$

one can construct

$$\mathbb{U}^y := (\boldsymbol{y}^c_1, -\boldsymbol{y}^c_1, \ldots, \boldsymbol{y}^c_m, -\boldsymbol{y}^c_m) \quad \text{and} \quad \mathbb{V}^x := (\boldsymbol{x}^c_1, -\boldsymbol{x}^c_1, \ldots, \boldsymbol{x}^c_p, -\boldsymbol{x}^c_p),$$

respectively. In the setup described in the Introduction, $\boldsymbol{x}_1^c = \mathbf{1}_n$. The general notation is used here because sometimes it may be interesting to work with another $\boldsymbol{x}_1^c$ (for example, when multiple identical observations occur in the sample, which may be relevant for resampling procedures) and because the algorithm presented in Section 3 does not require any special assumption on $\boldsymbol{x}_1^c$ at all. Finally, to make the connection between the text and the code as tight as possible, all vector inequalities are interpreted coordinate-wise and some basic MATLAB notation is used hereinafter, mainly for submatrices and subvectors with possibly permuted rows or columns.

With this notation, the optimization problem (1), for any $\boldsymbol{u} = (u_1, \ldots, u_m)' \in \mathcal{S}^{m-1}$, can be represented as the linear program

$$\min_{\boldsymbol{z}_P} \boldsymbol{c}_P' \boldsymbol{z}_P \quad \text{subject to } \mathbb{A}_P \boldsymbol{z}_P = \boldsymbol{b}_P,\ \boldsymbol{z}_P \geq \mathbf{0}, \tag{P}$$

with its dual twin brother

$$\max_{(\lambda, \boldsymbol{\mu}_P')'} \lambda \quad \text{subject to } \mathbb{A}_P'(\lambda, \boldsymbol{\mu}_P')' \leq \boldsymbol{c}_P, \tag{D}$$

where

$$\begin{aligned}
&\boldsymbol{z}_P = (\boldsymbol{a}'(\widetilde{\boldsymbol{a}}), \boldsymbol{b}'(\widetilde{\boldsymbol{b}}), \boldsymbol{r}_+', \boldsymbol{r}_-')' \in \mathbb{R}^{2p+2m+2n},\\
&\boldsymbol{a} = \boldsymbol{a}(\widetilde{\boldsymbol{a}}) = (\widetilde{a}_{1+}, \widetilde{a}_{1-}, \ldots, \widetilde{a}_{p+}, \widetilde{a}_{p-})' \in \mathbb{R}^{2p},\\
&\boldsymbol{b} = \boldsymbol{b}(\widetilde{\boldsymbol{b}}) = (\widetilde{b}_{1+}, \widetilde{b}_{1-}, \ldots, \widetilde{b}_{m+}, \widetilde{b}_{m-})' \in \mathbb{R}^{2m},\\
&\boldsymbol{c}_P = (\mathbf{0}_{2p+2m}', \tau \mathbf{1}_n', (1-\tau)\mathbf{1}_n')' \in \mathbb{R}^{2p+2m+2n},\\
&\boldsymbol{b}_P = (1, \mathbf{0}_n')' \in \mathbb{R}^{n+1},\\
&\mathbb{A}_P = \begin{pmatrix} \mathbb{A}^1_{P(1\times(2p+2m+2n))} \\ \mathbb{A}^2_{P(n\times(2p+2m+2n))} \end{pmatrix} = \begin{pmatrix} \mathbf{0}_{2p}' & \boldsymbol{\omega}_{2m}' & \mathbf{0}_n' & \mathbf{0}_n' \\ -\mathbb{V}^x_{n\times 2p} & \mathbb{U}^y_{n\times 2m} & -\mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} \end{pmatrix},\\
&\boldsymbol{\omega}_{2m} = (u_1, -u_1, \ldots, u_m, -u_m)' \in \mathbb{R}^{2m},
\end{aligned}$$

and $(\lambda, \boldsymbol{\mu}_P')'$ is the Lagrange multiplier vector corresponding to the equality constraints in (P).

Now, consider some $\boldsymbol{u}_0$ such that there exists a solution $(\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0}', \widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0}')'$ to (1) with only non-zero entries, and denote by $\widehat{\boldsymbol{z}}_P$ the corresponding optimal solution to (P). One can then define

- $\boldsymbol{I}_a$ (resp., $\widetilde{\boldsymbol{I}}_a$) as the vector containing indices of positive coordinates in $\boldsymbol{a}(\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0})$ (resp., $\boldsymbol{a}(-\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0})$), and $\widehat{\boldsymbol{I}}_a$ as the vector collecting the indices from $\{1, 2, \ldots, 2p\}$ contained neither in $\boldsymbol{I}_a$ nor in $\widetilde{\boldsymbol{I}}_a$. The vectors $\boldsymbol{I}_a$ and $\widetilde{\boldsymbol{I}}_a$ have a common dimension p', say, so that $\widehat{\boldsymbol{I}}_a$ has a dimension $2(p-p')$. For instance, if $\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0} = (2, 0, -4)'$, then one has $\boldsymbol{a}(\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0}) = (2, 0, 0, 0, 0, 4)$, $\boldsymbol{I}_a = (1, 6)'$, $\boldsymbol{a}(-\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0}) = (0, 2, 0, 0, 4, 0)$, $\widetilde{\boldsymbol{I}}_a = (2, 5)'$, and $\widehat{\boldsymbol{I}}_a = (3, 4)'$.
- $\boldsymbol{I}_b$, $\widetilde{\boldsymbol{I}}_b$ and $\widehat{\boldsymbol{I}}_b$ as the vectors obtained by adding $2p$ to each entry of the vectors obtained analogously from $\boldsymbol{b}(\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0})$ and $\boldsymbol{b}(-\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0})$. The vectors $\boldsymbol{I}_b$ and $\widetilde{\boldsymbol{I}}_b$ have a common dimension m', say, so that $\widehat{\boldsymbol{I}}_b$ has a dimension $2(m-m')$; with the same $\widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0}$ as above (yielding $p=3$) and $\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0} = (-1, 2, 0)'$, one obtains $\boldsymbol{I}_b = (2, 3)' + (6, 6)' = (8, 9)'$, $\widetilde{\boldsymbol{I}}_b = (1, 4)' + (6, 6)' = (7, 10)'$, and $\widehat{\boldsymbol{I}}_b = (5, 6)' + (6, 6)' = (11, 12)'$.
- $\boldsymbol{I}_Z$, $\boldsymbol{I}_e$ and $\widetilde{\boldsymbol{I}}_e$ as the vectors containing indices of observations with zero, positive, and negative residuals, respectively. Their dimensions – ζ, π, and ν, say (satisfying $\zeta + \pi + \nu = n$) – of course are the numbers of zero, positive and negative residuals, respectively. For any $\boldsymbol{u}_0$, an optimal solution $\widehat{\boldsymbol{z}}_P$ with $\zeta = p + m - 1$ can almost surely be found.

It is further assumed that $\boldsymbol{I}_a, \widetilde{\boldsymbol{I}}_a, \widehat{\boldsymbol{I}}_a, \boldsymbol{I}_b, \widetilde{\boldsymbol{I}}_b, \widehat{\boldsymbol{I}}_b, \boldsymbol{I}_Z, \boldsymbol{I}_e$ and $\widetilde{\boldsymbol{I}}_e$ are sorted in ascending order. Although an optimal solution with non-zero entries (i.e., with $m = m'$, $p = p'$ and empty vectors $\widehat{\boldsymbol{I}}_a$ and $\widehat{\boldsymbol{I}}_b$) can be found almost surely for any $\boldsymbol{u}_0$, the general case $m' < m$ and $p' < p$ may occur (but *always* with $\zeta = p' + m' - 1$, equivalently with $p' + m' + \pi + \nu = n + 1$) during the simplex post-optimization (see Section 2.3) leading from the optimal basis for $\boldsymbol{u}_0$ to different optimal bases for other vectors $\boldsymbol{u}$. This is the reason why the general case is considered here.

Below, only the case $\pi \neq 0$ and $\nu \neq 0$ is treated, but the other (simpler) cases can be handled analogously. Finally, put

$$\boldsymbol{I}_B = (\boldsymbol{I}_a', \boldsymbol{I}_b', 2(p+m)\mathbf{1}_\pi' + \boldsymbol{I}_e', (2p+2m+n)\mathbf{1}_\nu' + \widetilde{\boldsymbol{I}}_e')', \qquad \boldsymbol{I}_R = (\boldsymbol{I}_Z', \boldsymbol{I}_e', \widetilde{\boldsymbol{I}}_e')'$$

and

$$\boldsymbol{I}_C = (\boldsymbol{I}_B', \widetilde{\boldsymbol{I}}_a', \widetilde{\boldsymbol{I}}_b', \widehat{\boldsymbol{I}}_a', \widehat{\boldsymbol{I}}_b', 2(p+m)\mathbf{1}_\zeta' + \boldsymbol{I}_Z', (2p+2m+n)\mathbf{1}_\zeta' + \boldsymbol{I}_Z', (2p+2m+n)\mathbf{1}_\pi' + \boldsymbol{I}_e', 2(p+m)\mathbf{1}_\nu' + \widetilde{\boldsymbol{I}}_e')';$$

the vector $\boldsymbol{I}_B$ then consists of all the indices of basic variables. Therefore, it seems natural to permute the rows and columns of $\mathbb{A}_P$ according to $\boldsymbol{I}_R$ and $\boldsymbol{I}_C$ (in the spirit of Narula and Wellington, 2002), and to replace (P) with the strictly equivalent problem

$$\min_{\boldsymbol{z}_N} \boldsymbol{c}_N' \boldsymbol{z}_N \quad \text{subject to } \mathbb{A}_N \boldsymbol{z}_N = \boldsymbol{b}_N,\ \boldsymbol{z}_N \geq \mathbf{0}, \tag{N}$$

where

$$\boldsymbol{z}_N = \boldsymbol{z}_P(\boldsymbol{I}_C), \qquad \boldsymbol{c}_N = \boldsymbol{c}_P(\boldsymbol{I}_C), \qquad \boldsymbol{b}_N = \boldsymbol{b}_P$$

and

$$\mathbb{A}_N = \begin{pmatrix} \mathbb{A}^1_{N(1\times(2p+2m+2n))} \\ \mathbb{A}^2_{N(n\times(2p+2m+2n))} \end{pmatrix} = \begin{pmatrix} \mathbb{A}^1_P(\boldsymbol{I}_C) \\ \mathbb{A}^2_P(\boldsymbol{I}_R, \boldsymbol{I}_C) \end{pmatrix}$$

(the vector $\boldsymbol{b}_P$ remains untouched by this change since its last n components are equal). Alternatively,

$$\boldsymbol{z}_N = \mathbb{P}'_C \boldsymbol{z}_P, \qquad \boldsymbol{c}_N = \mathbb{P}'_C \boldsymbol{c}_P, \qquad \boldsymbol{b}_N = \begin{pmatrix} 1 & \mathbf{0}'_n \\ \mathbf{0}_n & \mathbb{P}_R \end{pmatrix} \boldsymbol{b}_P \quad \text{and} \quad \mathbb{A}_N = \begin{pmatrix} 1 & \mathbf{0}'_n \\ \mathbf{0}_n & \mathbb{P}_R \end{pmatrix} \mathbb{A}_P \mathbb{P}_C,$$

where $\mathbb{P}_R$ and $\mathbb{P}_C$ are the row and column permutation matrices (so that $\mathbb{P}'_R = \mathbb{P}_R^{-1}$ and $\mathbb{P}'_C = \mathbb{P}_C^{-1}$). One can easily check that

$$\begin{aligned} \boldsymbol{c}_N &= (\mathbf{0}'_{p'}, \mathbf{0}'_{m'}, \tau\mathbf{1}'_\pi, (1-\tau)\mathbf{1}'_\nu, \mathbf{0}'_{p'}, \mathbf{0}'_{m'}, \mathbf{0}'_{2(p-p')}, \mathbf{0}'_{2(m-m')}, \tau\mathbf{1}'_\zeta, (1-\tau)\mathbf{1}'_\zeta, (1-\tau)\mathbf{1}'_\pi, \tau\mathbf{1}'_\nu)' \\ &=: (\boldsymbol{c}'_0, \boldsymbol{c}'_1, \boldsymbol{c}'_2, \boldsymbol{c}'_3, \widetilde{\boldsymbol{c}}'_0, \widetilde{\boldsymbol{c}}'_1, \widetilde{\boldsymbol{c}}'_2, \widetilde{\boldsymbol{c}}'_3, \widetilde{\boldsymbol{c}}'_4, \widetilde{\boldsymbol{c}}'_5, \widetilde{\boldsymbol{c}}'_6, \widetilde{\boldsymbol{c}}'_7)' \\ &=: (\boldsymbol{c}'_{(n+1)\times 1}, \widetilde{\boldsymbol{c}}'_{(2p+2m+n-1)\times 1})' \end{aligned}$$

and that $\mathbb{A}_N$ is of the form $\mathbb{A}_N = \left(\mathbb{B}_{(n+1)\times(n+1)} \vdots \widetilde{\mathbb{B}}_{(n+1)\times(2p+2m+n-1)}\right)$, with

$$\mathbb{B} = \begin{pmatrix} \mathbf{0}'_{p'} & \boldsymbol{x}'_{m'} & \mathbf{0}'_\pi & \mathbf{0}'_\nu \\ \mathbb{E}^1_{\zeta\times p'} & \mathbb{F}^1_{\zeta\times m'} & \mathbb{O}_{\zeta\times\pi} & \mathbb{O}_{\zeta\times\nu} \\ \mathbb{E}^2_{\pi\times p'} & \mathbb{F}^2_{\pi\times m'} & -\mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ \mathbb{E}^3_{\nu\times p'} & \mathbb{F}^3_{\nu\times m'} & \mathbb{O}_{\nu\times\pi} & \mathbb{I}_{\nu\times\nu} \end{pmatrix}$$

and

$$\widetilde{\mathbb{B}} = \begin{pmatrix} \mathbf{0}'_{p'} & -\boldsymbol{x}'_{m'} & \mathbf{0}_{2(p-p')} & \bar{\boldsymbol{x}}_{2(m-m')} & \mathbf{0}'_\zeta & \mathbf{0}'_\zeta & \mathbf{0}'_\pi & \mathbf{0}'_\nu \\ -\mathbb{E}^1_{\zeta\times p'} & -\mathbb{F}^1_{\zeta\times m'} & -\bar{\mathbb{E}}^1_{\zeta\times 2(p-p')} & -\bar{\mathbb{F}}^1_{\zeta\times 2(m-m')} & -\mathbb{I}_{\zeta\times\zeta} & \mathbb{I}_{\zeta\times\zeta} & \mathbb{O}_{\zeta\times\pi} & \mathbb{O}_{\zeta\times\nu} \\ -\mathbb{E}^2_{\pi\times p'} & -\mathbb{F}^2_{\pi\times m'} & -\bar{\mathbb{E}}^2_{\pi\times 2(p-p')} & -\bar{\mathbb{F}}^2_{\pi\times 2(m-m')} & \mathbb{O}_{\pi\times\zeta} & \mathbb{O}_{\pi\times\zeta} & \mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ -\mathbb{E}^3_{\nu\times p'} & -\mathbb{F}^3_{\nu\times m'} & -\bar{\mathbb{E}}^3_{\nu\times 2(p-p')} & -\bar{\mathbb{F}}^3_{\nu\times 2(m-m')} & \mathbb{O}_{\nu\times\zeta} & \mathbb{O}_{\nu\times\zeta} & \mathbb{O}_{\nu\times\pi} & -\mathbb{I}_{\nu\times\nu} \end{pmatrix},$$

where $\boldsymbol{x}_{m'}$ and $\bar{\boldsymbol{x}}_{2(m-m')}$ are two disjoint subvectors of $\boldsymbol{\omega}_{2m}$ and $\mathbb{E}^i$, $\mathbb{F}^i$, $\bar{\mathbb{E}}^i$, and $\bar{\mathbb{F}}^i$, $i = 1, 2, 3$, are some known data-dependent matrices related to $\mathbb{U}^y$ or $\mathbb{V}^x$.

The columns of $\mathbb{B}$ correspond to the optimal basic variables of (N) so that $\widehat{\boldsymbol{z}}_N(n+2 : 2p+2m+2n)$ is zero and $\widehat{\boldsymbol{z}}_N(1 : n+1) = \mathbb{B}^{-1}\boldsymbol{b}_N = \mathbb{B}^{-1}(:, 1)$, where $\mathbb{B}^{-1}$ can be easily computed thanks to the special blockwise structure of $\mathbb{B}$:

$$\mathbb{B}^{-1} = \begin{pmatrix} \mathbb{C}_1^{-1} & \mathbb{O}_{(\zeta+1)\times\pi} & \mathbb{O}_{(\zeta+1)\times\nu} \\ \mathbb{C}_2\mathbb{C}_1^{-1} & -\mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ -\mathbb{C}_3\mathbb{C}_1^{-1} & \mathbb{O}_{\nu\times\pi} & \mathbb{I}_{\nu\times\nu} \end{pmatrix},$$

where

$$\mathbb{C}_1 = \begin{pmatrix} \mathbf{0}'_{p'} & \boldsymbol{x}'_{m'} \\ \mathbb{E}^1_{\zeta\times p'} & \mathbb{F}^1_{\zeta\times m'} \end{pmatrix}, \qquad \mathbb{C}_2 = \left(\mathbb{E}^2_{\pi\times p'} \vdots \mathbb{F}^2_{\pi\times m'}\right), \quad \text{and} \quad \mathbb{C}_3 = \left(\mathbb{E}^3_{\nu\times p'} \vdots \mathbb{F}^3_{\nu\times m'}\right).$$

Under the assumption that the data points $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^p \times \mathbb{R}^m$, $i = 1, \ldots, n$, deprived of their first coordinate, come from a continuous distribution over $\mathbb{R}^{m+p-1}$, all matrix inverses considered in the paper do exist — and, more generally, the proposed algorithm applies — with probability one.

Focus now on the standard case for which $m = m'$ and $p = p'$. There

$$\mathbb{A}_N = \begin{pmatrix} \mathbf{0}'_p & \boldsymbol{x}'_m & \mathbf{0}'_\pi & \mathbf{0}'_\nu & \mathbf{0}'_p & -\boldsymbol{x}'_m & \mathbf{0}'_\zeta & \mathbf{0}'_\zeta & \mathbf{0}'_\pi & \mathbf{0}'_\nu \\ \mathbb{E}^1_{\zeta\times p} & \mathbb{F}^1_{\zeta\times m} & \mathbb{O}_{\zeta\times\pi} & \mathbb{O}_{\zeta\times\nu} & -\mathbb{E}^1_{\zeta\times p} & -\mathbb{F}^1_{\zeta\times m} & -\mathbb{I}_{\zeta\times\zeta} & \mathbb{I}_{\zeta\times\zeta} & \mathbb{O}_{\zeta\times\pi} & \mathbb{O}_{\zeta\times\nu} \\ \mathbb{E}^2_{\pi\times p} & \mathbb{F}^2_{\pi\times m} & -\mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} & -\mathbb{E}^2_{\pi\times p} & -\mathbb{F}^2_{\pi\times m} & \mathbb{O}_{\pi\times\zeta} & \mathbb{O}_{\pi\times\zeta} & \mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ \mathbb{E}^3_{\nu\times p} & \mathbb{F}^3_{\nu\times m} & \mathbb{O}_{\nu\times\pi} & \mathbb{I}_{\nu\times\nu} & -\mathbb{E}^3_{\nu\times p} & -\mathbb{F}^3_{\nu\times m} & \mathbb{O}_{\nu\times\zeta} & \mathbb{O}_{\nu\times\zeta} & \mathbb{O}_{\nu\times\pi} & -\mathbb{I}_{\nu\times\nu} \end{pmatrix},$$

where $\zeta = p + m - 1$, and one can write

$$\mathbb{C}_1 = \begin{pmatrix} 0 & (\mathbf{0}'_{p-1}, \boldsymbol{x}'_m) \\ \mathbb{E}^1(:, 1) & \mathbb{D}_{\zeta\times\zeta} \end{pmatrix}.$$

Writing $\boldsymbol{x}$ for $\boldsymbol{x}_m$, the blockwise inversion of $\mathbb{C}_1$ leads to

$$\mathbb{C}_1^{-1} = \frac{1}{t(\boldsymbol{x})}\left(\mathbb{G}_0 + \sum_{i=1}^m x_i \mathbb{G}_i\right)$$

with $t(\boldsymbol{x}) = \boldsymbol{x}'\boldsymbol{s}(p:\zeta)$, $\boldsymbol{s} = -\mathbb{D}^{-1}\mathbb{E}^1(:,1)$,

$$\mathbb{G}_0 = \begin{pmatrix} 1 & \mathbf{0}'_\zeta \\ \boldsymbol{s} & \mathbb{O}_{\zeta\times\zeta} \end{pmatrix} \quad \text{and} \quad \mathbb{G}_i = \begin{pmatrix} 0 & -\mathbb{D}^{-1}(p-1+i,:) \\ \mathbf{0}_\zeta & \boldsymbol{s}(p-1+i)\mathbb{D}^{-1} - \boldsymbol{s}\mathbb{D}^{-1}(p-1+i,:) \end{pmatrix},$$

$i = 1,\ldots,m$; note that $\mathbb{G}_i(p+i,:) = \mathbf{0}'_{m+p}$, $i = 1,\ldots,m$. Here, it should be stressed that

$$\widehat{\boldsymbol{z}}_N(1:p+m) = \frac{1}{t(\boldsymbol{x})}\begin{pmatrix} 1 \\ \boldsymbol{s} \end{pmatrix}$$

depends on $\boldsymbol{x}$ (or $\boldsymbol{u}$) only through $t(\boldsymbol{x})$ and $\mathbb{B}$, which ensures that all non-zero $\boldsymbol{u}$'s associated with the same optimal basis $\mathbb{B}$ lead to a common upper halfspace $H^{(n)+}_{\tau\boldsymbol{u}}$.

2.2. Towards the cones associated with the optimal bases

Of course, the question is when $\mathbb{B} = \mathbb{B}(\boldsymbol{u})$ ceases to be optimal. According to the theory of linear programming, $\mathbb{B}$ is optimal if and only if $\boldsymbol{x}$ (or $\boldsymbol{u}$) satisfies both primal and dual feasibility conditions (PF) and (DF):

$$\boldsymbol{z} = \mathbb{B}^{-1}(:,1) = \frac{1}{t(\boldsymbol{x})}\begin{pmatrix} \mathbb{G}_0(:,1) \\ \mathbb{C}_2\mathbb{G}_0(:,1) \\ -\mathbb{C}_3\mathbb{G}_0(:,1) \end{pmatrix} \geq \mathbf{0}_{n+1}, \tag{PF}$$

$$\boldsymbol{d}' := \boldsymbol{c}'\mathbb{B}^{-1}\widetilde{\mathbb{B}} - \widetilde{\boldsymbol{c}}' \leq \mathbf{0}'_{2p+2m+n-1}. \tag{DF}$$

Fortunately, as is shown below, the $(2p+2m+2n)$ conditions in (PF) and (DF) may be reduced dramatically in the special context considered here.

First, (PF) is equivalent to the scalar inequality

$$t(\boldsymbol{x}) \geq 0$$

($t(\boldsymbol{x}) > 0$ almost surely) since (PF) must be satisfied at least for $\boldsymbol{u}_0$ by assumption, $\boldsymbol{z}(1) = 1/t(\boldsymbol{x})$ and $\boldsymbol{z}$ changes with $\boldsymbol{x}$ only through $t(\boldsymbol{x})$ (with the same constant matrix $\mathbb{B}$).

Then, focus on $\boldsymbol{d}$ and partition it according to $\widetilde{\boldsymbol{c}}$ into

$$\boldsymbol{d} = (\boldsymbol{d}'_0, \boldsymbol{d}'_1, \boldsymbol{d}'_2, \boldsymbol{d}'_3, \boldsymbol{d}'_4, \boldsymbol{d}'_5, \boldsymbol{d}'_6, \boldsymbol{d}'_7)'.$$

Simple algebra leads to $\boldsymbol{d}_0 = \mathbf{0}_{p'}$, $\boldsymbol{d}_1 = \mathbf{0}_{m'}$, $\boldsymbol{d}_6 = -\tau\mathbf{1}_\pi - (1-\tau)\mathbf{1}_\pi = -\mathbf{1}_\pi$, and $\boldsymbol{d}_7 = -(1-\tau)\mathbf{1}_\nu - \tau\mathbf{1}_\nu = -\mathbf{1}_\nu$, so that the corresponding inequalities in (DF) are always satisfied. If further $p = p'$ and $m = m'$, then moreover $\boldsymbol{d}_2 = \boldsymbol{d}_3 = \emptyset$, and (DF) thus becomes equivalent to

$$(\boldsymbol{d}'_4, \boldsymbol{d}'_5)' \leq \mathbf{0}_{2\zeta}.$$

This last set of 2ζ inequalities can be rewritten as

$$\mathbb{Q}_x\boldsymbol{x} \leq \mathbf{0}_{2\zeta}, \tag{3}$$

where

$$\mathbb{Q}_x = \begin{pmatrix} \boldsymbol{q}'_1 \\ \vdots \\ \boldsymbol{q}'_{2\zeta} \end{pmatrix} = \begin{pmatrix} -\mathbb{V}_x - \tau\mathbf{1}_\zeta\boldsymbol{s}(p:\zeta)' \\ \mathbb{V}_x - (1-\tau)\mathbf{1}_\zeta\boldsymbol{s}(p:\zeta)' \end{pmatrix} = \begin{pmatrix} -\mathbb{V}_{\text{mod}} \\ \mathbb{V}_{\text{mod}} - \mathbf{1}_\zeta\boldsymbol{s}(p:\zeta)' \end{pmatrix},$$

$$\mathbb{V}_{\text{mod}} = \mathbb{V}_x + \tau\mathbf{1}_\zeta\boldsymbol{s}(p:\zeta)', \quad \mathbb{V}_x = \begin{pmatrix} \boldsymbol{v}_1 \cdots \boldsymbol{v}_m \end{pmatrix},$$

$$\boldsymbol{v}_i = \mathbb{G}_i(:, 2:m+p)'\boldsymbol{h}, \quad i = 1,\ldots,m,$$

and

$$\boldsymbol{h} = (h_1,\ldots,h_{m+p})' = \tau\mathbb{C}'_2\mathbf{1}_\pi - (1-\tau)\mathbb{C}'_3\mathbf{1}_\nu.$$

Most importantly, (3) (equivalent to (DF)) entails

$$0 \leq \min_{i=1,\ldots,\zeta}\left\{\mathbb{V}_{\text{mod}}(i,:)\boldsymbol{x}\right\} \leq \max_{i=1,\ldots,\zeta}\left\{\mathbb{V}_{\text{mod}}(i,:)\boldsymbol{x}\right\} \leq \boldsymbol{s}(p:\zeta)'\boldsymbol{x} = t(\boldsymbol{x}), \tag{4}$$

hence implies (PF). Consequently, the whole set of $(2m+2p+2n)$ primal and dual feasibility conditions (PF) and (DF) is equivalent to (3).

Note that the vector $\boldsymbol{\mu}'_N := (\lambda, \boldsymbol{\mu}_\zeta^{r0\prime}, \boldsymbol{\mu}_\pi^{r+\prime}, \boldsymbol{\mu}_\nu^{r-\prime}) = \boldsymbol{c}'\mathbb{B}^{-1}$ hidden in (DF) solves the problem dual to (N) and contains the Lagrange multipliers corresponding to the equality constraints in (N). Clearly,

$$\lambda = \frac{1}{t(\boldsymbol{x})}\big(\tau \mathbf{1}'_\pi \mathbb{C}_2\mathbb{G}_0(:,1) - (1-\tau)\mathbf{1}'_\nu \mathbb{C}_3\mathbb{G}_0(:,1)\big)(=\boldsymbol{c}'_N\widehat{\boldsymbol{z}}_N = \boldsymbol{c}'_P\widehat{\boldsymbol{z}}_P),$$

$$\boldsymbol{\mu}^{r0} = \frac{1}{t(\boldsymbol{x})}\mathbb{V}_x\boldsymbol{x}, \qquad \boldsymbol{\mu}^{r+} = -\tau\mathbf{1}_\pi, \quad \text{and} \quad \boldsymbol{\mu}^{r-} = (1-\tau)\mathbf{1}_\nu,$$

which, in view of (3), implies $-\tau\mathbf{1}_\zeta \le \boldsymbol{\mu}^{r0} \le (1-\tau)\mathbf{1}_\zeta$.

The inequalities from (3), equivalent to the primal and dual feasibility conditions (PF) and (DF), can be rewritten by means of $\boldsymbol{u}$ as

$$\mathbb{Q}_u\boldsymbol{u} \le \mathbf{0}_{2\zeta}, \tag{5}$$

where $\mathbb{Q}_u$ is defined through $\mathbb{Q}_u\boldsymbol{u} \equiv \mathbb{Q}_x\boldsymbol{x}$ (actually, $\mathbb{Q}_u := \mathbb{Q}_x\mathrm{diag}(\mathrm{sign}(\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0}))$, where $\mathrm{diag}(\mathrm{sign}(\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0}))$ stands for the diagonal matrix whose entry (i,i) is the sign of $(\widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0})_i$). If the assumption $\boldsymbol{u} \in \mathcal{S}^{m-1}$ is removed, then all $\boldsymbol{u}$'s satisfying (5) form a polyhedral cone, say $\mathcal{C}_{\boldsymbol{u}_0}$. Such cones (corresponding to various $\boldsymbol{u}_0$'s) span the whole space $\mathbb{R}^m$ and the goal is to find them all, together with the corresponding optimal bases and upper halfspaces.

2.3. Finding the conic segmentation

Assume that all non-redundant constraints in (5) and facets of $\mathcal{C}_{\boldsymbol{u}_0}$ have been identified. Each such facet must be shared with another (adjacent) cone. That is why one may simply pass through all the cones $\mathcal{C}_{\boldsymbol{u}}$ counter-clockwise when $m = 2$. In general, it is possible to use the breadth-first search algorithm and always consider all such $\mathcal{C}_{\boldsymbol{u}}$'s that are adjacent to a cone treated in the previous step and that have not been considered yet.

It remains to clarify the process leading to the adjacent cone from a facet $\mathcal{F}$ of $\mathcal{C}_{\boldsymbol{u}_0}$. This facet corresponds to the j-th row of $\mathbb{Q}_u$, say, and has an interior point $\boldsymbol{u}_{\mathcal{F}}$ (defined, e.g., as the average of all non-zero vertices of $\mathcal{F} \cap [-1,1]^m$) that also identifies the facet $\mathcal{F}$ uniquely. This point is still certain to meet the primal feasibility conditions (PF) and the strategy therefore consists in using it as an input in the simplex post-optimization algorithm (that preserves primal feasibility and looks for dual feasibility) until the optimal basis of the adjacent cone is found.

This process can be described in more detail as follows. The $\boldsymbol{I}_C(n+1+p'+m'+j)$-th original variable will be the first to enter the basis. Then one should compute the auxiliary vector

$$\boldsymbol{\varrho} := \mathbb{B}^{-1}\widetilde{\mathbb{B}}(:, p'+m'+j),$$

find an index i satisfying

$$\frac{z_i}{\varrho_i} = \min\left\{\frac{z_h}{\varrho_h} : \varrho_h > 0, h = 1,\ldots,n+1\right\}, \tag{6}$$

and displace the $\boldsymbol{I}_C(i)$-th original basic variable to get a new primal feasible basis, say $\mathbb{B}_1$ (it may be noted that $\widetilde{\mathbb{B}}(:, m'+p'+j)$ contains only one non-zero coordinate if $j > 2(p-p') + 2(m-m')$). The basis $\mathbb{B}_1$ is optimal if and only if

$$\boldsymbol{d}_{2345} = (\boldsymbol{d}'_2, \boldsymbol{d}'_3, \boldsymbol{d}'_4, \boldsymbol{d}'_5)' \le \mathbf{0}_{2(p+m-1)}, \tag{7}$$

where $\boldsymbol{d}' = \boldsymbol{d}'_{\mathbb{B}_1} = \boldsymbol{c}'_{\mathbb{B}_1}\mathbb{B}_1^{-1}\widetilde{\mathbb{B}}_1 - \widetilde{\boldsymbol{c}}'_{\mathbb{B}_1}$. Although the blockwise structure of $\mathbb{B}_1^{-1}$ can be employed again, $\mathbb{C}_1^{-1}$ should be computed directly this time, with $\boldsymbol{x}$ corresponding to $\boldsymbol{u}_{\mathcal{F}}$.

If $\mathbb{B}_1$ fails this optimality test or $\zeta \ne p+m-1$, then one has to find an index j such that $\boldsymbol{d}_{2345}(j) \ge 0$ and repeat the previous steps until the optimal basis of the adjacent cone with $\zeta = p+m-1$ is found. Of course, the choice of j must not lead to the situation for which the new original variable to enter is the same as the one just removed.

3. Algorithm

The procedure described in the previous section leads quite straightforwardly to the algorithm presented here. To sum up, the basic form of the algorithm can always be performed in the following steps, where $\to$ indicates the flow of computation and the highlighted text refers to the topical sections of the Appendix that discuss some issues in more detail.

1. Adjust the data and τ if necessary; see *Input Data* and *Choice of* τ, respectively.
2. For a given directional vector $\boldsymbol{u}_0$, consider (P) and find its optimal solution $\widehat{\boldsymbol{z}}_P$ and optimal basis $\mathbb{B} = \mathbb{B}(\boldsymbol{u}_0)$; see *Computing the first directional quantile*. ($\widehat{\boldsymbol{z}}_P \to \widehat{\boldsymbol{a}}_{\tau\boldsymbol{u}_0}, \widehat{\boldsymbol{b}}_{\tau\boldsymbol{u}_0}, \boldsymbol{r}_+, \boldsymbol{r}_- \to \boldsymbol{I}_a, \widetilde{\boldsymbol{I}}_a, \widehat{\boldsymbol{I}}_a, \boldsymbol{I}_b, \widetilde{\boldsymbol{I}}_b, \widehat{\boldsymbol{I}}_b, \boldsymbol{I}_Z, \boldsymbol{I}_e, \widetilde{\boldsymbol{I}}_e \to \boldsymbol{I}_B, \boldsymbol{I}_R, \boldsymbol{I}_C \to \mathbb{A}_N, \boldsymbol{c}_N \to \mathbb{B}, \widetilde{\mathbb{B}}$).

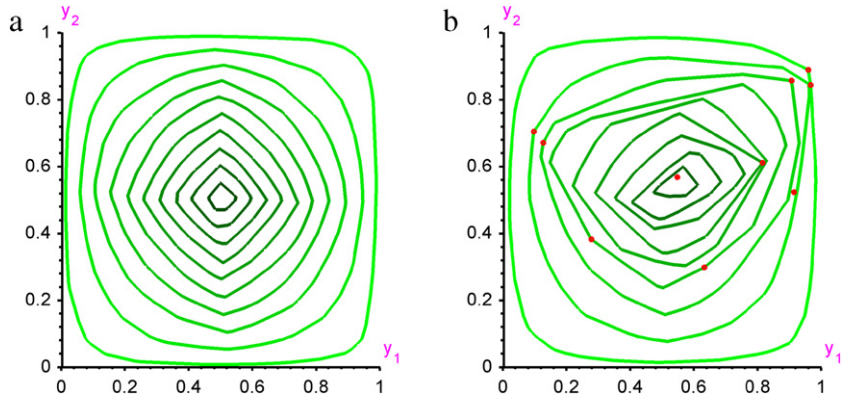

Fig. 1. In subfigure (a), quantile contours of order $\tau = 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40$, and 0.45 are plotted from a sample of $n = 2499$ observations drawn independently from the uniform distribution over $[0, 1]^2$. Subfigure (b) reports the corresponding contours after the weights of the first ten data points (that are plotted in the figure) were changed from 1 to 2499/20.

3. Set $\mathcal{B}_{\text{new}} := \{\mathbb{B}(\boldsymbol{u}_0)\}$ and $\mathcal{T} := \{\emptyset\}$.
4. Set $\mathcal{B}_{\text{old}} := \mathcal{B}_{\text{new}}$, then $\mathcal{B}_{\text{new}} := \emptyset$.
5. For each $\mathbb{B} = \mathbb{B}(\boldsymbol{u})$ in $\mathcal{B}_{\text{old}}$,
   (a) compute $\mathbb{Q}_u$ that determines the inequalities (5) defining the cone $\mathcal{C}_{\boldsymbol{u}}$ of all directions leading to the same quantile hyperplane as $\boldsymbol{u}$. ($\mathbb{B} \to \mathbb{C}_1, \mathbb{C}_2, \mathbb{C}_3 \to \mathbb{D}, \boldsymbol{s}, \boldsymbol{h} \to \mathbb{V}_x \to \mathbb{Q}_x \to \mathbb{Q}_u$)
   (b) find all facets and vertices of the polytope $\mathcal{P}_{\boldsymbol{u}} := \mathcal{C}_{\boldsymbol{u}} \cap [-1, 1]^m$, drop the facets not belonging to $\mathcal{C}_{\boldsymbol{u}}$ and compute $\boldsymbol{u}_{\mathcal{F}}$ for each remaining facet; see *Finding non-redundant constraints, facets and interior points*
   (c) for each such $\boldsymbol{u}_{\mathcal{F}}$, check whether $\boldsymbol{u}_{\mathcal{F}}$ belongs to $\mathcal{T}$ or not. If it does (equivalently, if the corresponding facet has already been considered), then do nothing. If it does not, then find $\mathbb{B}_{\text{new}}(\boldsymbol{u}_{\mathcal{F}})$ from $\mathbb{B}$ by means of the simplex post-optimization described at the end of Section 2.3 and add $\boldsymbol{u}_{\mathcal{F}}$ to $\mathcal{T}$ and $\mathbb{B}_{\text{new}}$ to $\mathcal{B}_{\text{new}}$; see *Realization of the breadth-first search algorithm*.
6. If $\mathcal{B}_{\text{new}}$ is non-empty, go back to Step 4. Otherwise, the algorithm terminates successfully (all cones $\mathcal{C}$ have been found and there is no new cone facet to investigate).

This algorithm can be implemented with computational complexity at worst $O_i + O(n\Sigma_n)$, where O_i denotes the computational complexity of solving the linear programming problem (P) from scratch in Step 2 and Σ_n stands for the total number of different quantile hyperplanes for a given τ. Both O_i and Σ_n, however, depend on the specific data configuration. On average, O_i can be made quite low by choosing a suitable solver for (P); see Section 6.4.4 in Koenker (2005). As for Σ_n, it can be as low as $O(1)$ and is never higher than $O(n^m)$. In most cases, it seems reasonable to assume it to be $O(n^{m-1})$ on average for a random $\tau \in (0, 0.5)$. This would be compatible with the empirical results of Section 5 that also indicate that the average computational complexity is not worse than $O(n^m)$.

4. Illustrations

This section presents some illustrative examples of quantile regions obtained from a MATLAB implementation of the algorithm described above. What is plotted for each quantile region is its boundary, called the *quantile contour*. Both simulated and real data are considered.

4.1. Simulated data

Bivariate location case. Starting with the bivariate location case ($m = 2$ and $p = 1$), data points $\boldsymbol{y}_i$, $i = 1, \ldots, n = 2499$, were generated independently from the uniform distribution over the unit square $[0, 1]^2$. Fig. 1(a) plots the resulting quantile contours for $\tau = 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40$, and 0.45. These contours match very well their population counterparts, namely the population halfspace depth contours; see Rousseeuw and Ruts (1999). The code can also deal with weighted observations (which in particular allows for multiple observations): if weights $\omega_i > 0$, $i = 1, \ldots, n$ (summing up to one or not) are given, the resulting "weighted" optimization problem is obtained by substituting $\boldsymbol{y}_{\omega i} := \omega_i \boldsymbol{y}_i$ and $\boldsymbol{x}_{\omega i} := \omega_i \boldsymbol{x}_i$, $i = 1, \ldots, n$, for the $\boldsymbol{y}_i$'s and $\boldsymbol{x}_i$'s in (1). Fig. 1(b) reports, for the same τ's as in Fig. 1(a), the quantile contours associated with weighted data points $\boldsymbol{y}_{\omega i} := \omega_i \boldsymbol{y}_i$, $i = 1, \ldots, n$, where the weights are given by

$$\omega_i = \begin{cases} \dfrac{2499}{20} & \text{for } i = 1, \ldots, 10 \\ 1 & \text{for } i = 11, \ldots, n = 2499, \end{cases}$$

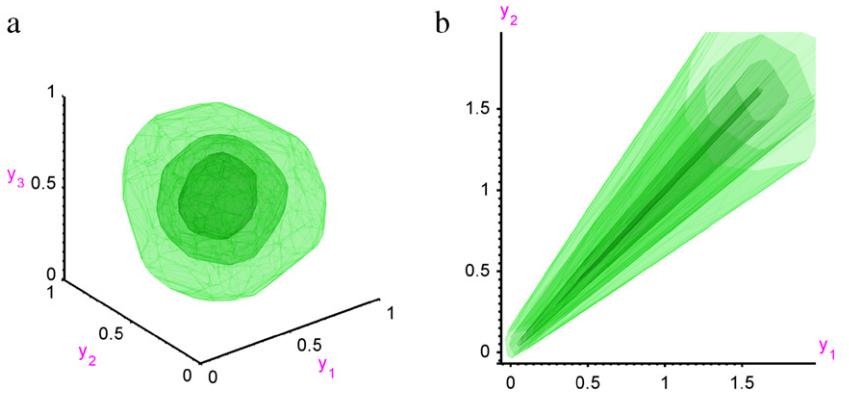

Fig. 2. In subfigure (a), quantile contours of order $\tau = 0.05, 0.15$, and 0.25 are plotted from a sample of $n = 249$ observations drawn independently from the uniform distribution over $[0, 1]^3$. Subfigure (b) reports the quantile regions of order $\tau = 0.05, 0.15, 0.30$, and 0.45 for $n = 249$ observations drawn independently from the regression model described in Section 4.1.

and the original data points $\boldsymbol{y}_i$ are the same as in Fig. 1(a). The ten points that can be seen in Fig. 1(b) are the original data points $\boldsymbol{y}_i$, $i = 1, \ldots, 10$, that receive the larger weight.

Trivariate location case. Fig. 2(a) illustrates the trivariate location case with $m = 3$ and $p = 1$. The sample considered there consists of $n = 249$ data points obtained independently from the uniform distribution over the unit cube $[0, 1]^3$. The figure reports the resulting quantile contours — that is, the sample halfspace depth contours — for $\tau = 0.05, 0.15$, and 0.25. This paper therefore brings a practical solution to the notoriously difficult problem of computing halfspace depth *regions* beyond dimension two.

Regression setup with two responses and one random covariate. The third setup considered is the simple heteroscedastic regression model

$$\boldsymbol{Y} = (W, W)' + \sqrt{W}\boldsymbol{\varepsilon},$$

where the random covariate W is uniformly distributed over $[0, 1]$ and the random vector $\boldsymbol{\varepsilon}$ (which is independent of W) is uniformly distributed over the unit square $[0, 1]^2$. Here, $n = 249$ data points $(\boldsymbol{x}_i', \boldsymbol{y}_i')' = (1, w_i, \boldsymbol{y}_i')' \in \mathbb{R}^p \times \mathbb{R}^m = \mathbb{R}^2 \times \mathbb{R}^2$, $i = 1, \ldots, n$, were obtained independently. Fig. 2(b) displays the resulting (trivariate, since they are objects of the $(w, \boldsymbol{y})$-space) regression quantile contours for $\tau = 0.05, 0.15, 0.30$, and 0.45. Of course, such regression contours are often hard to plot and to interpret, so that it is usually better to consider (a finite collection of) cuts obtained as the intersections of the regression contours under study with hyperplanes of the form $w = w_0$, where w_0 is some fixed value of the random covariate; see Section 7 of HPŠ10 for an illustration. A similar strategy is also adopted here for the real data considered in Section 4.2.

4.2. Real data

A real dataset of Rouncefield (1995) is now considered. The dataset contains some development and demographic characteristics for different countries, and it may be interesting to study the dependence of both male life expectancy at birth (Y_1) and death rate (Y_2) on the various covariates available. Actually, the goal here is not to perform a thorough regression analysis for the bivariate response $(Y_1, Y_2)'$ involving the complete collection of covariates, but rather to show in a simple model how quantile regression contours might look like in practice. Therefore, only an exploratory analysis of the dependence of $(Y_1, Y_2)'$ on GNP per capita (Z) is performed, with regressors $X_1 = 1, X_2 = \log Z$ and $X_3 = (\log Z)^2$ (which yields $m = 2$ and $p = 3$).

Various regression quantile contours (for the 91 countries whose records do not contain any missing value) were computed. These contours are objects in $\mathbb{R}^4$, hence cannot be plotted. However, parallel to the artificial regression illustration in Section 4.1, cuts of these contours associated with various fixed values of the covariate Z can be considered here. In the present setup, cuts are not obtained by intersecting the quantile contours with some hyperplanes, but rather with some vectorial spaces of dimension two in $\mathbb{R}^4$; fixing the value of Z to some z_0, say, indeed fixes the value of X_2 and X_3. The resulting cuts live in $\mathbb{R}^2$ and can be plotted easily.

Fig. 3(a) reports, in a single two-dimensional picture, the 300 cuts of the $(\tau = 0.10)$-quantile regression contour that are associated with $z_0 = 100, 200, 300, \ldots, 29\,900$, and $30\,000$. Fig. 3(b)–(f) show the corresponding cuts computed from the quantile regions with order $\tau = 0.15, 0.20, 0.25, 0.30$, and 0.35, respectively. Clearly, these cuts provide interesting information about the trend (for high values of τ) and about the shape (for low values of τ).

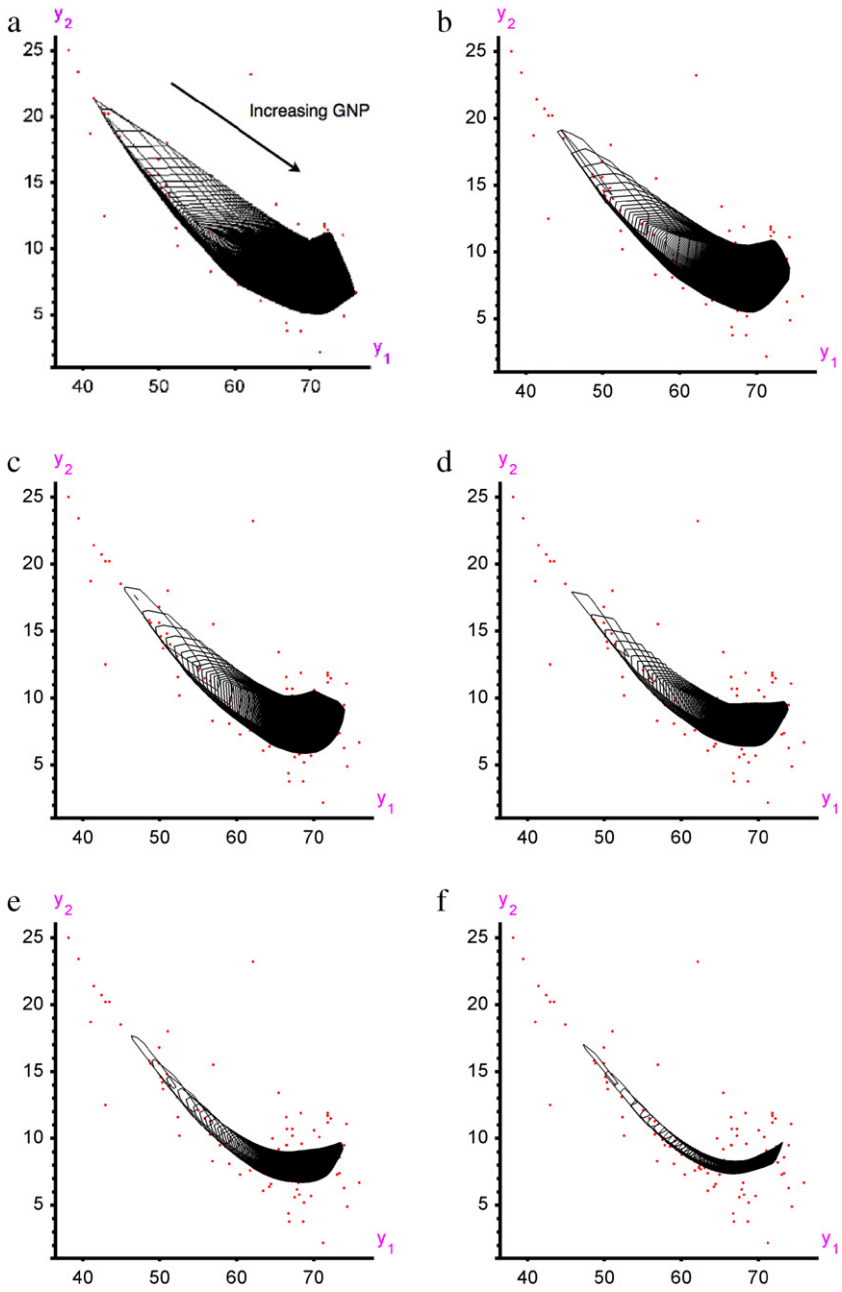

Fig. 3. Subfigure (a) reports, for the real dataset considered in Section 4.2, 300 two-dimensional cuts (each associated with one fixed value of GNP per capita) of the (four-dimensional) quantile regions of order $\tau = 0.10$. Subfigures (b)–(f) show the corresponding plots for $\tau = 0.15$, 0.20, 0.25, 0.30, and 0.35, respectively; see Section 4.2 for details.

5. Simulations

This section presents empirical results that quantify the speed (and show the possibilities) of the MATLAB implementation of the algorithm proposed in this paper. An Apple computer with Intel Core Duo 1.83 GHz, 512 MB RAM only, WIN XP SP2 and MATLAB 7.3.0.267 was used. Of course, other hardware or initial settings (see the Appendix) may lead to different results.

5.1. Speed comparison

In the location case ($p = 1$), the quantile regions provided coincide with the halfspace depth contours. As already mentioned, there is no exact implementable algorithm that could be used as a competitor to the proposed MATLAB code

Table 1
(2D location settings: $m = 2$ and $p = 1$.) Average execution time (in seconds) of our code is provided for given scenario S, number of observations n, and order τ in the bivariate location context. The numbers in parentheses indicate how many times it is faster than the benchmark.

S	$n \setminus \tau$	Absolute and relative execution times											
		0.010		0.025		0.050		0.100		0.200		0.400	
1:	50	0.08	(1.6)	0.09	(2.2)	0.10	(2.8)	0.12	(3.7)	0.14	(5.2)	0.17	(5.6)
2:	50	0.08	(2.1)	0.10	(2.4)	0.11	(2.9)	0.13	(3.8)	0.15	(4.7)	0.16	(5.5)
1:	100	0.09	(3.8)	0.10	(4.2)	0.13	(5.8)	0.16	(6.9)	0.22	(8.3)	0.27	(9.6)
2:	100	0.10	(3.8)	0.12	(4.5)	0.15	(5.5)	0.18	(7.2)	0.23	(8.4)	0.26	(9.5)
1:	150	0.10	(4.2)	0.13	(5.5)	0.16	(7.3)	0.23	(8.6)	0.31	(10.8)	0.38	(12.1)
2:	150	0.11	(4.5)	0.15	(5.7)	0.19	(7.1)	0.24	(9.1)	0.32	(10.5)	0.37	(11.8)
1:	200	0.12	(5.5)	0.16	(6.9)	0.20	(8.7)	0.28	(10.4)	0.39	(12.2)	0.51	(13.3)
2:	200	0.15	(5.5)	0.19	(6.9)	0.24	(8.5)	0.31	(10.3)	0.40	(12.1)	0.49	(13.1)
1:	300	0.18	(5.9)	0.23	(7.4)	0.32	(9.3)	0.42	(12.1)	0.60	(14.0)	0.78	(15.6)
2:	300	0.21	(5.9)	0.29	(6.9)	0.36	(9.3)	0.48	(11.7)	0.61	(13.6)	0.75	(15.1)
1:	500	0.23	(7.7)	0.33	(9.5)	0.47	(11.6)	0.68	(14.0)	1.00	(15.9)	1.33	(16.9)
2:	500	0.31	(7.0)	0.42	(8.9)	0.56	(11.4)	0.77	(13.6)	1.03	(15.7)	1.29	(16.8)
1:	1 000	0.51	(8.0)	0.78	(9.9)	1.24	(10.9)	1.95	(12.1)	3.13	(12.5)	4.07	(14.0)
2:	1 000	0.68	(7.4)	1.08	(9.1)	1.53	(10.4)	2.15	(12.2)	2.72	(14.9)	3.41	(15.6)
1:	2 000	1.06	(8.8)	1.85	(10.2)	3.03	(11.0)	5.14	(11.4)	7.85	(12.3)	10.82	(12.8)
2:	2 000	1.57	(7.6)	2.49	(9.2)	3.67	(10.4)	5.75	(11.2)	8.05	(12.3)	10.32	(12.7)
1:	5 000	3.22	(9.1)	6.33	(9.7)	10.36	(10.5)	17.39	(11.1)	26.82	(12.0)	39.19	(11.7)
2:	5 000	5.31	(7.2)	7.92	(9.6)	13.12	(9.8)	19.85	(10.8)	28.93	(11.4)	37.97	(11.6)
1:	10 000	9.99	(7.2)	19.33	(7.9)	33.45	(8.2)	56.07	(8.7)	90.05	(9.1)	121.30	(9.7)
2:	10 000	14.93	(6.4)	25.42	(7.7)	40.35	(8.2)	64.04	(8.5)	92.03	(9.1)	115.09	(9.6)
1:	20 000	34.62	(7.2)	71.35	(7.3)	126.92	(7.5)	205.54	(8.1)	316.75	(8.7)	432.15	(9.0)
2:	20 000	51.21	(6.3)	91.65	(7.1)	151.56	(7.0)	229.96	(7.9)	327.60	(8.5)	411.33	(8.8)

for $m > 2$. For the bivariate case ($m = 2$), however, this MATLAB code can be compared to that coauthored and kindly provided to the authors by Ivan Mizera, chosen as a benchmark here.

In order to do so, n i.i.d. bivariate observations were generated (i) from the bivariate standard normal distribution $N(0, 1)^2$ ($S = 1$) and (ii) from the centered bivariate uniform distribution over the unit square $U([-0.5, 0.5])^2$ ($S = 2$). For any combination of $\tau = \{0.010, 0.025, 0.050, 0.100, 0.200, 0.400\}$ and $n \in \{50, 100, 150, 200, 300, 500, 1000, 2000, 5000, 10\,000, 20\,000\}$, the computation was run ten times for each scenario—actually, with the following changes to the default settings of the MATLAB code: CTechST.InCheckI = 0, CTechST.ReportI = 0, CTechST.TestModeI = 0, and CTechST.OutSaveI = 0 (this suppresses checking the input for correctness, detailed output on the screen, computing some auxiliary technical statistics and storing the output on the disk, all that to make the MATLAB code faster and possible to use in an extensive simulation). Note that the output for $m = 2$ and $n \leq 10\,000$ is usually small enough to be kept in the internal memory; so the last option does not affect the results too much here. Average execution times in seconds are reported in Table 1 and show that the computation hardly takes more than 2 min even for $n = 10\,000$.

Of course, the comparison with the benchmark should be interpreted with care as each program leads to a different output. The proposed MATLAB code produces halfspaces whose intersection equals the sample halfspace depth region. Therefore they can be used straightforwardly for identifying points inside, on, or outside the contours. On the other hand, the benchmark leads to the vertices of the halfspace depth region and identifies its inner points (details were not available to the authors). Both representations may be useful but a vertex–facet or facet–vertex enumeration method has to be used for converting one into the other. Besides, it should be kept in mind that the proposed MATLAB code provides enough material for computing two neighboring contours at once (see the comments below the proof of Theorem 4.2 in HPŠ10) while the benchmark does not.

It should also be noted that the present study does not compare the algorithms but only their implementations. The benchmark has originally been developed only for auxiliary validation, with no speed optimization in mind. On the other hand, the search for the first optimal solution in the MATLAB code is not likely to be the fastest possible as well.

Despite the limitations of this comparison, the results seem to demonstrate high stability and superiority of the proposed code because it was *always* observed faster than the benchmark, sometimes even more than 16 times. It excels especially when applied to medium-sized datasets and not too extreme values of τ.

The decrease of relative efficiency of the proposed code for very small values of τ or n can be explained by the fact that it is the slow finding of the initial solution that contributes the most to the overall execution time in these cases. Indeed, profiling of the code in MATLAB shows that this contribution is usually higher than 30% even for $n = 5000$ if $\tau = 0.01$ (and exceeds 75% for $n = 50$ and the same τ). On the other hand, if $\tau = 0.3$, then this contribution is still often larger than 30% for $n = 50$ but usually drops below 5% for $n = 5000$. Different memory space requirements may also play some role, especially if n is set very high.

5.2. General regression case

Next, the general regression case is considered through the simple model

Table 2
(2D regression settings: $m = 2$.) Average execution time (in seconds) of our code, based on $r = 10$ replications, is provided for quantile order τ, p regressors (including the intercept) and n observations. The numbers in parentheses indicate how many times it is faster than the code from Paindaveine and Šiman (submitted for publication).

p	$n \setminus \tau$	Absolute and relative execution times											
		0.010		0.025		0.050		0.100		0.200		0.400	
1:	100	0.09	(0.7)	0.10	(0.5)	0.13	(0.7)	0.16	(0.6)	0.22	(0.6)	0.27	(0.6)
2:	100	0.09	(0.9)	0.11	(0.9)	0.15	(0.8)	0.19	(0.7)	0.24	(0.7)	0.31	(0.7)
3:	100	0.10	(0.9)	0.12	(0.9)	0.15	(0.9)	0.20	(0.8)	0.28	(0.8)	0.35	(0.7)
6:	100	0.14	(0.9)	0.15	(0.9)	0.20	(0.8)	0.25	(0.8)	0.34	(0.7)	0.46	(0.7)
1:	200	0.12	(0.7)	0.16	(0.6)	0.20	(0.7)	0.28	(0.6)	0.39	(0.6)	0.51	(0.6)
2:	200	0.13	(0.8)	0.17	(0.8)	0.23	(0.8)	0.32	(0.7)	0.44	(0.7)	0.58	(0.7)
3:	200	0.14	(0.9)	0.18	(0.9)	0.24	(0.8)	0.35	(0.8)	0.51	(0.7)	0.66	(0.7)
6:	200	0.18	(0.8)	0.23	(0.8)	0.30	(0.8)	0.45	(0.8)	0.67	(0.7)	0.87	(0.7)
1:	300	0.16	(0.6)	0.20	(0.6)	0.29	(0.7)	0.41	(0.6)	0.59	(0.6)	0.77	(0.6)
2:	300	0.17	(0.9)	0.23	(0.8)	0.32	(0.8)	0.46	(0.7)	0.66	(0.7)	0.88	(0.7)
3:	300	0.18	(0.9)	0.25	(0.8)	0.35	(0.8)	0.53	(0.8)	0.76	(0.7)	1.03	(0.7)
6:	300	0.23	(0.8)	0.33	(0.8)	0.46	(0.8)	0.71	(0.7)	0.99	(0.7)	1.36	(0.7)
12:	300	0.38	(0.8)	0.48	(0.8)	0.67	(0.7)	1.02	(0.7)	1.51	(0.7)	2.10	(0.7)
1:	500	0.22	(0.7)	0.31	(0.6)	0.45	(0.7)	0.66	(0.7)	0.98	(0.6)	1.33	(0.6)
2:	500	0.23	(0.9)	0.34	(0.8)	0.50	(0.8)	0.76	(0.7)	1.12	(0.7)	1.51	(0.7)
3:	500	0.26	(0.9)	0.40	(0.8)	0.59	(0.8)	0.91	(0.7)	1.31	(0.7)	1.77	(0.7)
6:	500	0.35	(0.9)	0.53	(0.8)	0.80	(0.8)	1.22	(0.7)	1.88	(0.7)	2.41	(0.7)
12:	500	0.61	(0.7)	0.87	(0.7)	1.28	(0.7)	2.01	(0.7)	2.83	(0.7)	3.81	(0.7)
1:	1 000	0.41	(0.7)	0.61	(0.7)	0.94	(0.7)	1.46	(0.7)	2.24	(0.7)	3.03	(0.7)
2:	1 000	0.46	(0.8)	0.68	(0.8)	1.06	(0.8)	1.70	(0.8)	2.54	(0.8)	3.44	(0.8)
3:	1 000	0.50	(0.8)	0.79	(0.9)	1.22	(0.8)	1.92	(0.8)	2.96	(0.8)	4.00	(0.8)
6:	1 000	0.65	(0.9)	1.07	(0.9)	1.67	(0.8)	2.67	(0.8)	4.12	(0.8)	5.74	(0.8)
12:	1 000	1.05	(0.9)	1.67	(0.9)	2.66	(0.9)	4.30	(0.8)	7.26	(0.8)	9.04	(0.7)
1:	2 000	0.84	(0.9)	1.41	(0.7)	2.25	(0.7)	3.64	(0.7)	5.59	(0.7)	7.64	(0.7)
2:	2 000	0.97	(0.8)	1.60	(0.9)	2.60	(0.8)	4.21	(0.8)	6.44	(0.8)	8.80	(0.8)
3:	2 000	1.11	(0.9)	1.88	(0.9)	3.04	(0.8)	4.93	(0.8)	7.57	(0.8)	10.35	(0.8)
6:	2 000	1.50	(0.9)	2.65	(0.9)	4.37	(0.8)	7.07	(0.8)	11.00	(0.8)	15.02	(0.8)
12:	2 000	2.55	(0.9)	4.58	(0.8)	7.52	(0.8)	12.47	(0.8)	19.19	(0.8)	26.24	(0.8)
1:	5 000	3.19	(0.8)	6.15	(0.8)	9.83	(0.8)	15.47	(0.8)	23.95	(0.8)	34.75	(0.7)
2:	5 000	3.66	(0.9)	6.83	(0.9)	11.18	(0.9)	18.56	(0.8)	28.56	(0.9)	40.38	(0.8)
3:	5 000	4.48	(1.0)	8.35	(0.9)	14.10	(0.8)	24.34	(0.8)	34.78	(0.9)	63.73	(0.7)
6:	5 000	7.40	(0.9)	13.00	(0.9)	24.66	(0.8)	38.47	(0.8)	55.48	(0.8)	96.50	(0.7)
12:	5 000	18.16	(0.7)	37.23	(0.6)	63.93	(0.7)	86.76	(0.7)	126.10	(0.7)	217.51	(0.6)
1:	10 000	11.91	(0.8)	22.49	(0.8)	37.23	(0.7)	56.15	(0.8)	98.82	(0.7)	138.13	(0.7)
2:	10 000	13.36	(0.8)	25.13	(0.9)	43.38	(0.8)	68.45	(0.9)	112.20	(0.9)	161.86	(0.8)
3:	10 000	16.04	(0.9)	32.49	(0.9)	54.25	(0.8)	85.09	(0.8)	143.79	(0.8)	201.51	(0.8)
6:	10 000	32.09	(0.7)	66.57	(0.7)	110.53	(0.7)	177.90	(0.6)	262.00	(0.7)	386.81	(0.7)
12:	10 000	74.71	(0.7)	156.73	(0.7)	274.30	(0.7)	456.35	(0.7)	733.66	(0.7)	1031.65	(0.6)

$$\boldsymbol{Y}_{p\times 1} = \mathbb{B}_{p\times m}\boldsymbol{X}_{m\times 1} + \boldsymbol{\varepsilon}_{p\times 1},$$

where $X_1 = 1$, $(X_2, \ldots, X_p)'$ has i.i.d. marginals that are uniformly distributed over $(0, 1)$, $\boldsymbol{\varepsilon}$ is p-variate standard normal, and $\mathbb{B}$ can be obtained from the $p \times m$ matrix of ones by replacing the elements in the first column with zeros. Average execution times in seconds (still with the same change to the default settings as in Section 5.1), for a total of r replications, are recorded for many combinations of n, p and τ in Table 2 (for $m = 2$ with $r = 10$), in Table 3 (for $m = 3$ with $r = 5$, and for $m \in \{4, 5\}$ with $r = 3$), and in Table 4 that focuses on outlier detection (for $m = 3$ with $r = 3$ and with very low values of τ).

These results can be used by the reader for estimating the time requirements of his/her own computation with the proposed code. It appears that the computation can hardly take more than some 18 min on average in the case of two-dimensional responses ($m = 2$), $n \leq 10\,000$ and $p \leq 12$. Unfortunately (but not surprisingly), the time requirements and the size of output grow with an increasing dimension of the response. If $m = 3$, then the computation appears advantageous for 500 observations at most, perhaps except for some very low τ's and p's. If $m > 3$, then it is hard to evaluate the correctness of the results. But it appears that all the halfspaces can still be computed in a reasonable time for a few hundreds of observations and extreme τ's even in four and five dimensions, which might be employed for outlier identification.

Virtually the same regression quantile regions can also be obtained from a competing directional (projectional) quantile concept; see Kong and Mizera (2008) and Theorem 4.3 in Paindaveine and Šiman (2011a). Therefore, it makes sense to use as a competitor here the MATLAB implementation for this competing concept; see Paindaveine and Šiman (submitted for publication).

Table 3
(Multidimensional regression settings.) Average execution time (in seconds) of our code, based on $r = 5$ replications if $m = 3$ and on $r = 3$ replications otherwise, is provided for quantile order τ, p regressors (including the intercept) and m-dimensional responses. The numbers in parentheses indicate how many times it is faster than the code from Paindaveine and Šiman (submitted for publication).

p	$n \setminus \tau$	Absolute and relative execution times													
		$m = 3$								$m = 4$				$m = 5$	
		0.010		0.025		0.100		0.200		0.010		0.025		0.010	
1:	100	1.31	(0.7)	2.03	(0.7)	12.29	(0.8)	28.00	(0.8)	6.40	(0.9)	18.56	(0.8)	47.52	(1.0)
2:	100	1.18	(0.9)	2.42	(0.9)	13.13	(0.9)	32.22	(0.9)	7.14	(0.7)	25.47	(0.9)	68.71	(0.6)
3:	100	1.59	(1.0)	2.74	(1.0)	16.09	(0.9)	40.21	(0.9)	11.95	(1.0)	31.56	(1.1)	141.44	(0.9)
4:	100	2.11	(1.1)	3.18	(1.1)	18.61	(1.0)	48.30	(0.9)	20.82	(1.2)	41.35	(1.3)	278.46	(1.2)
6:	100	3.69	(1.2)	4.32	(1.3)	23.61	(1.1)	62.26	(1.0)	56.74	(1.6)	70.54	(1.6)	870.27	(2.1)
1:	200	2.50	(0.8)	6.67	(0.8)	45.69	(0.9)	119.01	(0.9)	22.95	(1.0)	141.65	(0.9)	250.36	(1.1)
2:	200	2.27	(0.9)	6.77	(0.9)	53.65	(0.9)	147.98	(0.9)	21.65	(1.0)	147.92	(1.0)	278.24	(1.0)
3:	200	2.66	(1.0)	7.95	(1.0)	68.20	(1.0)	188.83	(1.0)	31.96	(1.2)	197.96	(1.1)	456.35	(1.2)
4:	200	3.34	(1.1)	9.59	(1.1)	84.39	(1.0)	229.96	(1.0)	42.98	(1.3)	254.30	(1.3)	722.96	(1.4)
6:	200	5.37	(1.2)	12.81	(1.1)	113.01	(1.0)	311.21	(1.0)	96.51	(1.6)	390.32	(1.5)	2 105.63	(1.9)
1:	300	4.05	(0.8)	11.70	(0.9)	106.65	(0.9)	295.88	(1.0)	59.52	(1.0)	399.05	(1.0)	954.42	(1.1)
2:	300	3.89	(1.0)	14.10	(1.0)	130.45	(1.0)	372.47	(1.0)	56.92	(1.1)	532.29	(1.1)	1 043.26	(1.1)
3:	300	4.78	(1.1)	17.45	(1.1)	167.02	(1.0)	488.94	(1.0)	81.45	(1.3)	745.07	(1.2)	1 699.08	(1.3)
4:	300	5.80	(1.1)	20.89	(1.1)	202.64	(1.0)	619.82	(1.0)	111.17	(1.4)	1 011.19	(1.3)	2 601.44	(1.6)
6:	300	8.53	(1.2)	28.38	(1.1)	282.73	(1.0)	914.93	(1.0)	193.51	(1.7)	1 645.31	(1.5)	6 312.76	(2.7)
12:	300	25.99	(1.4)	55.12	(1.3)	552.25	(1.1)	2 049.57	(1.1)						
1:	400	6.40	(0.9)	23.34	(1.0)	200.31	(1.0)	619.14	(1.0)	126.85	(1.1)	1 315.83	(1.1)	2 893.89	(1.1)
2:	400	6.59	(1.0)	25.98	(1.0)	249.81	(1.0)	802.98	(1.0)	134.28	(1.2)	1 613.57	(1.1)	3 378.44	(1.1)
3:	400	8.16	(1.1)	33.05	(1.1)	321.81	(1.1)	1 096.18	(1.0)	190.38	(1.3)	2 345.77	(1.2)	5 887.78	(1.5)
4:	400	9.86	(1.2)	39.29	(1.1)	402.81	(1.1)	1 412.22	(1.0)	260.52	(1.4)	3 340.39	(1.3)	10 795.80	(2.2)
6:	400	13.09	(1.2)	53.08	(1.1)	580.36	(1.1)	2 163.63	(1.0)	418.55	(1.6)	5 824.98	(1.5)	49 921.42	
12:	400	34.72	(1.4)	102.65	(1.2)	1261.70	(1.1)	4 989.74	(1.1)						
1:	500	9.51	(1.0)	32.58	(1.0)	343.61	(1.0)	1 153.78	(1.0)	239.93	(1.1)	2 654.41	(1.1)	16 261.68	
2:	500	9.72	(1.1)	39.45	(1.1)	436.31	(1.1)	1 554.20	(1.1)	256.04	(1.2)	4 007.90	(1.2)	21 937.23	
3:	500	11.93	(1.1)	50.36	(1.1)	571.70	(1.1)	2 169.23	(1.1)	368.81	(1.4)	6 409.06	(1.3)	68 397.79	
4:	500	14.64	(1.2)	61.26	(1.1)	729.85	(1.1)	2 833.84	(1.0)	490.68	(1.5)	10 260.22	(1.7)		
6:	500	20.31	(1.3)	85.52	(1.2)	1057.88	(1.1)	4 438.13	(1.1)	817.93	(1.7)	30 238.22	(2.2)		
12:	500	46.38	(1.4)	160.29	(1.2)	2286.70	(1.1)	10 633.56	(1.1)						

Table 4
(3D outlier detection: $m = 3$.) Average execution time (in seconds) of our code, based on $r = 3$ replications, is provided for quantile order τ, p regressors (including the intercept) and n three-dimensional responses. The numbers in parentheses indicate how many times it is faster than the code from Paindaveine and Šiman (submitted for publication).

τ	Absolute and relative execution times											
	$p \setminus n$	750		1000		1200		1500		2000		
0.010	1:	18.14	(1.1)	35.55	(1.2)	56.09	(1.3)	94.28	(1.4)	195.29	(1.5)	
	2:	23.48	(1.2)	41.68	(1.3)	68.35	(1.3)	117.58	(1.4)	246.84	(1.5)	
	3:	29.19	(1.2)	54.74	(1.3)	88.14	(1.3)	152.60	(1.4)	326.46	(1.5)	
	4:	35.66	(1.2)	67.12	(1.3)	108.72	(1.3)	190.93	(1.4)	418.35	(1.4)	
	6:	48.88	(1.3)	97.84	(1.3)	155.05	(1.2)	290.53	(1.3)	637.53	(1.4)	
0.025	1:	86.18	(1.1)	171.83	(1.2)	262.79	(1.3)	474.01	(1.4)	1 058.50	(1.4)	
	2:	109.89	(1.2)	210.19	(1.3)	328.72	(1.3)	623.02	(1.4)	1 410.02	(1.4)	
	3:	142.86	(1.2)	279.95	(1.2)	439.60	(1.3)	837.58	(1.3)	1 913.23	(1.4)	
	4:	174.72	(1.2)	344.79	(1.2)	561.45	(1.2)	1063.29	(1.3)	2 519.76	(1.3)	
	6:	249.20	(1.2)	512.89	(1.2)	836.57	(1.2)	1649.89	(1.2)	3 914.40	(1.2)	
0.050	1:	294.01	(1.1)	593.70	(1.2)	1006.57	(1.2)	1968.37	(1.3)	4 719.26	(1.3)	
	2:	376.06	(1.2)	760.21	(1.2)	1328.50	(1.2)	2620.97	(1.3)	6 531.65	(1.3)	
	3:	496.03	(1.1)	1042.85	(1.2)	1834.28	(1.2)	3616.83	(1.3)	9 239.28	(1.2)	
	4:	619.62	(1.1)	1328.00	(1.2)	2400.54	(1.2)	4805.97	(1.2)	12 477.26	(1.2)	
	6:	924.38	(1.1)	2039.12	(1.1)	3700.73	(1.1)	7747.52	(1.2)	20 362.39	(1.2)	

Acknowledgements

The research work of Davy Paindaveine was supported by a Mandat d'Impulsion Scientifique of the Fonds National de la Recherche Scientifique, Communauté Française de Belgique, and by an A.R.C. contract of the Communauté Française de Belgique. That of Miroslav Šiman was supported by a *chercheur postdoctoral temporaire* contract of the Fonds National de la

Recherche Scientifique, Communauté Française de Belgique, and by Project 1M06047 of the Ministry of Education, Youth and Sports of the Czech Republic. The authors would like to thank Roger Koenker and Ivan Mizera for inspiring discussions and advice, and express their gratitude to Ivan Mizera for kindly providing the MATLAB code for computation of bivariate halfspace depth contours he coauthored with David Eppstein. Finally, they would like to thank two anonymous referees and an associate editor for their careful reading of the first version of the paper and their constructive comments that led to substantial improvements of the manuscript.

Appendix. Technical details

This section discusses some technical matters related to the algorithm described in this paper.

Choice of τ. If $n\tau$ is an integer, then the linear programming problem (P) has infinitely many solutions for each $\boldsymbol{u}$. If such a complication occurs, it is solved by a small perturbation of τ, which can hardly make any important difference in most applications. Besides, there is only a finite number of different quantile regions anyway, so that such small perturbations of τ could always be done without loss of generality when the goal is only to compute the quantile regions.

Input data. The code assumes $m \in \{2, 3, \ldots, 8\}$ and $n \leq 100\,000$ and its output should be quite reliable for $m \in \{2, 3\}$, $p \leq 10$ and $n \leq 10\,000$ (if $m = 2$) or 500 (if $m = 3$) at least. Now, the program was heavily tested only on data from the simulation study of Section 5, with all coordinates less than 5 or so. This is why it is suggested to standardize the input observations in some way to a similar range whenever possible, which should enhance numerical stability of the algorithm. Besides, most real data are discrete because they are measured or recorded only with limited precision. This makes some bad data configurations more likely than almost impossible. Therefore it is also recommended to perturb the input data points by some random noise of a reasonably small magnitude to prevent their discreteness from causing any trouble.

When a few identical observations occur, one may either aggregate the same rows of $\mathbb{A}_P$ into a single one or introduce (positive) weights into $\boldsymbol{c}_P$ and proceed analogously (the formulae would have to be changed a little but the crucial simplification of (DF) would persist). The first approach is preferred as it is faster, easier to implement and still leads to the right quantile coefficients. Since the algorithm does not rely on any special form of $\boldsymbol{x}_1^c$, the code can also handle such aggregated or weighted rows (corresponding to weighted residuals). Therefore the program can be used even for bootstrap and subsampling methods quite easily. We might also refer to Hlubinka et al. (2010) for another interesting attempt to combine weights with halfspace depth ideas.

Computing the first directional quantile. The problem (P) is solved with the aid of the free MATLAB toolbox SEDUMI 1.1 (see Pólik, 2005; Sturm, 1999) that exploits sparsity and is very fast, flexible, and easy-to-use. Of course, any fast and reliable solver designed for univariate quantile regression might be substituted here.

As mentioned above, the assumption $\boldsymbol{u} \in \mathcal{S}^{m-1}$ can be relaxed without any loss of generality because all non-zero vectors $\boldsymbol{u}$ in the same direction lead to the same upper halfspace $H^{(n)+}_{\tau\boldsymbol{u}}$. In general, the proposed MATLAB code chooses $\boldsymbol{u}_0$ as a normalized corner of the hypercube $[-1, 1]^m$. Large or high-dimensional problems can be solved more effectively by segmenting the whole space to $\mathcal{U}_0$ regions of the form

$$\mathcal{U}_0 = \{\boldsymbol{u} \in \mathbb{R}^m : \text{sign}(\boldsymbol{u}) = \text{sign}(\boldsymbol{u}_0)\}$$

and considering each of these 2^m different orthants separately.

If the starting direction leads to troubles, then other choices are tried until the optimal solution with the required number of non-zero coordinates is found.

Finding non-redundant constraints, facets and interior points. If $m = 2$, then the problem of finding non-redundant constraints and facets can be solved by assigning angles (say θ's) to all the constraints in a clever way. The interior point can then be found simply by means of the facet normal vector.

For $m > 2$, the problem is far more complicated. First, the problem is made bounded by restricting it to vectors $\boldsymbol{u}$ in $[-1, 1]^m$, which turns the cones from (5) into polytopes. Then all vertices and facets of such a polytope are found by means of the dual relationship between vertex and facet enumeration (see Bremner et al., 1998) and program *qhull* (see Barber et al., 1996) for the latter one, fortunately accessible in MATLAB (in fact, it was sufficient to modify the function *con2vert.m* by Michael Kleder from MATLAB Central File Exchange). This enumeration procedure requires an interior point of the resulting polytope to start. It is searched for from the scaled center of the known (parent) facet and in the direction of its normal vector.

In principle, $\boldsymbol{u}_{\mathcal{F}}$ might be found even without the artificial bounding with subsequent vertex enumeration and the zero vertex problem might be addressed as well; see Chvátal (1983). However, the proposed code is tailored for *qhull*, which is an already developed and mature tool for solving similar problems that is quite stable, fast and familiar with rounding errors.

Realization of the breadth-first search algorithm. When this algorithm is employed, then some identifiers (scaled facet centers or facet normal vectors) of all (or lastly) used facets are stored in sorted archive(s) and a new facet is used for building the adjacent cone only if its identifier differs from all those archived, which is checked by the binary search algorithm.

Plotting the contours. The program output describes the upper halfspaces whose intersection equals the quantile region of interest (if all of them are uniquely defined). Vertices of these regions could be obtained by the vertex enumeration mentioned above. The quantile contour with known vertices can then be plotted as their convex hull, for example. Such a procedure was also used to generate all figures of this paper.

Computing many (or all) contours at once. The first (initial) solutions could be found faster for all relevant τ's at once than for each τ separately, by linear programming parametric in τ. In the purely location case, it would be advantageous to compute the contours from the highest $\tau < 0.5$ to the lowest and to reduce the dataset in each step (together with adjusting τ accordingly), since inner points are redundant for computing outer contours. If the interest is even in the individual quantile hyperplanes and their coefficients in the general regression case, one could still replace all the surely interior observations with a single aggregated pseudo-observation keeping the new resulting subgradient conditions the same as before (as Roger Koenker kindly suggested to us). These proposals are not implemented in the proposed MATLAB code as it is designed to compute a single contour only.

References

Barber, C.B., Dobkin, D.P., Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software 22, 469–483.

Bremner, D., Fukuda, K., Marzetta, A., 1998. Primal–dual methods for vertex and facet enumeration. Discrete and Computational Geometry 20, 333–357.

Chakraborty, B., 2003. On multivariate quantile regression. Journal of Statistical Planning and Inference 110, 109–132.

Chaudhuri, P., 1996. On a geometric notion of quantiles for multivariate data. Journal of the American Statistical Association 91, 862–872.

Chvátal, V., 1983. Linear Programming. W.H. Freeman & Co., New York.

Hallin, M., Paindaveine, D., Šiman, M., 2010. Multivariate quantiles and multiple-output regression quantiles: from L_1 optimization to halfspace depth. Annals of Statistics 38, 635–669 (with discussion).

Hlubinka, D., Kotík, L., Vencálek, O., 2010. Weighted halfspace depth. Kybernetika 46, 125–148.

Koenker, R., 2005. Quantile Regression, 1st ed. Cambridge University Press, New York.

Koenker, R., Bassett, G.J., 1978. Regression quantiles. Econometrica 46, 33–50.

Koltchinskii, V., 1997. M-estimation, convexity and quantiles. Annals of Statistics 25, 435–477.

Kon-Popovska, M., 2003. On some aspects of the matrix data perturbation in linear program. Yugoslav Journal of Operations Research 13, 153–164.

Kong, L., Mizera, I., 2008. Quantile tomography: using quantiles with multivariate data. http://arxiv.org/abs/0805.0056.

Lukas, M.A., Shi, M., 2006. Sensitivity analysis of constrained linear L_1 regression: perturbations to constraints, addition and deletion of observations. Computational Statistics & Data Analysis 51, 1213–1231.

Narula, S.C., Wellington, J.F., 2002. Sensitivity analysis for predictor variables in the MSAE regression. Computational Statistics & Data Analysis 40, 355–373.

Paindaveine, D., Šiman, M., 2011a. On directional multiple-output quantile regression. Journal of Multivariate Analysis 102, 193–212.

Paindaveine, D., Šiman, M., 2011b. Computing all directional regression quantiles from projection quantiles (submitted for publication).

Pólik, I., 2005. Addendum to the SeDuMi user guide: version 1.1.

Raković, S.V., Grieder, P., Jones, C., 2004. Computation of Voronoi diagrams and Delaunay triangulation via parametric linear programming. ETH Technical Report AUT04-03.

Rouncefield, M., 1995. The statistics of poverty and inequality. Journal of Statistics Education 3.

Rousseeuw, P.J., Ruts, I., 1999. The depth function of a population distribution. Metrika 49, 213–244.

Shi, M., Lukas, M.A., 2005. Sensitivity analysis of constrained linear L_1 regression: perturbations to response and predictor variables. Computational Statistics & Data Analysis 48, 779–802.

Sturm, J.F., 1999. Using SeDuMi 1.02, a Matlab toolbox for optimization over symmetric cones. Optimization Methods and Software 11–12, 625–653.

Tukey, J.W., 1975. Mathematics and the picturing of data. In: Proceedings of the International Congress of Mathematicians (Vancouver, BC, 1974) Canad. Math. Congress. vol. 2. Quebec, pp. 523–531.

Wei, Y., 2008. An approach to multivariate covariate-dependent quantile contours with application to bivariate conditional growth charts. Journal of the American Statistical Association 103, 397–409.