Computational Statistics manuscript No.
(will be inserted by the editor)

Computing Multiple-Output Regression Quantile Regions from Projection Quantiles

Davy Paindaveine · **Miroslav Šiman**

Received: date / Accepted: date

Abstract In the multiple-output regression context, Hallin, Paindaveine and Šiman (2010) introduced a powerful data-analytical tool based on *regression quantile regions*. However, the computation of these regions, that are obtained by considering in all directions an original concept of directional regression quantiles, is a very challenging problem. Paindaveine and Šiman (2010b) described a first elegant solution relying on linear programming techniques. The present paper provides another solution based on the fact that the quantile regions can also be computed from a competing concept of *projection* regression quantiles, elaborated in Kong and Mizera (2008) and Paindaveine and Šiman (2010a). As a by-product, this alternative solution further provides various characteristics useful for statistical inference. We describe in detail the algorithm solving the parametric programming problem involved, and illustrate the resulting procedure on simulated data. We show through simulations that the MATLAB implementation of the algorithm proposed in this paper is faster than that from Paindaveine and Šiman (2010b) in various cases.

Keywords Directional quantile · Halfspace depth · Multiple-output regression · Parametric programming · Quantile regression

Mathematics Subject Classification (2000) 65C60 · 62J99

Davy Paindaveine
Université Libre de Bruxelles
Avenue F.D. Roosevelt, 50
ECARES, CP114
B-1050 Brussels, Belgium
Tel.: +32-2-650 38 45
Fax: +32-2-650 44 75
E-mail: dpaindav@ulb.ac.be

Miroslav Šiman
Institute of Information Theory and Automation of the ASCR
Pod Vodárenskou věží 4
CZ-18208, Prague 8
Czech Republic

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. In a world where multivariate data are the rule rather than the exception, this clearly has been a severe limitation, which explains why many 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).

A new concept of multiple-output regression quantile, with powerful data-analytical abilities, has recently been defined in Hallin, Paindaveine and Šiman (2010)—hereafter referred to as HPŠ10. In the empirical setup where the m-variate response Y is to be regressed on the p-variate vector of regressors $X = (1, W')'$, this quantile can be defined as follows. For a sample $(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}^m$, $i = 1, \ldots, n$, the HPŠ10 regression (τu)-quantile—for fixed $\tau \in (0,1)$ and $u \in \mathscr{S}^{m-1} := \{y \in \mathbb{R}^m : \|y\| = 1\}$—is defined as any element of the collection $\Pi^{(n)}_{\text{HPŠ};\tau u}$ of hyperplanes $\pi^{(n)}_{\text{HPŠ};\tau u} := \{(w', y')' \in \mathbb{R}^{p-1} \times \mathbb{R}^m : \widehat{b}'_{\text{HPŠ};\tau u} y - \widehat{a}'_{\text{HPŠ};\tau u}(1, w')' = 0\}$, with

$$\begin{pmatrix} \widehat{a}_{\text{HPŠ};\tau u} \\ \widehat{b}_{\text{HPŠ};\tau u} \end{pmatrix} \in \arg\min \sum_{i=1}^{n} \rho_\tau(\widetilde{b}' y_i - \widetilde{a}' x_i) \quad \text{subject to} \quad u'\widetilde{b} = 1, \tag{1}$$

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

This quantile, which is clearly of a directional nature, generates *regression quantile regions* when all directions u are considered for a fixed $\tau \in (0,1)$. More precisely, defining as

$$H^{(n)+}_{\text{HPŠ};\tau u} := \{(w', y')' \in \mathbb{R}^{p-1} \times \mathbb{R}^m : \widehat{b}'_{\text{HPŠ};\tau u} y - \widehat{a}'_{\text{HPŠ};\tau u}(1, w')' \geq 0\}$$

the upper (τu)-quantile halfspace associated with the optimal solution $(\widehat{a}'_{\text{HPŠ};\tau u}, \widehat{b}'_{\text{HPŠ};\tau u})'$ to (1), one can consider the τ-quantile region

$$R^{(n)}_{\text{HPŠ}}(\tau) := \bigcap_{u \in \mathscr{S}^{m-1}} \bigcap \{H^{(n)+}_{\text{HPŠ};\tau u}\} \tag{2}$$

for any $\tau \in (0,1)$, where $\bigcap \{H^{(n)+}_{\text{HPŠ};\tau u}\}$ stands for the intersection over all optimal solutions corresponding to fixed τ and 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 still form a family of polyhedral regions nested up to the classical quantile crossings. As illustrated in Section 7 of HPŠ10, these regression quantile regions allow for a much richer regression analysis than any traditional multiple-output regression method can provide.

Quite interestingly, these quantile regions can also be obtained from a different family of directional multiple-output regression quantiles, elaborated in Kong and Mizera (2008) and Paindaveine and Šiman (2010a). In the same empirical setup as above, these alternative quantiles—referred to as *projection quantiles* in the sequel—can be defined as any element of the collection $\Pi^{(n)}_{\text{proj};\tau u}$ of hyperplanes $\pi^{(n)}_{\text{proj};\tau u} := \{(w',y')' \in \mathbb{R}^{p-1}\times\mathbb{R}^m : \widehat{b}'_{\text{proj};\tau u}y - \widehat{a}'_{\text{proj};\tau u}(1,w')' = 0\}$, with

$$\begin{pmatrix}\widehat{a}_{\text{proj};\tau u}\\ \widehat{b}_{\text{proj};\tau u}\end{pmatrix} \in \arg\min \sum_{i=1}^{n}\rho_\tau(\widetilde{b}'y_i - \widetilde{a}'x_i) \quad \text{subject to} \quad \widetilde{b} = u. \tag{3}$$

In this context, we say that u is a τ-critical direction if there exists a $\pi^{(n)}_{\text{proj};\tau u} \in \Pi^{(n)}_{\text{proj};\tau u}$ that contains exactly $m+p-1$ data points, and we will denote the collection of τ-critical directions by K_τ. Setting $H^{(n)+}_{\text{proj};\tau u} := \{(w',y')' \in \mathbb{R}^{p-1}\times\mathbb{R}^m : \widehat{b}'_{\text{proj};\tau u}y - \widehat{a}'_{\text{proj};\tau u}(1,w')' \geq 0\}$, it can then be shown that, under very mild conditions,

$$R^{(n)}_{\text{HP}\check{\text{S}}}(\tau) = R^{(n)}_{\text{proj}}(\tau) := \bigcap_{u\in\mathscr{S}^{m-1}\cap K_\tau}\bigcap\{H^{(n)+}_{\text{proj};\tau u}\} \tag{4}$$

for any $\tau \in (0,1)$, where $\bigcap\{H^{(n)+}_{\text{proj};\tau u}\}$ stands for the intersection over all optimal solutions for which the corresponding (τu)-quantile hyperplane $\pi^{(n)}_{\text{proj};\tau u}$ contains exactly $m+p-1$ data points; see Theorem 4.4 in Paindaveine and Šiman (2010a). This shows that the HPŠ10 quantile regions can indeed be often obtained from projection quantiles. As an important by-product, we can get many characteristics useful for statistical inference, including the hyperplane coefficients $\widehat{a}_{\text{proj};\tau u}$ and $\widehat{b}_{\text{proj};\tau u}$, and the Lagrange multiplier vectors corresponding to the equality constraint in (3).

In this paper, we develop an algorithm that solves the parametric programming problem (3) and allows to compute efficiently the quantile regions $R^{(n)}_{\text{HP}\check{\text{S}}}(\tau)$ through (3)-(4). The proposed procedure therefore appears as a competitor of the one described in Paindaveine and Šiman (2010b)[1] that computes the regions $R^{(n)}_{\text{HP}\check{\text{S}}}(\tau)$ directly from (1)-(2). We shall see that the problem (3) falls into the category of linear programs with parametric right hand side. They are quite common in practice and their theory is well developed. A MATLAB toolbox for them has also been written; see Kvasnica, Grieder, and Baotić (2004). However, the general problem can be simplified substantially in the special case considered here, which gives rise to the fast and simple solver provided in this paper. Our work confirms the trend that applications of parametric programming in computational geometry still grow in number; see Raković, Grieder, and Jones (2004) for another paper on this topic.

The outline of the paper is as follows. In Section 2, we describe in detail a procedure that solves the parametric programming problem (3) and allows for the computation of the quantile regions $R^{(n)}_{\text{proj}}(\tau)$. In Section 3, we present a step-by-step description of the corresponding algorithm[2]. In Section 4, we provide some illustrations of

[1] The present paper somewhat mimics the structure and wording of Paindaveine and Šiman (2010b) to highlight their mutual similarities and differences.

[2] Our MATLAB implementation of this algorithm can be freely downloaded from the web page `http://homepages.ulb.ac.be/~dpaindav`

these quantile regions and compare them with the quantile regions $R^{(n)}_{\mathrm{HP\check{S}}}(\tau)$ computed from (1)-(2). In Section 5, we conduct some simulations to evaluate the efficiency of our implementation of the algorithm and to compare it with the procedure described in Paindaveine and Šiman (2010b). Finally, some technical matters related to the proposed algorithm are discussed in Section 6.

2 Description of the procedure

In this section, we describe how the problem (3) with given fixed $\tau \in (0,1)$ can be solved for all u's from $\mathscr{S}^{m-1}$ by means of parametric programming, with the focus on τ-critical directions (since these are the only directions to be considered to compute the quantile regions $R^{(n)}_{\mathrm{proj}}(\tau)$ from (3)-(4)). We assume throughout that, when deprived of their first coordinate, the data points $(x_i, y_i) \in \mathbb{R}^p \times \mathbb{R}^m$, $i = 1, \ldots, n$, come from a continuous distribution over $\mathbb{R}^{m+p-1}$. Under this assumption, the algorithm we describe below applies with probability one—problems can be expected only from very exceptional data configurations, typically leading to degeneracy, unwanted zero coordinates or non-invertible matrices in the procedure.

In what follows, we rewrite the problem (3) as a linear program in a convenient way and show that the assumption $u \in \mathscr{S}^{m-1}$ can be relaxed without any harm into $u \in \mathbb{R}^m$ (or more precisely, into $u \in \mathbb{R}^m \setminus \{\mathbf{0}\}$). We demonstrate that (i) the resulting space $\mathbb{R}^m$ of the u's can be segmented into polyhedral cones, each corresponding to *the same optimal basis* of the associated linear program, and that (ii) *the edges (or generating directions) of these cones must comprise all the τ-critical directions*. Besides, we describe the relation between any fixed (non-zero) u in each such cone and the corresponding quantile hyperplane coefficients $\widehat{a}_{\mathrm{proj};\tau u}$ and $\widehat{b}_{\mathrm{proj};\tau u}$, and explain, for any given non-zero vector u_0, how to get the cone containing u_0. Finally, we describe the way how to find all neighboring cones adjacent to a given one by means of simple dual simplex post-optimization, which paves the way for finding the whole conic segmentation, hence for solving the problem completely.

First, let us introduce the following notation. Let 0_ℓ be the ℓ-dimensional zero vector and 1_ℓ be the ℓ-dimensional vector of ones. Denote by $\mathbb{I}_{\ell\times\ell}$ and $\mathbb{O}_{r\times s}$ the ℓ-dimensional identity matrix and the zero $r \times s$ matrix, respectively. The positive and negative parts of an ℓ-vector $v = (v_1, \ldots, v_\ell)'$ are defined as $v_+ := (\max(v_1, 0), \ldots, \max(v_\ell, 0))'$ and $v_- := (\max(-v_1, 0), \ldots, \max(-v_\ell, 0))'$, respectively. We write $r = (r_1, \ldots, r_n)'$ for the vector of residuals $r_i = r_i(\widetilde{a}, \widetilde{b}) := \widetilde{b}\, y_i - \widetilde{a}' x_i$, $i = 1, \ldots, n$. From the $n \times m$ (response) matrix

$$\mathbb{Y} := (y_1, \ldots, y_n)' =: (y_1^c, \ldots, y_m^c)$$

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

$$\mathbb{X} := (x_1, \ldots, x_n)' =: (x_1^c, \ldots, x_p^c),$$

we define

$$\mathbb{U}^y = \mathbb{U}^y_{n\times 2m} := (y_1^c, -y_1^c, \ldots, y_m^c, -y_m^c) \quad \text{and} \quad \mathbb{V}^x = \mathbb{V}^x_{n\times 2p} := (x_1^c, -x_1^c, \ldots, x_p^c, -x_p^c),$$

respectively. In the setup described in the Introduction, $x_1^c = 1_n$. The general notation is used here because sometimes it may be interesting to work with another x_1^c (for example, when multiple identical observations occur in the sample, which may be relevant for resampling procedures) and because our algorithm does not require any special assumption on x_1^c at all. Finally, all vector inequalities are interpreted coordinatewise 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 (3), for any $u \in \mathscr{S}^{m-1}$, can be represented as the linear program

$$\min_{z_P} c_P' z_P \quad \text{subject to} \quad \mathbb{A}_P z_P = b_P,\ z_P \geq 0, \tag{P}$$

with its dual twin brother

$$\max_{\mu_P = (\mu^{b\prime}, \mu_P^{r\prime})'} u'\mu^b \quad \text{subject to} \quad \mathbb{A}_P' \mu_P \leq c_P, \tag{D}$$

where, writing $\mathbb{M}_{m\times 2m}$ for the Kronecker product $\mathbb{I}_{m\times m} \otimes (1,-1)$, we set

$$\begin{aligned}
z_P &= (b'(\widetilde{b}), a'(\widetilde{a}), r_+', r_-')' \in \mathbb{R}^{2m+2p+2n},\\
b &= b(\widetilde{b}) = (\widetilde{b}_{1+}, \widetilde{b}_{1-}, \ldots, \widetilde{b}_{m+}, \widetilde{b}_{m-})' \in \mathbb{R}^{2m},\\
a &= a(\widetilde{a}) = (\widetilde{a}_{1+}, \widetilde{a}_{1-}, \ldots, \widetilde{a}_{p+}, \widetilde{a}_{p-})' \in \mathbb{R}^{2p},\\
c_P &= (0_{2m+2p}', \tau 1_n', (1-\tau)1_n')' \in \mathbb{R}^{2m+2p+2n},\\
b_P &= (u_m', 0_n')' \in \mathbb{R}^{m+n},\\
\mathbb{A}_P &= \begin{pmatrix} \mathbb{A}^1_{P(m\times(2m+2p+2n))} \\ \mathbb{A}^2_{P(n\times(2m+2p+2n))} \end{pmatrix} = \begin{pmatrix} \mathbb{M}_{m\times 2m} & \mathbb{O}_{m\times 2p} & \mathbb{O}_{m\times n} & \mathbb{O}_{m\times n} \\ \mathbb{U}^y_{n\times 2m} & -\mathbb{V}^x_{n\times 2p} & -\mathbb{I}_{n\times n} & \mathbb{I}_{n\times n} \end{pmatrix};
\end{aligned}$$

here, μ_P is the Lagrange multiplier vector corresponding to the equality constraint from (P).

Consider now some u_0 such that there exists a solution $(\widehat{a}'_{\text{proj};\tau u_0}, \widehat{b}'_{\text{proj};\tau u_0})'$ to (3) with only non-zero entries (which implies that u_0 itself has non-zero coordinates only), and denote by $\widehat{z}_P$ the corresponding optimal solution to (P). We then define

- I_b (resp., $\widetilde{I}_b$) as the vector containing indices (sorted in ascending order) of positive coordinates in $b(\widehat{b}_{\text{proj};\tau u_0})$ (resp., in $b(-\widehat{b}_{\text{proj};\tau u_0})$). Note that I_b and $\widetilde{I}_b$ have common dimension m. For instance, if $\widehat{b}_{\text{proj};\tau u_0} = (2,-4)'$, then one has $b(\widehat{b}_{\text{proj};\tau u_0}) = (2,0,0,4)'$, $I_b = (1,4)'$, $b(-\widehat{b}_{\text{proj};\tau u_0}) = (0,2,4,0)'$, and $\widetilde{I}_b = (2,3)'$;
- I_a and $\widetilde{I}_a$ as the vectors obtained by adding $2m$ to each entry of the vectors obtained analogously to I_b and $\widetilde{I}_b$, but from $a(\widehat{a}_{\text{proj};\tau u_0})$ and $a(-\widehat{a}_{\text{proj};\tau u_0})$. Note that I_a and $\widetilde{I}_a$ have common dimension p. With the same $\widehat{b}_{\text{proj};\tau u_0}$ as above (yielding $m = 2$) and $\widehat{a}_{\text{proj};\tau u_0} = (-1,2)'$, one obtains $I_a = (2,3)' + (4,4)' = (6,7)'$ and $\widetilde{I}_a = (1,4)' + (4,4)' = (5,8)'$;
- I_Z, I_e and $\widetilde{I}_e$ as the vectors containing indices (still sorted in ascending order) 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.

We will consider only the case $\pi \neq 0$ and $\nu \neq 0$ below, but the other (simpler) cases can be handled analogously. Finally, we put

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

and

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

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

$$\min_{z_N} c'_N z_N \quad \text{subject to} \quad \mathbb{A}_N z_N = b_N,\ z_N \geq 0, \tag{N}$$

where

$$z_N = z_P(I_C), \quad c_N = c_P(I_C), \quad b_N = b_P,$$

and

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

(the vector b_P remains untouched by this change since its n last components are equal). Alternatively, we can write

$$z_N = \mathbb{P}'_C z_P, \quad c_N = \mathbb{P}'_C c_P, \quad b_N = \begin{pmatrix} \mathbb{I}_{m\times m} & \mathbb{O}_{m\times n} \\ \mathbb{O}_{n\times m} & \mathbb{P}_R \end{pmatrix} b_P,$$

and

$$\mathbb{A}_N = \begin{pmatrix} \mathbb{I}_{m\times m} & \mathbb{O}_{m\times n} \\ \mathbb{O}_{n\times m} & \mathbb{P}_R \end{pmatrix} \mathbb{A}_P \mathbb{P}_C,$$

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

$$\begin{aligned} c_N &= (0'_m,\ 0'_p,\ \tau 1'_\pi,\ (1-\tau)1'_\nu,\ 0'_m,\ 0'_p,\ \tau 1'_p,\ (1-\tau)1'_p,\ (1-\tau)1'_\pi,\ \tau 1'_\nu)' \\ &=: (c'_0,\ c'_1,\ c'_2,\ c'_3,\ \widetilde{c}'_0,\ \widetilde{c}'_1,\ \widetilde{c}'_2,\ \widetilde{c}'_3,\ \widetilde{c}'_4,\ \widetilde{c}'_5)' \\ &=: (c'_{(m+n)\times 1}, \widetilde{c}'_{(m+2p+n)\times 1})' \end{aligned}$$

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

$$\mathbb{B} = \begin{pmatrix} \mathbb{J}_{m\times m} & \mathbb{O}_{m\times p} & \mathbb{O}_{m\times\pi} & \mathbb{O}_{m\times\nu} \\ \mathbb{E}^1_{p\times m} & \mathbb{F}^1_{p\times p} & \mathbb{O}_{p\times\pi} & \mathbb{O}_{p\times\nu} \\ \mathbb{E}^2_{\pi\times m} & \mathbb{F}^2_{\pi\times p} & -\mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ \mathbb{E}^3_{\nu\times m} & \mathbb{F}^3_{\nu\times p} & \mathbb{O}_{\nu\times\pi} & \mathbb{I}_{\nu\times\nu} \end{pmatrix}$$

and

$$\widetilde{\mathbb{B}} = \begin{pmatrix} -\mathbb{J}_{m\times m} & \mathbb{O}_{m\times p} & \mathbb{O}_{m\times p} & \mathbb{O}_{m\times p} & \mathbb{O}_{m\times\pi} & \mathbb{O}_{m\times\nu} \\ -\mathbb{E}^1_{p\times m} & -\mathbb{F}^1_{p\times p} & -\mathbb{I}_{p\times p} & \mathbb{I}_{p\times p} & \mathbb{O}_{p\times\pi} & \mathbb{O}_{p\times\nu} \\ -\mathbb{E}^2_{\pi\times m} & -\mathbb{F}^2_{\pi\times p} & \mathbb{O}_{\pi\times p} & \mathbb{O}_{\pi\times p} & \mathbb{I}_{\pi\times\pi} & \mathbb{O}_{\pi\times\nu} \\ -\mathbb{E}^3_{\nu\times m} & -\mathbb{F}^3_{\nu\times p} & \mathbb{O}_{\nu\times p} & \mathbb{O}_{\nu\times p} & \mathbb{O}_{\nu\times\pi} & -\mathbb{I}_{\nu\times\nu} \end{pmatrix},$$

where $\mathbb{J}$ stands for the invertible $m\times m$ diagonal matrix with $\mathbb{J}_{\ell\ell} = \operatorname{sign}((u_0)_\ell)$, $\ell = 1,\ldots,m$, and where $\mathbb{E}^i$ and $\mathbb{F}^i$, $i = 1,2,3$, 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 $\widehat{z}_N(m+n+1:2m+2p+2n)$ is zero and $\widehat{z}_N(1:m+n) = \mathbb{B}^{-1}b_N = \mathbb{B}^{-1}(:,1:m)u_0$. This readily implies that $\widehat{z}_N(1:m) = \operatorname{abs}(u_0) := u_{0+} + u_{0-}$, and that the norm of u_0 does not affect the resulting quantile hyperplane $\pi^{(n)}_{\text{proj};\tau u_0}$ but only the scale of its coefficients. Also note that $\mathbb{B}^{-1}$ can be easily computed thanks to the special blockwise structure of $\mathbb{B}$. We simply have

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

where

$$\mathbb{C}_1 = \begin{pmatrix} \mathbb{J}_{m\times m} & \mathbb{O}_{m\times p} \\ \mathbb{E}^1_{p\times m} & \mathbb{F}^1_{p\times p} \end{pmatrix}, \quad \mathbb{C}_2 = \begin{pmatrix} \mathbb{E}^2_{\pi\times m} \vdots \mathbb{F}^2_{\pi\times p} \end{pmatrix}, \quad \text{and} \quad \mathbb{C}_3 = \begin{pmatrix} \mathbb{E}^3_{\nu\times m} \vdots \mathbb{F}^3_{\nu\times p} \end{pmatrix}.$$

Blockwise inversion of $\mathbb{C}_1$ analogously leads to

$$\mathbb{C}_1^{-1} = \begin{pmatrix} \mathbb{J} & \mathbb{O} \\ -(\mathbb{F}^1)^{-1}\mathbb{E}^1\mathbb{J} & (\mathbb{F}^1)^{-1} \end{pmatrix} =: \begin{pmatrix} \mathbb{J} & \mathbb{O} \\ \mathbb{K} & \mathbb{L} \end{pmatrix}.$$

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

$$z = \mathbb{B}^{-1}b_N \geq 0_{m+n}, \tag{PF}$$

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

The vector $\widehat{\mu}'_N := (\mu^{b\,\prime}_m, \mu^{r0\prime}_p, \mu^{r+\prime}_\pi, \mu^{r-\prime}_\nu) = c'\mathbb{B}^{-1}$ hidden in (DF) solves the problem dual to (N) and contains the Lagrange multipliers corresponding to the equality constraint in (N). Clearly,

$$\mu^{b\prime} = \tau 1'_\pi(\mathbb{E}^2\mathbb{J} + \mathbb{F}^2\mathbb{K}) - (1-\tau)1'_\nu(\mathbb{E}^3\mathbb{J} + \mathbb{F}^3\mathbb{K}),$$

$$\mu^{r0\prime} = \tau 1'_\pi\mathbb{F}^2\mathbb{L} - (1-\tau)1'_\nu\mathbb{F}^3\mathbb{L}, \qquad \mu^{r+\prime} = -\tau 1'_\pi, \text{ and } \mu^{r-\prime} = (1-\tau)1'_\nu$$

(note that if $p = 1$ and $x^c_1 = 1_n$, then $\mu^{r0} = \tau\pi - (1-\tau)\nu$). The Lagrange multiplier vector $\widehat{\mu}_P$ from the original problem (P) can then be obtained from

$$\widehat{\mu}_N = \begin{pmatrix} \mathbb{I}_{m\times m} & \mathbb{O}_{m\times n} \\ \mathbb{O}_{n\times m} & \mathbb{P}_R \end{pmatrix}\widehat{\mu}_P.$$

Note that $\widehat{\mu}_P(1:m) = \widehat{\mu}_N(1:m)$.

Now, let us partition d according to $\widetilde{c}$ into

$$d = (d'_0, d'_1, d'_2, d'_3, d'_4, d'_5)'.$$

Then simple algebra leads to

$$z = \begin{pmatrix} \mathbb{J} \\ \mathbb{K} \\ \mathbb{E}^2\mathbb{J} + \mathbb{F}^2\mathbb{K} \\ -(\mathbb{E}^3\mathbb{J} + \mathbb{F}^3\mathbb{K}) \end{pmatrix} u \tag{5}$$

$d_0 = 0_m$, $d_1 = 0_p$, $d_2 = -\mu^{r0} - \tau 1_p$, $d_3 = \mu^{r0} - (1-\tau)1_p = -d_2 - 1_p$, $d_4 = -\tau 1_\pi - (1-\tau)1_\pi = -1_\pi$, and $d_5 = -(1-\tau)1_\nu - \tau 1_\nu = -1_\nu$. Note that the matrices $\mathbb{E}^i$, $i = 1,2,3$, are used only in the product with $\mathbb{J}$ that can be obtained from $\mathbb{E}\mathbb{J} = \mathbb{Y}(I_R,:)$, where we set $\mathbb{E} := (\mathbb{E}^{1\prime} \vdots \mathbb{E}^{2\prime} \vdots \mathbb{E}^{3\prime})'$.

Note as well that (DF) is equivalent to

$$(d'_2, d'_3)' \leq 0_{2p},$$

which can be rewritten as

$$-\tau 1_p \leq \mu^{r0} \leq (1-\tau)1_p.$$

Most importantly, (DF) must be satisfied at least for u_0 and does depend on u only implicitly through $\mathbb{B}$. Consequently, $\mathbb{B}$ remains dual feasible whenever (PF) holds.

All u's satisfying (PF) form a polyhedral cone, say $\mathscr{C}_{u_0}$. Such cones (corresponding to various u_0's) fill the whole space $\mathbb{R}^m$ and our goal is to find them all, together with the corresponding optimal bases. Note that all u's in the interior of such cones lead to the hyperplane solutions fitting exactly p observations and that the solution hyperplanes fitting $m+p-1$ data points must be included among those corresponding to the generating vectors of all these cones.

Let us assume that we have identified all non-redundant constraints in (PF) and facets of $\mathscr{C}_{u_0}$. Each such facet must be shared with another (adjacent) cone. That is why we may simply pass through all the cones $\mathscr{C}_u$ counter-clockwise when $m = 2$. In general, it is possible to use the breadth-first search algorithm and always consider all such $\mathscr{C}_u$'s that are adjacent to a cone treated in the previous step and have not been considered yet.

It remains to clarify the process leading to the adjacent cone from a facet of $\mathscr{C}_{u_0}$. This facet, say, corresponds to the i-th constraint in (PF) and has an interior point u_f. This point is still certain to meet the dual feasibility conditions (DF) and therefore we may further proceed with the dual simplex post-optimization until the optimal basis of the adjacent cone is found.

Let us describe this process in detail. The $I_C(i)$-th original variable will be the first to leave the basis. Then we should compute the auxiliary vector

$$t = (t'_0, t'_1, t'_2, t'_3, t'_4, t'_5)' = \mathbb{B}^{-1}(i,:)\widetilde{\mathbb{B}}$$

(partitioned according to d) and find an index j satisfying

$$\frac{d_j}{t_j} = \min\Big\{\frac{d_h}{t_h} : t_h < 0, h = 1,\ldots,m+2p+n\Big\}.$$

The $I_C(n+m+j)$-th original variable should then replace the removed one. We get a new dual feasible basis, say $\mathbb{B}_1$.

Note that

	$i \le m$	$m+1 \le i \le m+p$	$m+p+1 \le i \le m+p+\pi$	$m+p+\pi+1 \le i$
$t_0' =$	$-\mathbb{I}_{m\times m}(i,:)$	$0_m'$	$0_m'$	$0_m'$
$t_1' =$	$0_p'$	$-\mathbb{I}_{p\times p}(i-m,:)$	$0_p'$	$0_p'$
$t_2' =$	$0_p'$	$-\mathbb{L}(i-m,:)$	$-\mathbb{F}^2(i-m-p,:)\mathbb{L}$	$\mathbb{F}^3(i-m-p-\pi,:)\mathbb{L}$
$t_3' =$	$0_p'$	$\mathbb{L}(i-m,:)$	$\mathbb{F}^2(i-m-p,:)\mathbb{L}$	$-\mathbb{F}^3(i-m-p-\pi,:)\mathbb{L}$
$t_4' =$	$0_\pi'$	$0_\pi'$	$-\mathbb{I}_{\pi\times\pi}(i-m-p,:)$	$0_\pi'$
$t_5' =$	$0_\nu'$	$0_\nu'$	$0_\nu'$	$-\mathbb{I}_{\nu\times\nu}(i-m-p-\pi,:)$

Consequently, we may consider only certain subvectors of t and d without any loss of generality.

If $i \le m+p$, then we can always choose $j = i$, which only changes the sign of a regression coefficient. If $i > m+p$, then the optimal j results from

$$\frac{d_j}{t_j} = \min\Big\{\frac{d_h}{t_h} : t_h < 0, h = m+p+1,\ldots,m+3p\Big\}$$

if this fraction is less than or equal to 1; otherwise we can set $j = 2p+i$. If moreover $p = 1$ and the only regressor is the unit vector, then $(t_2', t_3')'$ equals $(-1,1)'$ for $i \le m+p+\pi$ and $(1,-1)'$ otherwise.

The resulting basis $\mathbb{B}_1$ is optimal if and only if

$$z_1 = \mathbb{B}_1^{-1}(:, 1:m)u_f \ge 0_{m+n}, \tag{6}$$

which appears to hold with probability one (otherwise, we would have to repeat the previous steps until the optimal basis of the adjacent cone is found, which is not supported by the accompanying MATLAB code).

3 Algorithm

To sum up, the basic form of the algorithm can always be performed in the following steps where the highlighted text refers to the topical subsections of Section 6 that discuss some issues in more detail:

1. Adjust the data and τ if necessary; see *Input Data* and *Choice of* τ, respectively.
2. Consider (P) and find its optimal basis $\mathbb{B} = \mathbb{B}(u_0)$ for a given directional vector u_0, see *Computing the first directional quantile*.
3. Set $\mathscr{B}_{\text{new}} := \{\mathbb{B}(u_0)\}$.
4. Set $\mathscr{B}_{\text{old}} := \mathscr{B}_{\text{new}}$, then $\mathscr{B}_{\text{new}} := \emptyset$.
5. For each $\mathbb{B} = \mathbb{B}(u)$ in $\mathscr{B}_{\text{old}}$,

(a) compute $z = z(u)$ from (5) that determines the inequalities in (PF) defining the cone $\mathscr{C}_u$ of all directions leading to the same optimal basis $\mathbb{B}$ as u
(b) find facets and vertices of $\mathscr{C}_u$; see *Finding non-redundant constraints, facets and interior points*
(c) drop all facets of $\mathscr{C}_u$ whose adjacent cones have already been investigated; see *Realization of the breadth-first search algorithm*
(d) for each remaining facet of $\mathscr{C}_u$, find its interior point u_f and use it in the simplex post-optimization step to determine the optimal basis $\mathbb{B}_{\text{new}}(u_f)$ of the adjacent cone sharing that facet with $\mathscr{C}_u$ (as described at the end of Section 2), and add $\mathbb{B}_{\text{new}}(u_f)$ to $\mathscr{B}_{\text{new}}$;

6. If $\mathscr{B}_{\text{new}}$ is non-empty, go to Step 4.

If $\mathscr{B}_{\text{new}}$ is empty, then the algorithm terminates successfully (all cones $\mathscr{C}$ have been found and there is no new cone facet to investigate).

4 Illustrations

This section presents some illustrative examples of the quantile regions $R^{(n)}_{\text{proj}}(\tau)$ obtained from our MATLAB implementation of the procedure described above. What we plot for each region is actually the corresponding quantile contour, namely its boundary $\partial R^{(n)}_{\text{proj}}(\tau)$. In each case, we compare the results with the contours $\partial R^{(n)}_{\text{HP}\check{\text{S}}}(\tau)$ computed from (1)-(2), as described in Paindaveine and Šiman (2010b).

Bivariate location case. We start with the bivariate location case obtained with $m = 2$ and $p = 1$. We independently generated data points y_i, $i = 1,\dots,n = 2499$, from the uniform distribution over the unit square $[0,1]^2$. Figure 1(a) plots the resulting quantile contours $\partial R^{(n)}_{\text{proj}}(\tau)$ 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 versions—namely the population halfspace depth contours (see Rousseeuw and Ruts 1999)—and seem to coincide, as expected, with the contours $\partial R^{(n)}_{\text{HP}\check{\text{S}}}(\tau)$ plotted in Figure 1(c)—that are computed from (1)-(2). Our code can deal with weighted observations (which in particular allows for multiple observations): if weights $\omega_i > 0$, $i = 1,\dots,n$ (summing up to one or not) are given, the resulting "weighted" optimization problem is obtained by substituting $y_{\omega i} := \omega_i y_i$ and $x_{\omega i} := \omega_i x_i$, $i = 1,\dots,n$, for the y_i's and x_i's in (3). Figure 1(b) reports, for the same τ's as in Figure 1(a), the quantile contours $\partial R^{(n)}_{\text{proj}}(\tau)$ associated with weighted data points $y_{\omega i} := \omega_i y_i$, $i = 1,\dots,n$, where the weights are given by

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

and the original data points y_i are the same as in Figure 1(a). The ten points plotted in Figure 1(b) are the original data points y_i, $i = 1,\dots,10$, that receive the larger weight. The corresponding contours $\partial R^{(n)}_{\text{HP}\check{\text{S}}}(\tau)$, which are reported in Figure 1(d), still appear to be equal to the contours computed from projection quantiles.

Trivariate location case. Figure 2 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 $[0,1]^3$. Figure 2(a) reports the quantile contours $\partial R^{(n)}_{\text{proj}}(\tau)$ computed from projection quantiles for $\tau =$ 0.05, 0.15, and 0.25. The only other method available for computing halfspace depth regions beyond dimension two is the one from Paindaveine and Šiman (2010b) that is based on (1)-(2); the corresponding contours $\partial R^{(n)}_{\text{HPŠ}}(\tau)$ are plotted in Figure 2(b). As expected, both methods seem to lead to the same contours.

Regression setup with two responses and one random covariate. We consider the simple heteroscedastic regression model

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

where the random covariate W is uniformly distributed over $[0,1]$ and the random vector ε (which is independent of W) is uniformly distributed over the unit square $[0,1]^2$. From this model, we independently generated data points $(x_i', y_i')' = (1, w_i, y_i')' \in \mathbb{R}^p \times \mathbb{R}^m = \mathbb{R}^2 \times \mathbb{R}^2$, $i = 1, \ldots, n = 249$. Figure 3(a) displays the resulting (trivariate, since they are objects of the (w,y)-space) regression quantile contours $\partial R^{(n)}_{\text{proj}}(\tau)$ for $\tau = 0.05$, 0.15, 0.30, and 0.45. Figure 3(b) provides the corresponding regression quantile contours $\partial R^{(n)}_{\text{HPŠ}}(\tau)$ computed from (1)-(2) as described in Paindaveine and Šiman (2010b).

5 Simulations

We now present some empirical results that quantify the speed (and show the possibilities) of our MATLAB implementation of the procedure described in this paper. We used an Apple computer with Intel Core Duo 1.83GHz, 512MB RAM, WIN XP SP2 and MATLAB 7.3.0.267. Of course, other hardware or initial settings may lead to different results.

5.1 Location case

We first focus on the location case ($p = 1$), hence on the computation of halfspace depth contours. We considered only the bivariate case ($m = 2$) so that we could compare the MATLAB implementation of the method described in this paper with that coauthored and kindly provided to us by Ivan Mizera that we chose for a benchmark (in Section 5.2, we will use the code from Paindaveine and Šiman (2010b), which is the only competitor available in the general regression case).

We generated n i.i.d. bivariate ($m = 2$) observations (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 $n \in \{50, 100, 150, 200, 300, 500, 1000,$

$2000, 5000, 10000, 20000\}$, we ran the computation ten times for each scenario S[3]. Average execution times in seconds are reported in Tables 1 and 2 (for S=1 and S=2, respectively) and show that the computation hardly takes more than 1 minute and a half even for $n = 10000$. As expected, they increase with τ and they are higher for S=2 than for S=1 if τ is low while the reverse becomes true with growing τ. However, their dependence on n seems, on average, higher than linear and not worse than quadratic in any case.

For the same reasons as those presented in Paindaveine and Šiman (2010b), the comparison with the benchmark is not free of limitations. Still, the results seem to demonstrate high stability and superiority of the code proposed in this paper over the benchmark because it was *always* observed faster, sometimes even more than 27 times. It excels especially when applied to medium-sized data sets and not too extreme values of τ.

The decrease of relative efficiency of our code for very small values of τ or n can be explained by the fact that it is the inefficient 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 70% 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 can also play some role, especially if n is set very high.

5.2 General regression case

Next we consider the general regression context represented by the simple model

$$Y_{p\times 1} = \mathbb{B}_{p\times m} X_{m\times 1} + \varepsilon_{p\times 1},$$

where $X_1 = 1$, $(X_2, \ldots, X_p)'$ has i.i.d. marginals that are uniformly distributed over $(0,1)$, ε 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[4] in seconds, for a total of r replications, are recorded for many combinations of n, p and τ in Table 3 (for $m = 2$ with $r = 10$) and in Table 4 (for $m = 3$ with $r = 5$, and for $m \in \{4,5\}$ with $r = 3$).

Here, the only competitor is the code described in Paindaveine and Šiman (2010b). The code presented in this paper seems to slightly outperform that of Paindaveine and Šiman (2010b) at least if $m = 2$ or (in the location case) if $m = 3$.[5]

[3] Actually, with the following changes to the default settings of our code: CTechST.PFZ1CheckI = 0, CTechST.InCheckI = 0, CTechST.ReportI = 0, CTechST.TestModeI = 0, and CTechST.OutSaveI = 0. This suppresses some almost surely redundant testing, 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 our code faster and possible to use in an extensive simulation. Note that the output for $m = 2$ and $n \leq 10000$ is usually small enough to be kept in the internal memory; so the last option does not affect the results too much here.

[4] Still with the same changes to the default settings as in Section 5.1.

[5] Nevertheless, note that the accuracy of efficiency ratios is limited due to the rounding error of execution times and that the average execution times themselves may (slightly) differ even for the same data set, which follows from comparing corresponding values from Tables 1 and 3.

6 Technical Details

In this section, we are going to discuss in more detail some technical matters related to the algorithm described in Sections 2 and 3.

Choice of τ. If $n\tau$ is an integer and $p = 1$, then linear programming problem (P) has infinitely many solutions for each u. In general, if such a complication occurs, we solve it 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 $n \leq 100000$ and its output should be quite reliable for $m \in \{2,3\}$, $p \leq 10$ and $n \leq 10000$ (if $m = 2$) or 500 (if $m = 3$) at least. The program was heavily tested only on data from our simulation study, with all coordinates less than 5 or so. This is why we suggest 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 we also recommend to perturb the input data points by some random noise of a reasonably small magnitude to prevent their discreteness from causing any troubles.

When a few identical observations occur, we may either aggregate the same rows of $\mathbb{A}_P$ into a single one or introduce (positive) weights into c_P and proceed analogously (the formulae would have to be changed a little but the crucial simplification of (DF) would persist). We prefer the first approach that is faster, easier to implement and still leads to the same quantile coefficients. Since the algorithm does not rely on any special form of x_1^c, the code can also handle such aggregated or weighted rows (corresponding to weighted observations). Therefore the program can be used even for bootstrap and subsampling methods quite easily. We might also refer to Hlubinka, Kotík, and Vencálek (2010) for another interesting attempt to combine weights and halfspace depth ideas.

Computing the first directional quantile. We decided to solve (P) with the aid of free MATLAB toolbox SEDUMI 1.1 (see Pólik 2005 and Sturm 1999) that exploits sparsity and is very fast, flexible, and easy-to-use. Any fast and reliable solver designed for univariate quantile regression might be substituted here. In fact, only fast computation of ordinary quantiles is needed (but not implemented) if $p = 1$.

As mentioned above, we can relax the assumption $u \in \mathscr{S}^{m-1}$ without any loss of generality because all nonzero vectors u in the same direction lead to the same upper halfspace $H^{(n)+}_{\mathrm{proj};\tau u}$. In general, we choose u_0 as a normalized corner of $[-1,1]^m$. Large or high-dimensional problems can be solved more effectively by segmenting the whole space to $\mathscr{U}_0$ regions of the form

$$\mathscr{U}_0 = \{u \in \mathbb{R}^m : \mathrm{sign}(u) = \mathrm{sign}(u_0)\},$$

and considering each of these 2^m different orthants separately. Note that the first m constraints in (PF) are trivially satisfied in each $\mathscr{U}_0$.

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, we make the problem bounded by restricting to vectors u in $[-1,1]^m$, which turns the cones into polytopes. Then we find all vertices and facets of such a polytope by means of the dual relationship between vertex and facet enumeration (see Bremner, Fukuda, and Marzetta (1998)) and program *qhull* (see Barber, Dobkin, and Huhdanpaa (1996)) for the latter one, fortunately accessible in MATLAB (In fact, we only 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. We search for it from the scaled center of the known (parent) facet and in the direction of its normal vector.

In principle, u_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, we decided to tailor our code 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 provides upper halfspaces including those whose intersection equals the quantile region of our interest (if all of them are uniquely defined). Vertices of these regions could be obtained by the vertex enumeration mentioned above (and also used in our code in a different context). The quantile contour with known vertices can be plotted easily as their convex hull, for example. This is essentially the procedure we used to generate all figures of the present 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 data set in each step (with adjusting τ accordingly), since inner points are redundant for computing outer contours. If we were interested even in the individual quantile hyperplanes and their coefficients, we 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 our code as it is designed to compute a single contour only.

Acknowledgements The research work of Davy Paindaveine, who is also member of ECORE (the association between CORE and ECARES), 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 partly supported by a *chercheur post-doctoral temporaire* contract of the Fonds National de la Recherche Scientifique, Communauté française de Belgique, and partly 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 advices. They also 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 for their careful reading of the first version of the paper and their constructive comments that led to substantial improvements of the manuscript.

References

1. Barber CB, Dobkin DP, and Huhdanpaa H (1996) The quickhull algorithm for convex hulls. ACM Trans. Math. Software 22:469-483.
2. Bremner D, Fukuda K, Marzetta A (1998) Primal-dual methods for vertex and facet enumeration. Discrete Comput. Geom. 20:333-357.
3. Chakraborty B (2003) On multivariate quantile regression. J. Statist. Plann. Inference 110:109-132.
4. Chaudhuri P (1996) On a geometric notion of quantiles for multivariate data. J. Amer. Statist. Assoc. 91:862-872.
5. Chvátal V (1983) Linear programming. W.H. Freeman & co, New York.
6. Hallin M, Paindaveine D, Šiman M (2010) Multivariate quantiles and multiple-output regression quantiles: From L_1 optimization to halfspace depth (with discussion). Ann. Statist. 38:635-669.
7. Hlubinka D, Kotík L, Vencálek O (2010) Weighted halfspace depth. Kybernetika 46:125-148.
8. Koenker R, Bassett GJ (1978) Regression quantiles. Econometrica 46:33-50.
9. Koltchinskii V (1997) M-estimation, convexity and quantiles. Ann. Statist. 25:435-477.
10. Kong L, Mizera I (2008) Quantile tomography: using quantiles with multivariate data. Submitted. http://arxiv.org/abs/0805.0056.
11. Kvasnica M, Grieder P, Baotić M (2004) Multi-Parametric Toolbox (MPT). http://control.ee.ethz.ch/~mpt
12. Narula SC, Wellington JF (2002) Sensitivity analysis for predictor variables in the MSAE regression. Comput. Statist. Data Anal. 40:355-373.
13. Paindaveine D, Šiman M (2010a) On directional multiple-output quantile regression. Tentatively accepted in J. Multivariate Anal.
14. Paindaveine D, Šiman M (2010b) Computing multiple-output regression quantile regions. Tentatively accepted in Comput. Statist. Data Anal.
15. Pólik I (2005) Addendum to the SeDuMi user guide: Version 1.1. Reference guide.
16. Raković SV, Grieder P, Jones C (2004) Computation of Voronoi diagrams and Delaunay triangulation via parametric linear programming. ETH Technical Report AUT04-03.
17. Rousseeuw PJ, Ruts I (1999) The depth function of a population distribution. Metrika 49:213-244.
18. Sturm JF (1999) Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. Optim. Methods Softw. 11-12:625-653.
19. Wei Y (2008) An approach to multivariate covariate-dependent quantile contours with application to bivariate conditional growth charts. J. Amer. Statist. Assoc. 103:397-409.

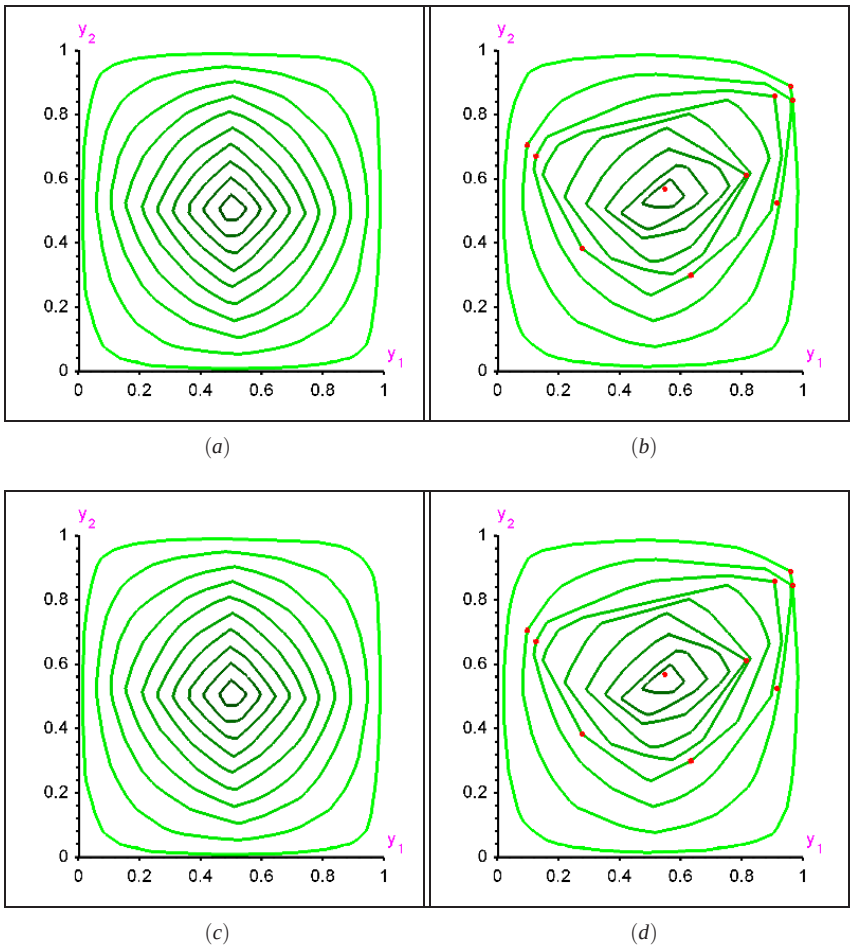
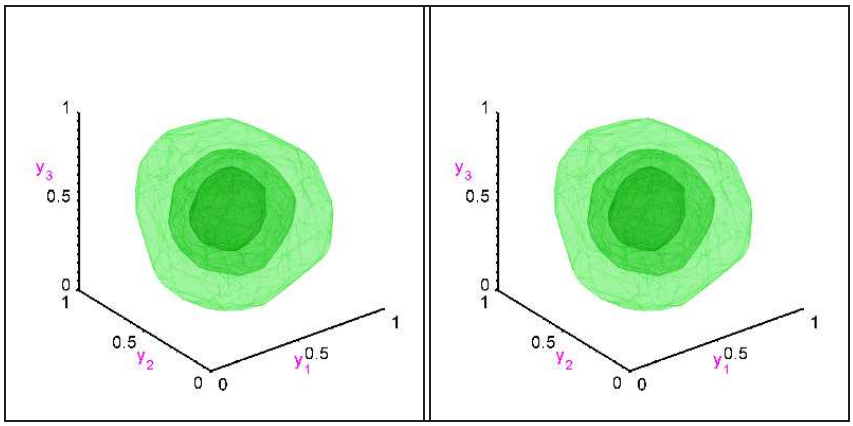
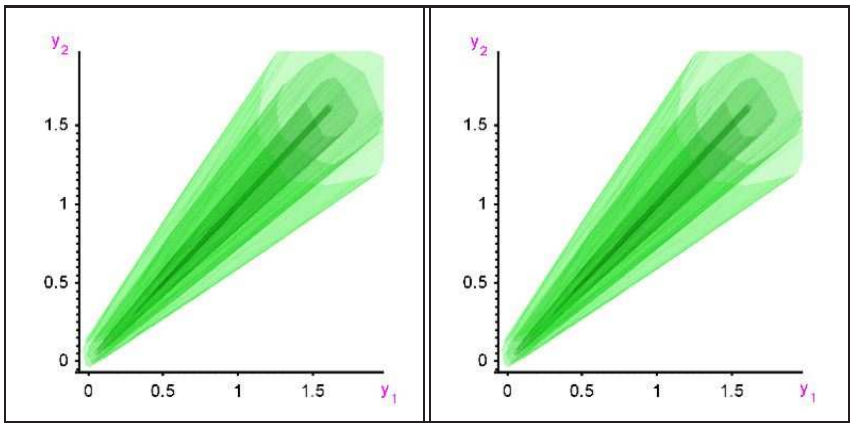


Fig. 1 In Subfigure (a), quantile contours $\partial R^{(n)}_{\mathrm{proj}}(\tau)$ of order $\tau = 0.01$, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, and 0.45 are plotted, by using the method described in this paper, from a sample of $n = 2499$ observations drawn independently from the uniform distribution over $[0,1]^2$. Subfigure (b) reports the corresponding contours after the weights of the first ten data points (plotted in red) were changed from 1 to 2499/20. Subfigures (c)-(d) provide the corresponding quantile contours $\partial R^{(n)}_{\mathrm{HP\check{S}}}(\tau)$ computed from the method described in Paindaveine and Šiman (2010b).



(*a*) (*b*)

Fig. 2 In Subfigure (a), quantile contours $\partial R^{(n)}_{\mathrm{proj}}(\tau)$ of order $\tau = 0.05$, 0.15, and 0.25 are plotted by using the method described in this paper from a sample of $n = 249$ observations drawn independently from the uniform distribution over $[0,1]^3$. Subfigure (b) provides the corresponding quantile contours $\partial R^{(n)}_{\mathrm{HP\check{S}}}(\tau)$ computed as described in Paindaveine and Šiman (2010b).



(*a*) (*b*)

Fig. 3 Subfigure (a) reports the quantile contours $\partial R^{(n)}_{\mathrm{proj}}(\tau)$ 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 (to which we refer for details). Subfigure (b) provides the corresponding quantile contours $\partial R^{(n)}_{\mathrm{HP\check{S}}}(\tau)$ computed as described in Paindaveine and Šiman (2010b).

Average absolute and relative execution times

$n\backslash\tau$	0.010		0.025		0.050		0.100		0.200		0.400	
50	0.03	(4.3)	0.04	(5.0)	0.04	(7.0)	0.07	(6.3)	0.09	(8.1)	0.10	(9.5)
100	0.06	(5.7)	0.05	(8.4)	0.09	(8.4)	0.10	(11.1)	0.14	(13.0)	0.17	(15.2)
150	0.05	(8.4)	0.07	(10.1)	0.09	(12.9)	0.14	(14.1)	0.19	(17.6)	0.23	(20.0)
200	0.07	(9.4)	0.10	(11.0)	0.13	(13.4)	0.18	(16.1)	0.23	(20.6)	0.30	(22.6)
300	0.09	(11.8)	0.12	(14.2)	0.19	(15.6)	0.26	(19.6)	0.36	(23.3)	0.47	(25.9)
500	0.16	(11.1)	0.19	(16.6)	0.29	(18.8)	0.42	(22.6)	0.62	(25.6)	0.81	(27.7)
1000	0.27	(15.2)	0.44	(17.6)	0.65	(20.8)	0.99	(23.9)	1.51	(26.0)	2.05	(27.7)
2000	0.75	(12.4)	1.03	(18.2)	1.67	(19.9)	2.66	(22.0)	4.06	(23.8)	5.74	(24.2)
5000	2.20	(13.3)	4.33	(14.2)	6.96	(15.6)	11.44	(16.9)	18.27	(17.7)	26.20	(17.5)
10000	8.18	(8.8)	14.34	(10.6)	24.88	(11.0)	42.14	(11.6)	68.00	(12.0)	97.04	(12.1)
20000	26.65	(9.4)	56.10	(9.2)	96.90	(9.8)	168.34	(9.9)	267.38	(10.3)	373.98	(10.4)

Table 1 (2D location setting: $m = 2$ and $p = 1$, scenario S=1) Average execution time (in seconds) of our code is provided for scenario S=1, 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.

Average absolute and relative execution times

$n\backslash\tau$	0.010		0.025		0.050		0.100		0.200		0.400	
50	0.04	(4.3)	0.05	(4.8)	0.05	(6.4)	0.07	(7.0)	0.09	(7.8)	0.10	(8.8)
100	0.06	(6.3)	0.07	(7.7)	0.09	(9.2)	0.11	(11.8)	0.13	(14.9)	0.16	(15.4)
150	0.06	(8.2)	0.11	(7.7)	0.10	(13.5)	0.15	(14.5)	0.19	(17.7)	0.22	(19.9)
200	0.09	(9.1)	0.12	(10.9)	0.15	(13.7)	0.19	(16.8)	0.24	(20.1)	0.29	(22.1)
300	0.12	(10.3)	0.14	(14.3)	0.22	(15.2)	0.28	(20.1)	0.35	(23.7)	0.44	(25.7)
500	0.18	(12.1)	0.22	(17.0)	0.34	(18.7)	0.46	(22.8)	0.63	(25.6)	0.78	(27.8)
1000	0.42	(12.0)	0.55	(17.9)	0.77	(20.7)	1.10	(23.8)	1.55	(26.1)	1.95	(27.2)
2000	0.90	(13.3)	1.30	(17.7)	1.96	(19.5)	2.92	(22.0)	4.14	(23.9)	5.50	(23.8)
5000	3.91	(9.7)	5.39	(14.2)	8.32	(15.5)	12.68	(16.9)	18.87	(17.5)	24.96	(17.6)
10000	12.06	(7.9)	19.13	(10.2)	30.35	(10.8)	47.91	(11.3)	70.27	(12.0)	91.92	(12.0)
20000	37.87	(8.5)	69.58	(9.4)	117.69	(9.1)	189.72	(9.5)	276.51	(10.1)	356.12	(10.2)

Table 2 (2D location settings: $m = 2$ and $p = 1$, scenario S=2) Average execution time (in seconds) of our code is provided for scenario S=2, 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.

Average absolute and relative execution times

p :	$n\backslash\tau$	0.010	0.025	0.050	0.100	0.200	0.400
1 :	100	0.06 (1.5)	0.05 (2.0)	0.09 (1.4)	0.10 (1.6)	0.14 (1.6)	0.17 (1.6)
2 :	100	0.08 (1.1)	0.10 (1.1)	0.12 (1.3)	0.14 (1.4)	0.17 (1.4)	0.21 (1.5)
3 :	100	0.09 (1.1)	0.11 (1.1)	0.13 (1.2)	0.16 (1.3)	0.21 (1.3)	0.25 (1.4)
6 :	100	0.12 (1.2)	0.13 (1.2)	0.15 (1.3)	0.20 (1.3)	0.25 (1.4)	0.32 (1.4)
1 :	200	0.08 (1.5)	0.10 (1.6)	0.13 (1.5)	0.18 (1.6)	0.24 (1.6)	0.31 (1.6)
2 :	200	0.11 (1.2)	0.14 (1.2)	0.18 (1.3)	0.23 (1.4)	0.31 (1.4)	0.40 (1.4)
3 :	200	0.12 (1.2)	0.16 (1.1)	0.20 (1.2)	0.27 (1.3)	0.37 (1.4)	0.47 (1.4)
6 :	200	0.15 (1.2)	0.19 (1.2)	0.25 (1.2)	0.34 (1.3)	0.48 (1.4)	0.61 (1.4)
1 :	300	0.09 (1.8)	0.12 (1.7)	0.20 (1.4)	0.26 (1.6)	0.37 (1.6)	0.49 (1.6)
2 :	300	0.15 (1.1)	0.19 (1.2)	0.25 (1.3)	0.34 (1.4)	0.47 (1.4)	0.62 (1.4)
3 :	300	0.16 (1.1)	0.21 (1.2)	0.28 (1.2)	0.40 (1.3)	0.56 (1.4)	0.72 (1.4)
6 :	300	0.19 (1.2)	0.27 (1.2)	0.38 (1.2)	0.50 (1.4)	0.71 (1.4)	0.96 (1.4)
12 :	300	0.30 (1.3)	0.37 (1.3)	0.50 (1.3)	0.70 (1.5)	1.05 (1.4)	1.44 (1.5)
1 :	500	0.16 (1.4)	0.19 (1.6)	0.30 (1.5)	0.44 (1.5)	0.63 (1.6)	0.85 (1.6)
2 :	500	0.20 (1.2)	0.27 (1.3)	0.39 (1.3)	0.56 (1.4)	0.81 (1.4)	1.08 (1.4)
3 :	500	0.24 (1.1)	0.32 (1.3)	0.46 (1.3)	0.66 (1.4)	0.96 (1.4)	1.28 (1.4)
6 :	500	0.30 (1.2)	0.43 (1.2)	0.61 (1.3)	0.87 (1.4)	1.30 (1.4)	1.74 (1.4)
12 :	500	0.44 (1.4)	0.61 (1.4)	0.91 (1.4)	1.37 (1.5)	2.06 (1.4)	2.59 (1.5)
1 :	1000	0.28 (1.5)	0.44 (1.4)	0.67 (1.4)	1.02 (1.4)	1.53 (1.5)	2.10 (1.4)
2 :	1000	0.37 (1.2)	0.57 (1.2)	0.84 (1.3)	1.33 (1.3)	1.97 (1.3)	2.70 (1.3)
3 :	1000	0.42 (1.2)	0.68 (1.2)	1.01 (1.2)	1.54 (1.2)	2.34 (1.3)	3.15 (1.3)
6 :	1000	0.61 (1.1)	0.92 (1.2)	1.41 (1.2)	2.14 (1.2)	3.26 (1.3)	4.34 (1.3)
12 :	1000	0.90 (1.2)	1.45 (1.2)	2.35 (1.1)	3.42 (1.3)	5.70 (1.3)	6.71 (1.3)
1 :	2000	0.74 (1.1)	1.01 (1.4)	1.64 (1.4)	2.65 (1.4)	3.94 (1.4)	5.45 (1.4)
2 :	2000	0.82 (1.2)	1.41 (1.1)	2.16 (1.2)	3.47 (1.2)	5.19 (1.2)	7.24 (1.2)
3 :	2000	0.98 (1.1)	1.64 (1.1)	2.56 (1.2)	4.05 (1.2)	6.23 (1.2)	8.65 (1.2)
6 :	2000	1.42 (1.1)	2.33 (1.1)	3.59 (1.2)	5.80 (1.2)	9.02 (1.2)	12.39 (1.2)
12 :	2000	2.30 (1.1)	3.86 (1.2)	6.28 (1.2)	10.11 (1.2)	15.12 (1.3)	20.56 (1.3)
1 :	5000	2.46 (1.3)	4.93 (1.2)	7.46 (1.3)	11.63 (1.3)	18.05 (1.3)	25.16 (1.4)
2 :	5000	3.23 (1.1)	6.00 (1.1)	9.53 (1.2)	15.40 (1.2)	24.50 (1.2)	34.28 (1.2)
3 :	5000	4.26 (1.1)	7.71 (1.1)	11.92 (1.2)	18.75 (1.3)	29.73 (1.2)	41.46 (1.5)
6 :	5000	6.41 (1.2)	11.88 (1.1)	20.35 (1.2)	29.87 (1.3)	46.71 (1.2)	65.28 (1.5)
12 :	5000	13.09 (1.4)	23.80 (1.6)	41.79 (1.5)	59.10 (1.5)	91.65 (1.4)	124.02 (1.8)
1 :	10000	9.61 (1.2)	17.75 (1.3)	26.22 (1.4)	42.36 (1.3)	71.10 (1.4)	100.87 (1.4)
2 :	10000	11.19 (1.2)	22.23 (1.1)	35.25 (1.2)	58.24 (1.2)	98.01 (1.1)	134.76 (1.2)
3 :	10000	14.84 (1.1)	28.01 (1.2)	44.54 (1.2)	71.65 (1.2)	117.46 (1.2)	170.83 (1.2)
6 :	10000	21.66 (1.5)	43.72 (1.5)	71.85 (1.5)	113.38 (1.6)	186.31 (1.4)	256.98 (1.5)
12 :	10000	55.20 (1.4)	111.74 (1.4)	187.01 (1.5)	309.01 (1.5)	485.73 (1.5)	667.55 (1.5)

Table 3 (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 (2010b).

Average absolute and relative execution times

	$m=3$				$m=4$		$m=5$
$p:n\backslash\tau$	0.010	0.025	0.100	0.200	0.010	0.025	0.010
1 : 100	0.96 (1.4)	1.42 (1.4)	9.86 (1.2)	22.73 (1.2)	5.60 (1.1)	13.99 (1.3)	48.63 (1.0)
2 : 100	1.04 (1.1)	2.23 (1.1)	11.74 (1.1)	28.66 (1.1)	5.17 (1.4)	24.03 (1.1)	41.72 (1.6)
3 : 100	1.58 (1.0)	2.79 (1.0)	15.19 (1.1)	37.12 (1.1)	11.40 (1.0)	34.14 (0.9)	125.90 (1.1)
4 : 100	2.29 (0.9)	3.55 (0.9)	18.23 (1.0)	45.80 (1.1)	25.18 (0.8)	53.39 (0.8)	346.56 (0.8)
6 : 100	4.60 (0.8)	5.40 (0.8)	24.95 (0.9)	61.84 (1.0)	91.55 (0.6)	116.28 (0.6)	1801.58 (0.5)
1 : 200	1.96 (1.3)	5.60 (1.2)	40.07 (1.1)	105.60 (1.1)	21.87 (1.0)	133.23 (1.1)	274.27 (0.9)
2 : 200	2.12 (1.1)	6.38 (1.1)	50.45 (1.1)	139.77 (1.1)	21.00 (1.0)	151.40 (1.0)	265.30 (1.0)
3 : 200	2.77 (1.0)	8.18 (1.0)	67.16 (1.0)	182.39 (1.0)	36.80 (0.9)	225.93 (0.9)	552.52 (0.8)
4 : 200	3.75 (0.9)	10.26 (0.9)	82.73 (1.0)	225.80 (1.0)	55.61 (0.8)	325.67 (0.8)	999.58 (0.7)
6 : 200	6.47 (0.8)	14.52 (0.9)	113.09 (1.0)	311.26 (1.0)	153.91 (0.6)	592.77 (0.7)	3933.44 (0.5)
1 : 300	3.40 (1.2)	10.43 (1.1)	101.18 (1.1)	281.95 (1.0)	60.46 (1.0)	390.83 (1.0)	1059.70 (0.9)
2 : 300	3.88 (1.0)	14.08 (1.0)	130.36 (1.0)	371.95 (1.0)	63.10 (0.9)	587.82 (0.9)	1124.81 (0.9)
3 : 300	5.19 (0.9)	18.53 (0.9)	170.44 (1.0)	496.96 (1.0)	104.67 (0.8)	905.64 (0.8)	2254.24 (0.8)
4 : 300	6.56 (0.9)	22.91 (0.9)	208.12 (1.0)	636.60 (1.0)	157.67 (0.7)	1321.53 (0.8)	4050.45 (0.6)
6 : 300	10.49 (0.8)	32.51 (0.9)	295.13 (1.0)	951.20 (1.0)	321.32 (0.6)	2473.22 (0.7)	16978.51 (0.4)
12 : 300	36.26 (0.7)	69.33 (0.8)	593.24 (0.9)	2154.13 (1.0)			
1 : 400	5.91 (1.1)	22.44 (1.0)	198.14 (1.0)	616.93 (1.0)	138.26 (0.9)	1389.91 (0.9)	3219.40 (0.9)
2 : 400	6.85 (1.0)	27.13 (1.0)	259.47 (1.0)	832.20 (1.0)	155.62 (0.9)	1842.90 (0.9)	3742.53 (0.9)
3 : 400	9.00 (0.9)	35.72 (0.9)	339.13 (0.9)	1138.06 (1.0)	244.52 (0.8)	2896.03 (0.8)	8584.16 (0.7)
4 : 400	11.36 (0.9)	43.23 (0.9)	426.63 (0.9)	1461.29 (1.0)	363.51 (0.7)	4375.24 (0.8)	24130.39 (0.4)
6 : 400	16.34 (0.8)	60.29 (0.9)	609.63 (1.0)	2248.87 (1.0)	669.73 (0.6)	8784.51 (0.7)	
12 : 400	47.52 (0.7)	123.45 (0.8)	1375.59 (0.9)	5411.35 (0.9)			
1 : 500	9.15 (1.0)	32.93 (1.0)	357.53 (1.0)	1186.97 (1.0)	274.41 (0.9)	2867.31 (0.9)	
2 : 500	10.40 (0.9)	43.17 (0.9)	470.28 (0.9)	1634.72 (1.0)	316.32 (0.8)	4727.81 (0.8)	
3 : 500	13.66 (0.9)	56.80 (0.9)	618.37 (0.9)	2285.95 (0.9)	504.06 (0.7)	8213.09 (0.8)	
4 : 500	17.39 (0.8)	70.29 (0.9)	788.91 (0.9)	2969.72 (1.0)	733.00 (0.7)	17159.85 (0.6)	
6 : 500	25.88 (0.8)	99.45 (0.9)	1142.40 (0.9)	4665.41 (1.0)	1407.06 (0.6)	67844.38 (0.4)	
12 : 500	64.65 (0.7)	198.35 (0.8)	2557.46 (0.9)	11624.00 (0.9)			

Table 4 (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 n m-dimensional responses. The numbers in parentheses indicate how many times it is faster than the code from Paindaveine and Šiman (2010b).