Journal of Computational Mathematics, Vol. 24, No. 6, 2006, 771–790.

DATA PREORDERING IN GENERALIZED PAV ALGORITHM FOR MONOTONIC REGRESSION [*1)]

Oleg Burdakov, Anders Grimvall and Oleg Sysoev
(*Department of Mathematics, Linköping University, SE-58183 Linköping, Sweden*)

Abstract

Monotonic regression (MR) is a least distance problem with monotonicity constraints induced by a partially ordered data set of observations. In our recent publication [In Ser. *Nonconvex Optimization and Its Applications*, Springer-Verlag, (2006) **83**, pp. 25-33], the Pool-Adjacent-Violators algorithm (PAV) was generalized from completely to partially ordered data sets (posets). The new algorithm, called GPAV, is characterized by the very low computational complexity, which is of second order in the number of observations. It treats the observations in a consecutive order, and it can follow any arbitrarily chosen topological order of the poset of observations. The GPAV algorithm produces a sufficiently accurate solution to the MR problem, but the accuracy depends on the chosen topological order. Here we prove that there exists a topological order for which the resulted GPAV solution is optimal. Furthermore, we present results of extensive numerical experiments, from which we draw conclusions about the most and the least preferable topological orders.

Mathematics subject classification: 90C20, 90C35, 62G08, 65C60.
Key words: Quadratic programming, Large scale optimization, Least distance problem, Monotonic regression, Partially ordered data set, Pool-adjacent-violators algorithm.

1. Introduction

Consider the monotonic regression (MR) problem for a partially ordered data set of n observations. We denote the vector of observed values by $Y \in R^n$. To express the partial order, we use a directed acyclic graph $G(N, E)$, where $N = \{1, 2, \ldots, n\}$ is a set of nodes and E is a set of edges. Each node is associated with one observation, and each edge is associated with one monotonicity relation as described below. In the MR problem, we must find among all vectors $u \in R^n$, which preserve the monotonicity of the partially ordered data set, the one closest to Y in the least-squares sense. It can be formulated as follows. Given Y, $G(N, E)$ and a strictly positive vector of weights $w \in R^n$, find the vector of fitted values $u^* \in R^n$ that solves the problem:

$$\begin{array}{ll} \min & \sum_{i=1}^{n} w_i (u_i - Y_i)^2 \\ \text{s.t.} & u_i \le u_j \quad \forall (i,j) \in E \end{array} \tag{1.1}$$

Let $\varphi(u)$ and C denote, respectively, the objective function and the feasible region of this problem. Note that (1.1) can be viewed as a problem of minimizing the weighted distance from the vector Y to the convex cone C. Since it is a strictly convex quadratic programming problem, there exists a unique optimal solution u^*. An example of problem (1.1) is considered at the end of Section 2.

[*] Received January 20, 2006.
[1)] This work was supported by the Swedish Research Council.

The MR problem has important statistical applications (see [2, 22]) in physics, chemistry, medicine, biology, environmental science etc. It is present also in operations research (production planning, inventory control etc.) and signal processing. These problems can often be regarded as monotonic data fitting problems (see Section 4). The most challenging of the applied MR problems are characterized by a very large value for n. For such large-scale problems, it is of great practical importance to develop algorithms whose complexity does not rise too rapidly with n.

It is easy to solve problem (1.1) when the constraints have the simple form

$$u_1 \leq u_2 \leq \ldots \leq u_n, \tag{1.2}$$

i.e. when the associated graph $G(N, E)$ is a path. For this special case of complete order, the most efficient and the most widely used algorithm is the Pool-Adjacent-Violators (PAV) algorithm [1, 17, 14]. Its computational complexity is $O(n)$ [11].

The conventional quadratic programming algorithms (see [20]) can be used for solving the general MR problem (1.1) only in the case of small and moderate values of n, up to few hundred. There are some algorithms [2, 22] especially developed for solving this problem. The minimum lower set algorithm [4, 5] is known to be the first algorithm of this kind. It was shown in [3] that this algorithm, being applied to problem (1.1) with the simple constraints (1.2), is of complexity $O(n^2)$, which is worse than the complexity of the PAV algorithm. The existing optimization-based [3, 16, 23] and statistical [18, 19, 24] MR algorithms have either too high computational complexity or too low accuracy of their approximate solutions. The best known complexity of the optimization-based algorithms is $O(n^4)$ [16, 23], which is prohibitive for large n. Perhaps, the most widely used algorithms for solving large-scale applied MR problems are based on simple averaging techniques. They can be easily implemented and have a relatively low computational burden, but the quality of their approximations to u^* is very case-dependent and furthermore, these approximations can be too far from optimal.

In [6, 7], we introduced a new MR algorithm, which can be viewed as a generalization of the Pool-Adjacent-Violators algorithm from the case of a completely ordered (1.2) to partially ordered data set of observations. We call it the GPAV algorithm. It combines both low computational complexity $O(n^2)$ and high accuracy. These properties extend the capabilities of the existing tools to solving very large-scale MR problems. The efficiency of the GPAV algorithm has been demonstrated in [6, 7] on large-scale test problems with obvious superiority over the simple averaging techniques [18, 19, 24]. In [13], it has been used advantageously for solving applied MR problems.

Our algorithm treats the nodes N or, equivalently, the observations in a consecutive order. Any topological order [9] of N is acceptable, but the accuracy of the resulted solution depends on the choice. In this paper, we focus on studying the effect of topological sort on the quality of the solution produced by the GPAV algorithm.

In Section 2, the GPAV algorithm is reformulated for the case when the nodes N are topologically ordered. We prove in Section 3 that, for any MR problem (1.1), there exists a topological order assuring that the GPAV algorithm produces the exact solution to this problem. In Section 4, test problems based on the monotonic data fitting are described. We introduce a wide variety of topological sorts and study their effect on the performance of the GPAV algorithm. Results of extensive numerical experiments are presented and discussed in this section. In Section 5, we draw conclusions about the most and the least preferable topological sorts.

2. GPAV Algorithm for Preordered Data Sets

For presenting this algorithm and discussing its properties, we will use the following definitions and notations. Denote the set of immediate predecessors $\{j \in N : (j,i) \in E\}$ for node $i \in N$ by i^-. The subset of nodes $B \subset N$ is called a *block* if, for any $i, j \in B$, all the nodes in all the paths from i to j belong to B. The block B_i is said to be an *immediate predecessor* for B_j, or *adjacent* to B_j, if there exist $k \in B_i$ and $l \in B_j$ such that $k \in l^-$. Let B_i^- denote the set of all blocks adjacent to block B_i. We associate each block with one of its upper nodes, which is called the *head node*. If i is the head node for some block, we denote this block by B_i. The set of all head nodes is denoted by H. The set of blocks $\{B_i\}_{i \in H}$, where $H \subset N$, is called a *block partitioning* of N if

$$\bigcup_{i \in H} B_i = N$$

and

$$B_i \cap B_j = \emptyset, \quad \forall i \neq j, \quad i, j \in H.$$

Let W_k denote the weight of the block B_k. It is computed by the formula

$$W_k = \sum_{i \in B_k} w_i.$$

For any point $u \in R^n$ feasible in (1.1), the set of nodes N can be uniquely partitioned into disjoint subsets using the following principles:

- if $j \in i^-$ and $u_i = u_j$, then i and j belong to one and the same subset;
- if $u_i \neq u_j$, then the nodes i and j belong to different subsets;
- if there is no path connecting the nodes i and j, they belong to different subsets.

It is easy to show that these subsets are blocks, and for any block B_k,

$$u_i = U_k, \quad \forall i \in B_k,$$

where U_k denotes a common value of the components for this block. The resulting partitioning of the nodes is called the block partitioning induced by u. By this definition, the following strict inequalities hold

$$U_j < U_k, \quad \forall j \in B_k^-. \tag{2.1}$$

This means that the same block partitioning can be obtained by excluding the edges of E which are associated with the non-active monotonicity constraints in (1.1).

The optimality conditions for the MR problem (1.1) are well studied (see e.g. [2, 3, 15, 21, 22]). We reformulate them in a form useful for further consideration as follows.

Theorem 2.1. *The feasible point $u^* \in R^n$ is the optimal solution to problem (1.1) iff, for any block B_k of the block partitioning $\{B_k\}_{k \in H}$ induced by u^*, the corresponding components u_i^*, $i \in B_k$, solve the MR sub-problem*

$$\begin{array}{ll} \min & \sum_{i \in B_k} w_i (u_i - Y_i)^2 \\ \text{s.t.} & u_i \leq u_j \quad \forall i, j \in B_k, \ (i,j) \in E \end{array} \tag{2.2}$$

Proof. The result is a straightforward consequence of the Karush-Kuhn-Tucker conditions [20] applied to problems (1.1) and (2.2).

Consider (2.2) as a separate problem determined by a given set of components B_k. It is known that if the block partitioning induced by the optimal solution to this problem produces the single block B_k, its components u_i^*, $i \in B_k$, have a common value

$$U_k = \frac{\sum_{i \in B_k} w_i Y_i}{W_k} \ . \tag{2.3}$$

The solution $u \in R^n$ generated by the GPAV algorithm is feasible, but not necessarily optimal to the MR problem (1.1). For each block B_k of the block partitioning $\{B_k\}_{k \in H}$ induced by u, the corresponding components u_i, $i \in B_k$, have the common value U_k defined by formula (2.3), but not all the blocks provide optimal solutions to the corresponding MR sub-problems (2.2). Our numerical experiments show that the set of constraints active for the GPAV solution, is almost identical to the optimal set of active constraints. This explains why the GPAV algorithm is able to solve MR problem (1.1) with sufficiently high accuracy.

We assume now that the nodes N of the directed acyclic graph $G(N, E)$ have a topological order. The GPAV algorithm sets initially $B_i = \{i\}$ and $B_i^- = i^-$ for all the nodes $i \in N$, and subsequently operates with the blocks only. It treats the blocks in the order consistent with the topological sort, namely, $B_1, B_2, \ldots, B_n$. For the block which is treated, say B_k, its common value (2.3) is compared with those of its adjacent blocks. While there exists an adjacent violator of the monotonicity, the block B_k absorbs the one with the most severe violation and inherits the list of blocks adjacent to the absorbed one. The common value U_k is updated accordingly.

The outlined GPAV algorithm for solving the MR problem (1.1) with topologically pre-ordered data set can be presented as follows.

Algorithm GPAV. Given: vectors $w, Y \in R^n$ and a directed acyclic graph $G(N, E)$ with topologically ordered nodes.

Set $H = N$.
For all $i \in N$, set $B_i = \{i\}$, $B_i^- = i^-$, $U_i = Y_i$ and $W_i = w_i$.
For $k = 1, 2, \ldots, n$, do:
  While there exists $i \in B_k^-$ such that $U_i \geq U_k$, do:
    Find $j \in B_k^-$ such that $U_j = \max\{U_i : \ i \in B_k^-\}$.
    Set $H = H \setminus \{j\}$.
    Set $B_k^- = B_j^- \cup B_k^- \setminus \{j\}$.
    Set $U_k = (W_k U_k + W_j U_j) \ / \ (W_k + W_j)$.
    Set $B_k = B_k \cup B_j$ and $W_k = W_k + W_j$.
    For all $i \in H$ such that $j \in B_i^-$, set $B_i^- = B_i^- \cup \{k\} \setminus \{j\}$.
For all $k \in H$ and for all $i \in B_k$, set $u_i = U_k$.

We formulate and discuss here some of its basic properties that will be used in the subsequent sections for a closer study of this algorithm.

Note that, at any stage of the algorithm, the head nodes $k \in H$ are such that

$$i \leq k, \quad \forall i \in B_k.$$

In the main body of the while-loop of iteration k, block B_k absorbs block B_j. This can increase, but never decreases, the block common value U_k, because $U_j \geq U_k$ before the absorbing starts.

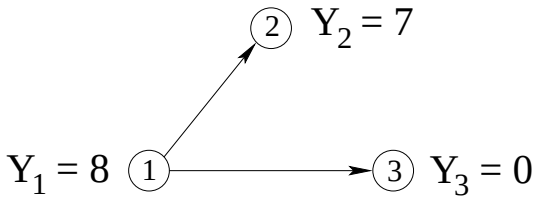

Fig. 2.1. An example of $G(N, E)$ and Y for MR problem (1.1) with $w_1 = w_2 = w_3 = 1$

Consider the current values of $u_1, u_2, \ldots, u_k$, produced at the first k iterations. They are defined by the corresponding block common values. The values of $u_1, u_2, \ldots, u_k$ are monotone; i.e. if one disregards the monotonicity constraints involving the other components, the remaining constraints are satisfied. Moreover, the values of $u_1, u_2, \ldots, u_k$ never increase at subsequent iterations. To show this, note that the current value u_i for $i \leq k$ can change at the iteration $l \geq k$ only when the block B_l absorbs the block B_j that contains the node i. In this case, the value of u_i changes from the block common value U_j to the new one

$$\frac{W_l U_l + W_j U_j}{W_l + W_j} .$$

This can never increase the value of u_i, because $U_j \geq U_l$.

A constraint, which does not change the set of feasible points when it is removed, is said to be redundant. We also call 'redundant' the edges in E associated with the redundant monotonicity constraints. Thus, the edge $(i, j) \in E$ is redundant if there exists an alternative path from i to j. For instance, in the case of the constraints $u_i \leq u_k$, $u_k \leq u_j$ and $u_i \leq u_j$, the last one is redundant and so the edge (i, j). The GPAV algorithm can work correctly in the presence of redundant constraints, but it is more efficient if problem (1.1) is reduced by eliminating such constraints.

One of the most important features of the GPAV algorithm is that it deals with a reduced acyclic graph of blocks, which is initially identical to $G(N, E)$. The graph shrinks whenever one block absorbs another one. This assures the low computational complexity of our algorithm, estimated in [6, 7] as $O(n^2)$. Note that the complexity of any MR algorithm can not be any better than this, because the worst-case estimate for the number of constrains in (1.1) is $O(n^2)$ even with all redundant constraints eliminated.

Let us address to the origin of the inaccuracy of the GPAV algorithm. For this purpose, consider the following MR problem:

$$\begin{aligned} \min \quad & (u_1 - 8)^2 + (u_2 - 7)^2 + (u_3 - 0)^2 \\ \text{s.t.} \quad & u_1 \leq u_2, \ u_1 \leq u_3 \end{aligned} \tag{2.4}$$

This example is illustrated by Fig. 2.1. The depicted acyclic graph $G(N, E)$ admits only two topological orders. For the order $1, 2, 3$, the GPAV solution is

$$u_1 = u_2 = u_3 = 5, \quad \varphi(u) = 38.$$

In the case of the order $1, 3, 2$, the GPAV algorithm produces the solution

$$u_1 = u_3 = 4, \quad u_2 = 7, \quad \varphi(u) = 32.$$

The last one is the optimal solution. It can be obtained by the GPAV algorithm, if after treating node 1, the next is the node among the immediate successors with the smallest value of Y_i.

The same principle assures optimality in the case of a star-type graph with more than two immediate successors of node 1. Although optimality is not guaranteed in the general case of the acyclic graph $G(N, E)$, the described principle proves to be efficient in practice. This is illustrated by the results of our numerical experiments presented in Section 4.

3. Optimal Topological Order

The solution produced by the GPAV algorithm satisfies the monotonicity constraints, but it is not guaranteed to be optimal. We prove here that the nodes in N can be ordered so that the resulted GPAV solution is optimal. For the proof we need a lemma:

Lemma 3.1. *Let the single block $B = N$ be the optimal solution to problem (1.1). Suppose*

$$U^* = \frac{\sum_{i\in N} w_i Y_i}{\sum_{i\in N} w_i}$$

denotes the common value of the optimal block. Then:
(i) *for any topological order of the nodes, the GPAV algorithm produces the same optimal single-block solution;*
(ii) *in the while-loop of iteration k, if the current common value U_k is such that*

$$U_k < U^*, \tag{3.1}$$

then the set B_k^- is not empty;
(iii) *after iteration k, all the current common values U_j with $j \le k$, $j \in H$, satisfy the inequality*

$$U_j \ge U^*. \tag{3.2}$$

Proof. To prove assertion (i), we observe that for any subset of nodes $B_k \subset N$, the value of U_k defined by (2.3) is the unique minimizer of the strictly convex function

$$\varphi_k(U) = \sum_{i\in B_k} w_i (U - Y_i)^2.$$

This means that

$$\varphi_k(U) > \varphi_k(U_k), \qquad \forall U \ne U_k. \tag{3.3}$$

We show that assertion (i) holds by assuming, to the contrary, that the GPAV algorithm can produce a multiple-block partitioning, say $\{B_k\}_{k\in H}$, $H \subset N$. Denote

$$H' = \{k \in H : \quad U_k \ne U^*\}.$$

By our assumption, this set is not empty. Then, applying (3.3) to the optimal value of the objective function in (1.1), we obtain

$$\begin{aligned} \sum_{i=1}^{n} w_i (U^* - Y_i)^2 &= \sum_{k\in H'} \varphi_k(U^*) + \sum_{k\in H\setminus H'} \varphi_k(U^*) \\ &> \sum_{k\in H'} \varphi_k(U_k) + \sum_{k\in H\setminus H'} \varphi_k(U^*) = \sum_{k\in H} \varphi_k(U_k), \end{aligned}$$

where the last term is the value of the same objective function computed for the GPAV solution. This contradicts the assumption that U^* is optimal, and therefore (i) holds.

We will now demonstrate the validity of assertion (iii). Inequality (3.2), which we need to prove, is clearly equivalent to

$$u_i \geq U^*, \quad i = 1, 2, \ldots, k.$$

To show that this inequality holds, it is sufficient to recall that, after the final iteration,

$$u_i = U^*, \quad i = 1, 2, \ldots, n,$$

and that, as noted in Section 2, the current values $u_1, u_2, \ldots, u_k$ produced by the first k iterations never increase at subsequent iterations. This proves (iii).

To prove assertion (ii), it suffices to notice that if, on the contrary, $B_k^- = \emptyset$, then this terminates the while-loop with a value U_k which must satisfy two contradictory inequalities; namely, (3.1) and (3.2).

Let H^* be the the set of head nodes of the block partitioning $\{B_k^*\}_{k \in H^*}$ induced by u^*, the optimal solution to the MR problem (1.1). Denote the corresponding block values by U_k^*, $k \in H^*$. Consider a special order of the nodes which is assumed to assure that

$$U_k^* \leq U_{k'}^*, \quad \forall k, k' \in H^*, \ k < k'. \tag{3.4}$$

Denote the the total number of blocks by h, i.e. $h = |H^*|$. Assumption (3.4) means that the ordering of blocks in ascending order of the block indices

$$k_1 < \ldots < k_i < k_{i+1} < \ldots < k_h, \tag{3.5}$$

produces a topological order for the reduced acyclic graph, in which the nodes and the edges are associated with the blocks and the monotonicity constraints (3.4) respectively. We suppose also that the nodes within each block B_k^* are numbered from $k - |B_k^*| + 1$ to k in the consecutive order consistent with a topological order of the nodes in the graph $G(N, E)$. This implies that $k_{i+1} = k_i + |B_k^*|$ in (3.5).

It is easy to verify that all these assumptions define an order of nodes which is a topological order. We call it an optimal order. It may not be unique, because there may exist different topological orders within each block.

In the example considered at the end of Section 2, the optimal blocks are $B_2^* = \{2\}$ and $B_3^* = \{1, 3\}$. The corresponding block common values $U_2^* = 7$ and $U_3^* = 4$ order the blocks as B_3^*, B_2^*. The topological order of nodes within the block B_3^* is $1, 3$. This provides the following optimal order of nodes: $1, 3, 2$. For the new numbering of nodes induced by the optimal order, (3.4) and (3.5) hold.

Theorem 3.1. *For any optimal order of nodes, the GPAV algorithm produces the optimal solution to the MR problem (1.1).*

Proof. We will prove, by induction in the block indices (3.5), that after iteration $k \in H$, the produced values of $u_1, \ldots, u_k$ are optimal for the MR problem (1.1). This is equivalent to induction in $i = 1, 2, \ldots, h$.

For $i = 1$, i.e. for iteration k_1, the induction assertion follows immediately from Theorem 2.1 and Lemma 3.1.

Suppose that the assertion holds for $i < h$. This means that

$$U_j = U^*_j, \quad j = k_1, \ldots, k_i. \tag{3.6}$$

Let us show that the induction assertion holds for the iteration k_{i+1}. For this purpose, consider the iterations from $k_i + 1$ to k_{i+1}, at which the nodes of the set $B^*_{k_{i+1}}$ are treated by the algorithm. From (2.1) and (3.6), we have the strict inequalities

$$U_j < U^*_{k_{i+1}}, \quad \forall j \in (B^*_{k_{i+1}})^-.$$

Furthermore, by Lemma 3.1, for the current values of U_k and B_k in the while-loop of any iteration $k = k_i + 1, k_i + 2, \ldots, k_{i+1}$, the inequality

$$U^*_{k_{i+1}} \le \max\{U_i : \ i \in B^-_k\}$$

holds whenever $U_k < U^*_{k_{i+1}}$. Thus, when the nodes of the block $B^*_{k_{i+1}}$ are treated by the algorithm, none of the preceding blocks B_j, $j = k_1, \ldots, k_i$, are absorbed in this process and therefore the values of $u_1, \ldots, u_{k_i}$ remain optimal. This means that the algorithm works at these iterations in the same way as it would work if applied to solving the reduced MR problem defined by $B^*_{k_{i+1}}$, similar to (2.2). Theorem 2.1 and Lemma 3.1 assure that $B_{k_{i+1}} = B^*_{k_{i+1}}$, with the optimal common value $U^*_{k_{i+1}}$ resulting from iteration k_{i+1}. This completes our proof by induction and also the proof of the theorem, which follows from the case $i = h$.

It has been shown above that any optimal order can be obtained by ordering the optimal blocks in ascending order of the block common values with subsequent choice of a topological order of nodes within each block. It is interesting to know that such order exists, although to find an optimal order may be as difficult as to solve the MR problem.

4. Numerical Results

Our test problems are based on the monotonic data fitting which is one of the most common types of applied MR problems. In the monotonic data fitting, it is assumed that there exists an unknown response function $y(x)$ of p explanatory variables $x \in R^p$. It is supposed to be monotonic in the sense that

$$y(x') \le y(x''), \quad \forall x' \preceq x'', \quad x', x'' \in R^n,$$

where $\preceq$ is a component-wise $\le$-type relation. Instead of the function $y(x)$, we have available a data set of n observed explanatory variables

$$X_i \in R^p, \quad i = 1, \ldots, n,$$

and the corresponding responses

$$Y_i \in R^1, \quad i = 1, \ldots, n.$$

The function and the data set are related as follows

$$Y_i = y(X_i) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{4.1}$$

where ε_i is an observation error. In general, $X_i \preceq X_j$ does not imply $Y_i \le Y_j$, because of this error.

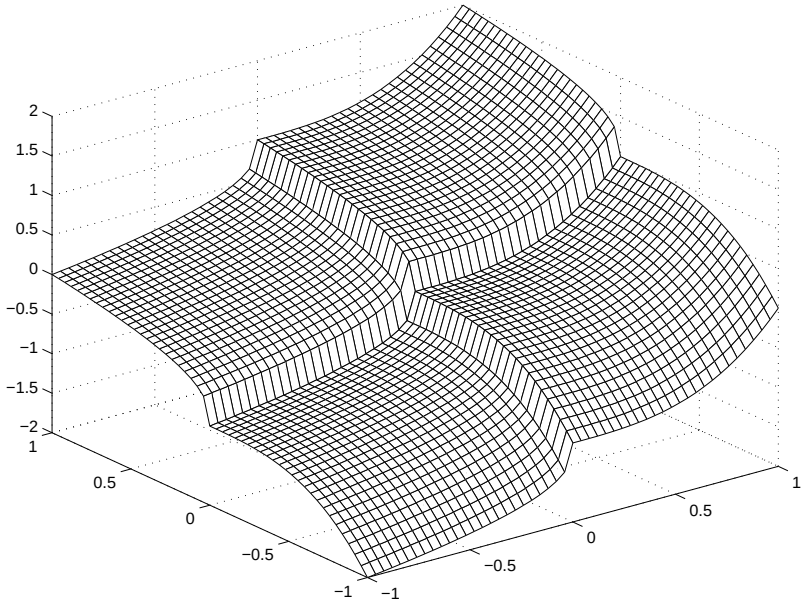

Fig. 4.1. Nonlinear function $y(x)$ defined by (4.2) and (4.3)

The relation $\preceq$ induces a partial order on the set $\{X_i\}_{i=1}^n$. The order can be presented by a directed acyclic graph $G(N, E)$ such that node $i \in N$ corresponds to the i-th observation. This graph is unique if all redundant relations are eliminated.

In monotonic data fitting, one must construct a monotonic response surface model $u(x)$ whose values $u(X_i)$ are as close as possible to the observed responses Y_i for all $i = 1, \ldots, n$. Denoting

$$u_i = u(X_i), \quad i = 1, \ldots, n$$

and using the sum of squares as the distance function, one can reformulate this problem as the MR problem (1.1). In the numerical experiments, we set the weights $w_i = 1$ for all $i = 1, \ldots, n$.

For our test problems, we use two types of functions $y(x)$ of two explanatory variables ($p = 2$); namely, linear and nonlinear. Our nonlinear function is given by the formula

$$y(x) = f(x_1) - f(-x_2), \tag{4.2}$$

where

$$f(t) = \begin{cases} \sqrt[3]{t}, & t \le 0, \\ t^3, & t > 0. \end{cases} \tag{4.3}$$

This function is shown in Fig. 4.1.

Our choice of the linear test problems is inspired by the observation that the optimal values u_i^* that correspond to a local area of values of x depend mostly on the local behavior of the response function $y(x)$ and on the values of the observation errors in this area. Due to the block structure of u^*, these local values of u_i^* typically do not change if to perturb the function values $y(x)$ in distant areas. Therefore, we assume that the local behavior can be well imitated by linear local models.

For the linear function, we consider

$$y(x) = \alpha_1 x_1 + \alpha_2 x_2, \tag{4.4}$$

with the following four possible combinations of the coefficients:

$$\alpha = (0,\, 0),\ (1,\, 0.1),\ (0.1,\, 1),\ (1,\, 1),$$

where $\alpha = (\alpha_1, \alpha_2)$. The rationale behind studying these combinations is the following. The choice $\alpha = (0, 0)$ simulates the behavior of the unknown response function in the local areas where it is almost flat. The choices $\alpha = (1, 0.1)$ and $\alpha = (0.1, 1)$ correspond to the areas where it increases much faster in one explanatory variable than in the other one. The choice $\alpha = (1, 1)$ imitates the cases in which the response function increases in one explanatory variable as rapidly as in the other. The nonlinear function combines the considered types of behavior. In addition, depending on the side from which x_1 or x_2 approaches zero, the function value changes either sharply, or remarkably slowly.

For the numerical experiments, samples of $n = 10^2$, $n = 10^3$ and $n = 10^4$ observations $\{X_i\}_{i=1}^n$ were generated for two types of distribution of the explanatory variables. One was the normal distribution with mean zero and the identity covariance matrix. The other was the independent uniform distribution of the explanatory variables in the interval $[-2, 2]$. The error terms ε_i in (4.1) were independent and identically distributed with mean zero and variance one. Two different types of distributions were used for generating the error terms, namely, normal and double-exponential, which are known as light-tailed and heavy-tailed distributions, respectively.

We call *preprocessing* the stage at which MR problem (1.1) is generated. The observed explanatory variables $\{X_i\}_{i=1}^n$ are involved in formulating this problem implicitly; namely, via the partial order intrinsic in this data set of vectors. Given $\{X_i\}_{i=1}^n$, we generate a directed acyclic graph $G(N, E)$ with all the redundant edges removed. The adjacency-matrix representation [9] is used for the graph. The preprocessing is accomplished by a topological sorting of the nodes N.

In our experiments, we study the following sortings, for which it is not difficult to show that they produce a topological sort of the nodes.

- The vectors $\{X_i\}_{i=1}^n$ are sorted in ascending order of their first components, and then if equal, in ascending order of their second components etc. MATLAB function SORTROWS implements this sorting. (1stComp)

- The vectors $\{X_i\}_{i=1}^n$ are sorted p times in ascending order of their first components, then second components and so on. If the same components of two different vectors are equal, they get the same ordinal number. This produces a specific ordinal number for each component of the vectors $\{X_i\}_{i=1}^n$. Then for each vector X_i, the sum of the ordinal numbers of its components is assigned to the node i. The nodes in N are sorted in ascending order of the sums. (SumOrd)

- For each vector X_i, the sum of its components is assigned to the corresponding node i. The nodes in N are sorted in ascending order of the sums. (SumComp)

- The nodes in N are sorted in ascending order of the number of their predecessors. (NumPred)

- The nodes in N are sorted in descending order of the number of their successors. (NumSucc)

- Set $\mathcal{N} = N$. While $\mathcal{N}$ is not empty, find one of the lower nodes in $\mathcal{N}$, say i, with the smallest value Y_i and exclude i from $\mathcal{N}$. The nodes in N are sorted in the order in which they have been excluded from $\mathcal{N}$. (MinVal)

- Set $\mathcal{N} = N$. While $\mathcal{N}$ is not empty, find all the lower nodes in $\mathcal{N}$ and exclude them from $\mathcal{N}$ in ascending order of the values Y_i. The nodes in N are sorted in the order in which they have been excluded from $\mathcal{N}$. (Hasse1)

- Set $\mathcal{N} = N$. While $\mathcal{N}$ is not empty, find all the upper nodes in $\mathcal{N}$ and exclude them from $\mathcal{N}$ in descending order of the values Y_i. The nodes in N are sorted in reverse of the order in which they have been excluded from $\mathcal{N}$. (Hasse2)

The advantages of involving the smallest values of Y_i in topological sorts were discussed at the end of Section 2. This advantage is exploited in (MinVal), (Hasse1) and (Hasse2).

The sorts (Hasse1) and (Hasse2) are related to Hasse diagrams drawn for posets [10]. The same two sorts, but without sorting in the values Y_i, were used in [8] for studying how rapidly the mean square difference between the expected response values $y(x_i)$ and the values u_i fitted by GPAV tends to zero, and how quickly the mean square residual

$$\sum_{i=1}^{n}(u_i - Y_i)^2/n$$

approaches the true variance of the random error, as the number of observations n increases. In [6, 7], we used (NumPred) for showing the advantages of the GPAV algorithm over the popular simple averaging algorithm.

The consecutive order in which the observations $\{X_i, Y_i\}_{i=1}^n$ are treated by the GPAV algorithm is determined by the topological sort applied. This order determines the propagation of the set of points $X_1, \ldots, X_i$ as i increases from 1 to n. We call it *set propagation*. In Figures 4.2 and 4.3, we plot, respectively, normally and uniformly distributed points X_i for $n = 10^4$. The visible strips are formed by four groups of equal numbers of points. The groups correspond to the following intervals for i:

$$\left[1, \frac{1}{4}n\right], \quad \left(\frac{1}{4}n, \frac{1}{2}n\right], \quad \left(\frac{1}{2}n, \frac{3}{4}n\right], \quad \left(\frac{3}{4}n, n\right],$$

where i is the ordinal number in the considered topological order. The strips illustrate the set propagation. Note that the sorts (1stComp), (SumOrd), (SumComp), (NumPred), (NumPred) do not depend on the values of Y_i. To generate these values for the other sorts, we used the linear function $y(x) = x_1 + x_2$ with two different types of error distribution, normal and double-exponential. When the sort (Hasse1) is applied, the pictures are very similar for both distributions. The same holds for (Hasse2). The difference caused by the different error distributions is more evident for (MinVal). Therefore, in Figures 4.2 and 4.3 we preset the normal error distribution case for all the sortings, while the double-exponential case is presented in Figure 4.4 for (MinVal) only. Notice that the (MinVal) strips have zig-zagging boundaries for both types of error distribution. In each left lower corner of these boundaries, one can find a point X_i with a large value of Y_i, so that GPAV treats this point only after treating the points to the left of the boundary line going up from X_i and those below the line going from X_i to the right. Since the double-exponential distribution is heavy-tailed, it has fewer such corner points compared with the normal distribution. The form of the data cloud is different for the normal

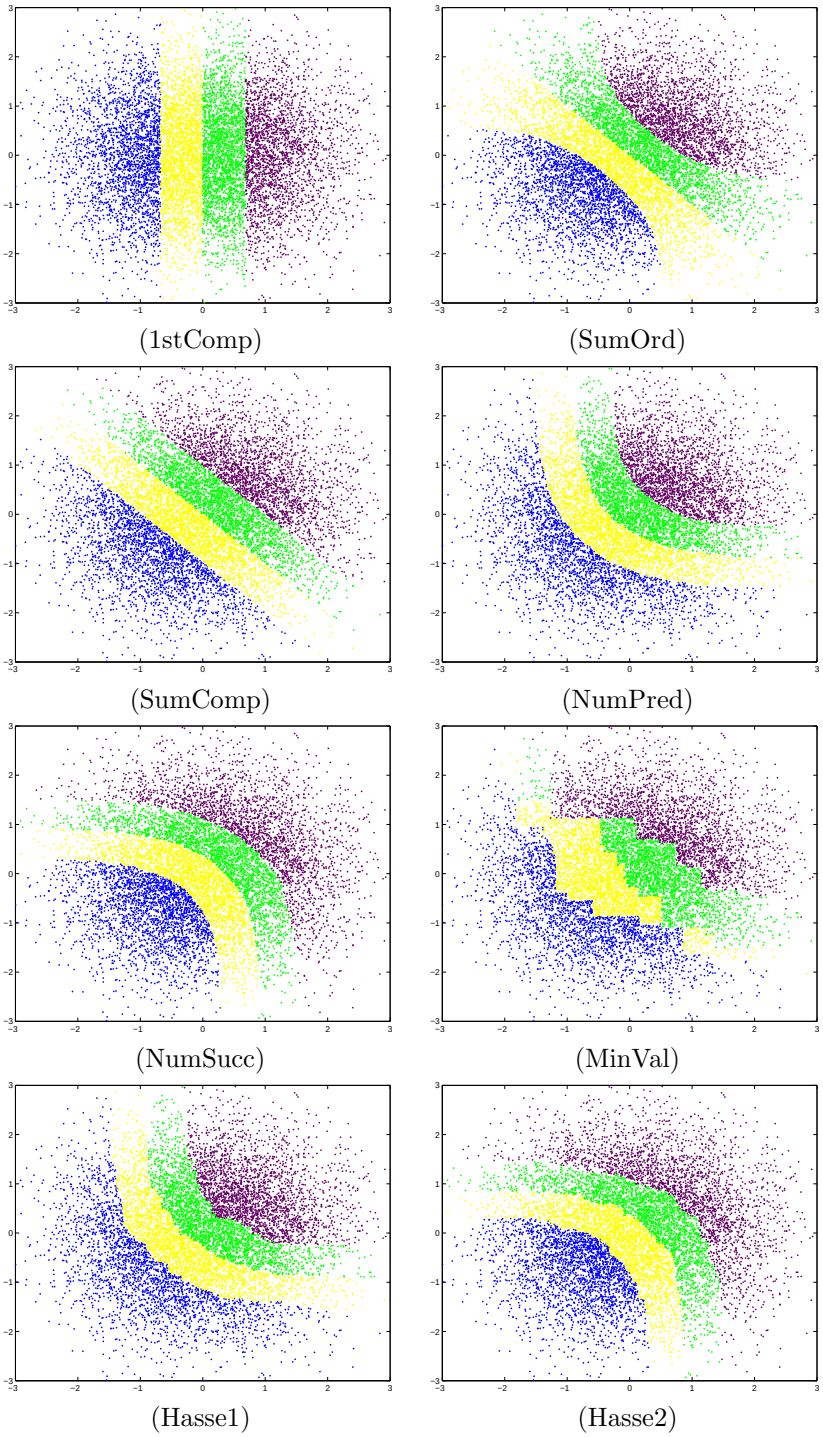

Fig. 4.2. Set propagation for normally distributed observed points X_i, normal error

and uniform distributions. It is round and square respectively for these two distributions. This affects the strip shape for some of the sortings, like (SumOrd). Figures 4.2–4.4 will help us to understand why some of the sortings considered are preferable for the GPAV algorithm to the

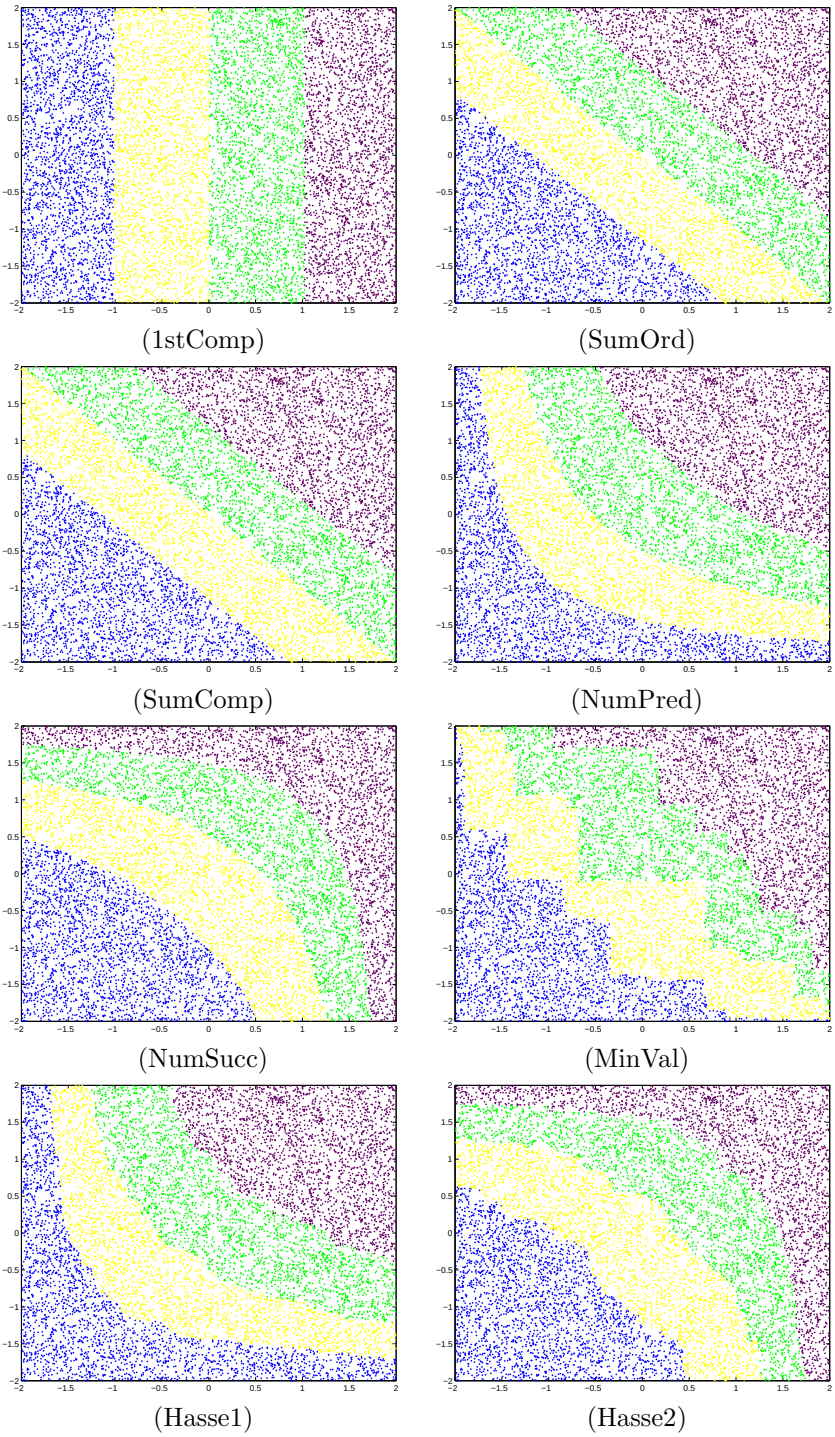

Fig. 4.3. Set propagation for uniformly distributed points X_i, normal error

others.

We used MATLAB for implementing our algorithms. The numerical results presented here were obtained on a PC running under Windows XP with a Pentium 4 processor (2.8 GHz, 1GB

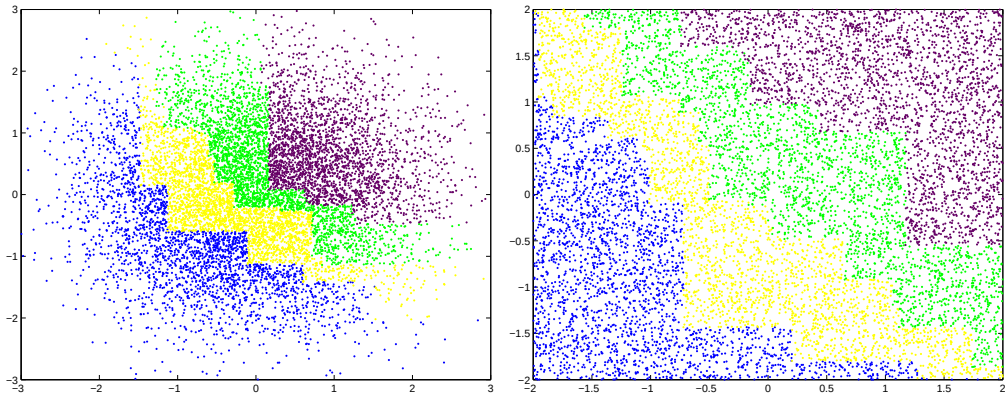

Fig. 4.4. Set propagation for (MinVal), normally (left) and uniformly (right) distributed points X_i, double-exponential error

RAM). We evaluate performance of the presented sortings by comparing the relative error

$$e(u) = \frac{\varphi(u) - \varphi(u^*)}{\varphi(u^*)},$$

where $\varphi(u)$ is the objective function value obtained by GPAV for the given sort, and $\varphi(u^*)$ is the optimal value obtained by MATLAB solver LSQLIN. Recalling that u^* is the orthogonal projection of Y on the convex set C and using a well known projection property (see, e.g., [12]), we observe that

$$\|u - u^*\|^2 \leq \|u - Y\|^2 - \|u^* - Y\|^2.$$

Since $\varphi(u) = \|u - Y\|^2$, the relative error $e(u)$ provides an upper estimate for the relative distance between the approximate solution u and the optimal solution u^*.

Because LSQLIN failed to solve any of the test problems for $n = 10^3$ and $n = 10^4$ within ten hours, in these large-scale cases the performance evaluation was based on the following relative deviation from the best result

$$d(u) = \frac{\varphi(u) - \varphi(u_{\text{best}})}{\varphi(u_{\text{best}})},$$

where $\varphi(u_{\text{best}})$ is the smallest among the eight objective function values obtained by GPAV with the use of the considered sortings for the same MR problem.

Tables 4.1 and 4.2 summarize the performance data obtained for normal and for uniform distribution of the explanatory variables. We consider the five functions $y(x)$, namely, four linear (4.4) and one nonlinear (4.2)-(4.3). For each function, the left and the right columns refer respectively to the normal and the double-exponential distributions of errors. For $n = 10^2$ and $n = 10^3$, we present the average value of $d(u) \cdot 100\%$ for 100 MR problems generated by each combination of the considered distributions of the explanatory variables and the error. The average value of $e(u) \cdot 100\%$ is calculated only in the case of $n = 10^2$, for the reasons discussed above. One can see that the average values of $d(u) \cdot 100\%$ are sufficiently close to those of $e(u) \cdot 100\%$. For $n = 10^4$, we report the results of solving only one MR problem for each combination of the distributions, because it takes about 3-4 minutes of CPU time to solve one such large-scale problem. Therefore, our main analysis will be focused on the results of the most extensive numerical experiments that correspond to $n = 10^2$ and $n = 10^3$. The results corresponding to $n = 10^4$ serve to illustrate our main conclusions.

Table 4.1: **Relative error and relative deviation for normally distributed explanatory variables**

average $e(u) \cdot 100\%$ for $n = 10^2$

sort	$\alpha = (0,0)$		$\alpha = (1,0.1)$		$\alpha = (0.1,1)$		$\alpha = (1,1)$		nonlinear	
(1stComp)	2.63	2.85	3.31	3.42	13.57	14.83	8.60	9.19	7.63	7.30
(SumOrd)	0.58	0.63	2.55	2.25	2.42	2.12	1.34	1.22	1.15	1.58
(SumComp)	0.56	0.64	2.35	1.95	2.36	2.09	1.22	1.25	1.33	1.37
(NumPred)	0.79	0.71	2.96	2.58	3.05	2.88	2.19	1.57	2.04	2.50
(NumSucc)	0.79	0.92	2.52	2.06	2.74	2.40	1.80	1.54	1.35	1.40
(MinVal)	0.62	0.68	0.87	0.88	1.02	0.71	0.77	0.72	0.71	0.74
(Hasse1)	0.56	0.50	2.41	1.94	2.11	1.69	0.90	0.83	1.09	1.42
(Hasse2)	0.75	0.89	2.49	2.25	2.46	2.17	1.62	1.43	0.89	0.90

average $d(u) \cdot 100\%$ for $n = 10^2$

(1stComp)	2.45	2.69	2.87	3.02	13.00	14.41	8.36	8.95	7.48	7.10
(SumOrd)	0.41	0.48	2.11	1.86	1.92	1.73	1.12	1.00	1.00	1.39
(SumComp)	0.39	0.48	1.91	1.56	1.85	1.70	0.99	1.02	1.19	1.18
(NumPred)	0.62	0.55	2.52	2.18	2.54	2.50	1.96	1.34	1.89	2.31
(NumSucc)	0.62	0.76	2.08	1.66	2.23	2.01	1.58	1.32	1.21	1.20
(MinVal)	0.45	0.52	0.44	0.49	0.53	0.33	0.54	0.50	0.57	0.55
(Hasse1)	0.39	0.35	1.97	1.55	1.61	1.31	0.67	0.61	0.94	1.22
(Hasse2)	0.58	0.73	2.05	1.86	1.96	1.78	1.40	1.21	0.74	0.71

average $d(u) \cdot 100\%$ for $n = 10^3$

(1stComp)	1.52	1.28	3.46	2.74	48.73	50.74	33.59	33.93	29.11	30.23
(SumOrd)	0.21	0.19	1.27	1.24	1.31	1.03	0.50	0.41	0.92	0.89
(SumComp)	0.02	0.03	0.70	0.68	0.81	0.53	0.03	0.02	0.62	0.55
(NumPred)	0.28	0.19	1.97	1.82	2.42	1.93	1.27	1.15	3.03	2.76
(NumSucc)	0.21	0.23	2.26	1.90	2.37	2.09	1.31	1.04	0.21	0.24
(MinVal)	0.42	0.34	0.28	0.41	0.18	0.48	1.09	1.09	1.12	1.25
(Hasse1)	0.24	0.15	2.29	2.01	2.60	1.82	0.97	0.86	2.47	2.37
(Hasse2)	0.28	0.33	2.72	2.18	2.81	2.33	1.53	1.21	0.16	0.14

$d(u) \cdot 100\%$ for $n = 10^4$

(1stComp)	0.55	0.41	0.75	0.44	75.37	85.53	71.12	74.42	56.80	71.79
(SumOrd)	0.10	0.09	2.48	0.86	3.38	1.62	0.75	0.82	2.87	3.03
(SumComp)	0.00	0.00	0.85	0.00	2.05	0.00	0.00	0.00	1.12	1.11
(NumPred)	0.08	0.09	3.28	1.91	3.91	1.63	1.59	1.49	7.43	7.76
(NumSucc)	0.11	0.05	4.03	2.51	5.29	2.14	1.63	1.45	0.00	0.00
(MinVal)	0.20	0.23	0.00	1.00	0.00	1.05	1.95	3.25	2.07	2.55
(Hasse1)	0.08	0.08	3.32	3.38	5.09	1.82	1.66	1.46	7.81	6.72
(Hasse2)	0.18	0.05	4.26	2.38	5.46	2.68	2.26	2.55	0.16	0.30

In Table 4.3, one can see how the average CPU time for the preprocessing and for running the GPAV algorithm increases with n for normally distributed explanatory variables. The CPU time for uniformly distributed explanatory variables is very similar. Since the rate of increase

Table 4.2: **Relative error and relative deviation for uniformly distributed explanatory variables**

average $e(u) \cdot 100\%$ for $n = 10^2$

sort	$\alpha = (0,0)$		$\alpha = (1,0.1)$		$\alpha = (0.1,1)$		$\alpha = (1,1)$		nonlinear	
(1stComp)	3.34	2.94	3.50	3.33	14.98	15.69	7.12	9.26	7.58	7.41
(SumOrd)	0.68	0.84	2.90	2.20	2.40	2.62	1.49	1.41	1.35	1.57
(SumComp)	0.76	0.79	2.65	2.22	2.34	2.53	1.48	1.38	1.34	1.68
(NumPred)	0.88	0.95	3.13	2.77	3.69	3.72	1.97	2.24	2.49	2.55
(NumSucc)	0.83	0.96	2.92	2.20	2.74	2.95	2.30	2.32	1.29	1.44
(MinVal)	0.72	0.66	1.21	0.91	0.70	0.66	0.46	0.60	0.64	0.62
(Hasse1)	0.62	0.58	2.03	2.14	1.86	2.08	1.01	0.78	1.16	1.50
(Hasse2)	0.85	1.11	2.88	2.54	2.19	2.75	2.00	2.14	0.89	0.96

average $d(u) \cdot 100\%$ for $n = 10^2$

(1stComp)	3.14	2.73	2.90	2.93	14.46	15.20	6.90	9.07	7.42	7.25
(SumOrd)	0.49	0.64	2.29	1.80	1.94	2.19	1.28	1.24	1.20	1.42
(SumComp)	0.57	0.59	2.04	1.82	1.88	2.10	1.26	1.21	1.19	1.53
(NumPred)	0.68	0.75	2.52	2.37	3.21	3.28	1.75	2.06	2.34	2.40
(NumSucc)	0.64	0.76	2.31	1.80	2.27	2.52	2.09	2.15	1.14	1.29
(MinVal)	0.52	0.46	0.61	0.52	0.23	0.24	0.25	0.43	0.49	0.47
(Hasse1)	0.43	0.38	1.43	1.74	1.40	1.65	0.79	0.60	1.00	1.35
(Hasse2)	0.65	0.91	2.27	2.14	1.72	2.32	1.79	1.96	0.74	0.81

average $d(u) \cdot 100\%$ for $n = 10^3$

(1stComp)	1.29	1.25	3.70	3.48	54.28	63.15	35.69	39.04	27.91	30.28
(SumOrd)	0.15	0.15	1.25	1.05	1.35	0.97	0.09	0.10	1.57	1.41
(SumComp)	0.14	0.15	1.31	1.02	1.34	0.96	0.09	0.10	1.57	1.43
(NumPred)	0.17	0.15	2.71	2.08	2.99	2.45	1.31	1.17	4.68	4.59
(NumSucc)	0.14	0.15	2.53	2.31	2.83	2.35	1.52	1.29	0.22	0.24
(MinVal)	0.34	0.30	0.11	0.23	0.12	0.30	0.78	0.71	1.00	1.01
(Hasse1)	0.14	0.09	2.98	2.61	2.81	2.25	1.12	0.97	3.66	3.82
(Hasse2)	0.24	0.23	2.93	2.81	3.46	2.84	1.81	1.49	0.22	0.14

$d(u) \cdot 100\%$ for $n = 10^4$

(1stComp)	0.40	0.24	1.70	1.12	108.51	117.33	95.32	105.58	82.04	113.01
(SumOrd)	0.04	0.02	1.34	0.43	1.72	0.62	0.00	0.00	3.39	1.76
(SumComp)	0.03	0.06	1.37	0.53	1.41	0.32	0.02	0.00	3.37	1.70
(NumPred)	0.02	0.00	4.22	1.61	4.57	1.57	1.47	1.42	10.58	10.14
(NumSucc)	0.00	0.02	3.16	2.63	4.80	2.55	1.81	1.42	0.00	0.00
(MinVal)	0.10	0.06	0.00	0.00	0.00	0.00	1.82	2.22	2.00	5.79
(Hasse1)	0.02	0.02	4.78	3.24	5.05	3.20	1.78	2.32	10.55	9.43
(Hasse2)	0.04	0.07	5.24	3.49	5.73	1.96	2.83	1.92	0.17	0.41

was almost the same for all types of error distributions and functions considered, we present here the computational time for $y(x) = x_1 + x_2$ and normally distributed errors. The CPU time of LSQLIN is available for $n = 10^2$ only; it was around 0.9 seconds. One can observe that,

Table 4.3: **CPU time (seconds) of running preprocessing and GPAV algorithm**

algorithm	$n = 10^2$	$n = 10^3$	$n = 10^4$
(1stComp)	0.011	0.373	50.3
(SumOrd)	0.012	0.388	50.5
(SumComp)	0.012	0.399	50.7
(NumPred)	0.013	0.399	51.4
(NumSucc)	0.019	0.834	130.9
(MinVal)	0.020	0.714	94.7
(Hasse1)	0.013	0.706	113.5
(Hasse2)	0.016	0.787	125.3
GPAV	0.019	0.596	108.2

for each value of n, the preprocessing time does not differ much for the sortings considered. Moreover, the time for the preprocessing and for running the GPAV algorithm have comparable values. For each sorting, the ratio between these two times changes with n moderately. For GPAV, the observed growth of time with n is in a good agreement with the $O(n^2)$ estimate of its computational complexity.

It should be emphasized that, in our numerical experiments, the variance of the error ε_i is comparable with the function values $y(X_i)$. Such observations, with a high level of noise, were deliberately chosen for checking the performance of the GPAV algorithm in this difficult case.

The results presented in Tables 4.1 and 4.2 indicate that all the sortings, except (1stComp), assure a fairly good accuracy of the GPAV solution. We discuss below the reasons why (1stComp) is such an unfavorable sorting for our algorithm. The problems generated on the base of the linear function with $\alpha = (0, 0)$ were the easiest for all the sortings including (1stComp). Therefore, we will focus our analysis on the results obtained on the basis of the other functions.

Obviously, the most difficult were linear problems with $\alpha = (1, 0.1)$ and $\alpha = (0.1, 1)$. Note that (1stComp) provides the best accuracy for $\alpha = (1, 0.1)$, while its worst performance corresponds to $\alpha = (0.1, 1)$. We tested even more difficult problems defined by $\alpha = (1, 0)$ and $\alpha = (0, 1)$. They can be viewed as pathological, because the cases, in which y does not depend on some of the explanatory variables, are not typical for the applied MR problems. The corresponding results are not reported in the tables. It is sufficient to mention here that the best results of (1stComp) refer to $\alpha = (1, 0)$. They were even better than those obtained by the sortings (SumOrd), (SumComp), (NumPred) and (NumSucc) which, like (1stComp), do not depend on the values of Y_i. The other sortings are discussed below. Figures 4.2 and 4.3 show that the set propagation for (1stComp) is identical, in the case of $\alpha = (1, 0)$, to the propagation of the set

$$\{X_j : \ y(X_j) \le c, \ 1 \le j \le n\} \tag{4.5}$$

as c increases. They are similar when $\alpha = (1, 0.1)$. On the other hand, these two sets propagate in almost orthogonal directions when $\alpha = (0.1, 1)$. This allows us to put forward a hypothesis that the more similarities there are between the set propagation of a sorting and the level sets of the function $y(x)$, the better the accuracy is provided by the sorting. The set propagation of (1stComp), in comparison to the other sortings, is much more different from the level sets of the linear function with $\alpha = (0.1, 1)$. This explains why (1stComp) shows the worst results in this case. Again, the set propagation for (SumComp) is identical to the level sets of the linear function with $\alpha = (1, 1)$. The best accuracy among the five sorts mentioned is clearly attributed to (SumComp) in this case and this supports the hypothesis. Moreover, for normally distributed

Table 4.4: **Distribution of the relative deviation for (SumComp) and (MinVal),** $n = 10^3$

linear function, $\alpha = (0, 0.1)$

error	sort	normal X_i				uniform X_i			
		0-1%	0-3%	av.	max	0-1%	0-3%	av.	max
norm.	(SumComp)	64	99	0.81	3.49	39	98	1.34	3.05
norm.	(MinVal)	95	99	0.18	3.83	96	100	0.12	2.78
d-exp.	(SumComp)	77	100	0.53	2.50	56	98	0.96	3.08
d-exp.	(MinVal)	83	96	0.48	3.86	89	99	0.30	7.60

linear function, $\alpha = (1, 1)$

norm.	(SumComp)	100	100	0.03	0.57	100	100	0.09	0.54
norm.	(MinVal)	49	100	1.09	2.75	66	100	0.78	2.70
d-exp.	(SumComp)	100	100	0.02	0.20	99	100	0.10	1.04
d-exp.	(MinVal)	52	97	1.09	3.68	71	100	0.71	2.79

nonlinear function

norm.	(SumComp)	80	100	0.62	2.26	27	95	1.57	3.89
norm.	(MinVal)	54	93	1.12	3.65	57	94	1.00	4.97
d-exp.	(SumComp)	82	100	0.55	2.33	31	98	1.43	3.61
d-exp.	(MinVal)	47	93	1.25	4.18	57	96	1.01	6.31

explanatory variables, the set propagation of (SumOrd) differs noticeably from the propagation of the set (4.5) when $\alpha = (1, 1)$, while they are identical for the uniform distribution. This explains why (SumOrd) shows a better performance in the last case.

(MinVal) resembles the propagation of the level sets, especially in the case of normally distributed errors. This is one of the reasons why this sorting usually provides the best accuracy, and also why it performs better for this error distribution than in the double-exponential case. The other reason is that (MinVal) is based on the idea presented at the end of Section 2 that suggests treating next the observation with the smallest value of Y_i among the observation admitted by GPAV for treating next. This strategy offers an extra opportunity for obtaining a higher accuracy.

The same basic idea is employed in sortings (Hasse1) and (Hasse2). This assures that these sortings, as well as sorting (MinVal), provide an optimal topological order in the MR problem presented by Figure 2.1.

In support of this strategy, it is necessary to mention that we compared sortings (Hasse1) and (Hasse2) with essentially the same sortings, also based on the Hasse diagrams, but which were not using the values of Y_i. Sortings (Hasse1) and (Hasse2) performed much better due to the introduced strategy. We do not report the details of this comparative analysis here.

Considering how the two different distributions of errors affect the performance of all the sortings, we notice that (SumOrd), (NumPred) and (Hasse1) are among those least sensitive to the presence of outliers produced by the double-exponential distribution of errors.

Obviously, (SumComp) and (MinVal) are among the best sortings in our numerical experiments. It should be mentioned that (Hasse2) is among the best, mainly for the nonlinear

function. Since Tables 4.1 and 4.2 present average values only, let us take a closer look, for (MinVal) and (SumComp), at the variation of their relative deviation. This is presented in Table 4.4. For each combination of function, distribution of errors and distribution of explanatory variables, we report in this table the number of problems, of 100 in total, which have been solved with relative deviation below 1% and below 3%. To facilitate the analysis, we reproduce in column 'av.' the average value of $d(u) \cdot 100\%$ from Tables 4.1 and 4.2. Note that the overwhelming majority of the generated MR problems have been solved by GPAV with the relative deviation below 3%. In column 'max', the maximal value of $d(u) \cdot 100\%$ is reported. Comparing this value for (SumComp) and (MinVal), one can observe that the first sorting is less sensitive to the outliers produced by the double-exponential distribution of errors. One more attractive feature of (SumComp) is that it is easier to implement this sorting than (MinVal).

5. Conclusions

We have introduced several sortings which can be used at the preprocessing stage. Let us summarize here the observations presented in the previous section. The GPAV algorithm has been shown to be efficient in term of the computational time, which was below four minutes for solving large-scale MR problems with 10^4 observations. In our extensive numerical experiments (of about $5 \cdot 10^4$ in total), the algorithm has produced high accuracy solutions for all the considered sortings, except (1stComp). If this unfavorable sort is disregarded, the average relative error and the average deviation from the best of the solution obtained was typically below 3%. The main conclusion is that all the sortings considered here, except (1stComp), can be recommended for the data preordering. A slight preference may be given to sortings (SumComp) and (MinVal).

We observed also that a sorting provides good accuracy when its set propagation follows in a certain sense, or at least, does not differ too much from the propagation of the level sets of the response function.

Moreover, the accuracy of the GPAV solution improves when the observed values Y_i are properly taken into account. The most successful was the strategy in which the observation chosen for treatment at the current iteration has the smallest observed value among the possible candidates for treatment.

References

[1] M. Ayer, H.D. Brunk, G.M. Ewing, W.T. Reid, E. Silverman, An empirical distribution function for sampling with incomplete information, *The Annals of Mathematical Statistics*, **26** (1955), 641–647.

[2] R.E. Barlow, D.J. Bartholomew, J.M. Bremner, H.D. Brunk, Statistical inference under order restrictions, Wiley, New York, 1972.

[3] M.J. Best, N. Chakravarti, Active set algorithms for isotonic regression: a unifying framework, *Mathematical Programming*, **47** (1990), 425–439.

[4] H.D. Brunk, Maximum likelihood estimates of monotone parameters, *The Annals of Mathematical Statistics*, **26** (1955), 607–616.

[5] H.D. Brunk, G.M. Ewing, W.R. Utz, Minimizing integrals in certain classes of monotone functions, *Pacific J. Math.*, **7** (1957), 833–847.

[6] O. Burdakov, O. Sysoev, A. Grimvall, M. Hussian, An algorithm for isotonic regression problems, The Proceedings of the 4th European Congress of Computational Methods in Applied Science and Engineering 'ECCOMAS 2004', Ed. P. Neittaanmaki *et al.*, 2004.

[7] O. Burdakov, O. Sysoev, A. Grimvall, M. Hussian, An $O(n^2)$ algorithm for isotonic regression problems, Large Scale Nonlinear Optimization, Ed. G. Di Pillo, M. Roma, Springer-Verlag, 2006, 25–33.

[8] O. Burdakov, A. Grimvall, M. Hussian, O. Sysoev, Hasse diagrams and the generalized PAV algorithm for monotonic regression in several explanatory variables, submitted to *Computational Statistics & Data Analysis*.

[9] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms (Second Edition), MIT Press, Cambridge, 2001.

[10] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge Univ. Press, Cambridge, 2002.

[11] S.J. Grotzinger, C. Witzgall, Projection onto order simplexes, *Applications of Mathematics and Optimization*, **12** (1984), 247–270.

[12] J.-B. Hiriart-Urruty, C. Lemarechal, Convex Analysis and Minimization Algorithms, Springer, Berlin, 1993.

[13] M. Hussian, A. Grimvall, O. Burdakov, O. Sysoev, Monotonic regression for the detection of temporal trends in environmental quality data, *MATCH Commun. Math. Comput. Chem.*, **54** (2005), 535–550.

[14] J.B. Kruskal, Nonmetric multidimensional scaling: a numerical method, *Psychometrika*, **29** (1964), 115–129.

[15] C.I.C. Lee, The min-max algorithm and isotonic regression, *The Annals of Statistics*, **11** (1983), 467–477.

[16] W.L. Maxwell, J.A. Muchstadt, Establishing consistent and realistic reorder intervals in production-distribution systems, *Operations Research*, **33** (1985), 1316–1341.

[17] R.E. Miles, The complete amalgamation into blocks, by weighted means, of a finite set of real numbers, *Biometrika*, **46**:3/4 (1959), 317–327.

[18] H. Mukarjee, Monotone nonparametric regression, *The Annals of Statistics*, **16** (1988), 741–750.

[19] H. Mukarjee, H. Stern, Feasible nonparametric estimation of multiargument monotone functions, *Journal of the American Statistical Association*, **425** (1994), 77–80.

[20] J. Nocedal, S.J. Wright, Numerical Optimization, Springer-Verlag, New York, 1999.

[21] P.M. Pardalos, G. Xue, Algorithms for a class of isotonic regression problems, *Algorithmica*, **23** (1999), 211–222.

[22] T. Robertson, F.T. Wright, R.L. Dykstra, Order Restricted Statistical Inference, Wiley, New York, 1988.

[23] R. Roundy, A 98% effective lot-sizing rule for a multiproduct multistage production/inventory system, *Mathematics of Operations Research*, **11** (1986), 699–727.

[24] M. Strand, Comparison of methods for monotone nonparametric multiple regression, *Communications in Statistics - Simulation and Computation*, **32** (2003), 165–178.