ISF and BDIFFMIN - Matlab functions for the hyperplane arrangement and the computation of the B-differential of the componentwise minimum of two affine vector functions

Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain

ISF and BDIFFMIN

Matlab functions for the hyperplane arrangement and the computation of the B-differential of the componentwise minimum of two affine vector functions

Version 1.0 (May 1, 2023)

Jean-Pierre Dussault [†]
Jean Charles Gilbert [‡]
Baptiste Plaquevent-Jourdain [††]

The Matlab function `Bdiffmin` aims at computing the B-differential of the componentwise minimum of two affine vector functions. To realize this task, `Bdiffmin` calls the Matlab function `Isf`, which has been designed to determine the chambers of an arrangement of hyperplanes having a point in common. The sign vectors computed by the latter function can be used to solve a number of other enumeration problems such as determining the signed feasibility of strict inequality systems, listing the orthants encountered by the null space of a matrix, itemizing the pointed cones generated by a set of vectors and their inverses, giving the bipartitions of a finite set of points that can be separated by an affine hyperplane and many others.

Keywords: B-differential • Bdiffmin • Componentwise minimum of functions • Hyperplane arrangement • Isf • Linearly separable bipartition of a finite set • Matlab • Matroid circuit • Pointed cone • Schläfli's bound • Stem vector • Strict linear inequalities • Symmetry • Winder's formula.

AMS MSC 2020: 05A18 • 05C40 • 26A24 • 26A27 • 46N10 • 47A50 • 47A63 • 49J52 • 49N15 • 52C35 • 65Y20 • 65K15 • 90C33 • 90C46.

[†]Département d'Informatique, Faculté des Sciences, Université de Sherbrooke, Québec, Canada (e-mail: `Jean-Pierre.Dussault@Usherbrooke.ca`). ORCID 0000-0001-7253-7462.

[‡]Inria Paris, 2 rue Simone Iff, CS 42112, 75589 Paris Cedex 12, France (e-mail: `Jean-Charles.Gilbert@inria.fr`) and Département de Mathématiques, Faculté des Sciences, Université de Sherbrooke, Québec, Canada. ORCID 0000-0002-0375-4663.

[††]Département de Mathématiques, Faculté des Sciences, Université de Sherbrooke, Québec, Canada (e-mail: `Baptiste.Plaquevent-Jourdain@USherbrooke.ca`) and Inria Paris, 2 rue Simone Iff, CS 42112, 75589 Paris Cedex 12, France (e-mail: `Baptiste.Plaquevent-Jourdain@inria.fr`. ORCID 0000-0001-7055-4568).

Table of contents

1 Introduction

This manual is dedicated to the description of the Matlab functions `isf` and `bdiffmin` and is a user-friendly introduction to their use.

`Isf` can solve diverse equivalent enumeration problems related to various areas of mathematics. These problems are described in section 2. One finds problems in linear algebra (sections 2.1 and 2.2), convex analysis (sections 2.3 and 2.4), computational discrete geometry (section 2.5) and nonsmooth analysis (section 2.6). These problems are all occasions where the function `isf` can be useful and there are others (see [29, 2, 3] and the references therein). The right way of formulating a problem in the language understood by `isf` and of adapting the output of the function is not always straightforward; it is the main purpose of section 2 to explain how to make these problem formulation and output adaptation. The computer description of `isf` is made in section 3.

`Bdiffmin` focuses on the computation of the B-differential of the componentwise minimum of two affine vector functions, the problem described in section 2.6. This problem deserves a special function and section since the link between the problems in section 2.1 and 2.6 is probably the least trivial, while the user of `bdiffmin` may not want to understand this link. The computer description of `bdiffmin` is made in section 4.

If you find the functions `isf` and `bdiffmin` useful for your work and that this one is described in a report, you may want to cite [6] (rather than just this manual), since that paper is at the root of the design of these functions. This reference [6] completes the information given in the present manual.

Notation

We denote by $|S|$ the number of elements of a set S (i.e., its *cardinality*). The *power set* of a set S is denoted by $\mathfrak{P}(S)$. The sets of natural and real numbers are denoted by $\mathbb{N}$ and $\mathbb{R}$, respectively, while $\mathbb{N}^* := \mathbb{N} \setminus \{0\}$ and $\mathbb{R}^* := \mathbb{R} \setminus \{0\}$. For n and $p \in \mathbb{N}^*$, $\mathbb{R}^{n\times p}$ denotes the set of $n \times p$ real matrices. For $p \in \mathbb{N}^*$, $[1\!:\!p] := \{1, \ldots, p\}$ is the set of the first p integers. For a subset S of a vector space, we denote by $\mathrm{vect}(S)$ the subspace spanned by S. The *Hadamard product* of u and $v \in \mathbb{R}^n$ is the vector denoted by $u \bullet v \in \mathbb{R}^n$, whose ith component is $u_i v_i$. The *range space* of an $m \times n$ matrix A is denoted by $\mathcal{R}(A)$, its *null space* by $\mathcal{N}(A)$, its *rank* is $\mathrm{rank}(A) := \dim \mathcal{R}(A)$ and its *nullity* is $\mathrm{null}(A) := \dim \mathcal{N}(A) = n - \mathrm{rank}(A)$ by the rank-nullity theorem. The ith row (resp. column) of a matrix A is denoted by $A_{i,:}$ (resp. $A_{:,i}$).

2 Solved problems

The Matlab functions `isf` and `bdiffmin` described in this manual can solve a number of equivalent enumeration problems, which include

- the signed feasibility of strict inequality systems (section 2.1),
- listing the orthants encountered by the null space of a matrix (section 2.2),
- itemizing the pointed cones generated by a set of vectors and their inverses (section 2.3),
- giving the bipartitions of a finite set of points that can be separated by an affine hyperplane (section 2.4),

- determining the chambers of an arrangement of hyperplanes having a point in common (section 2.5) and
- computing the B-differential of the componentwise minimum of two affine vector functions (section 2.6).

The `isf` function deals with the first problem (the one in section 2.1), while the solutions to the other problems can be deduced from the sign vectors provided by `isf` and the transformation rules described in the next subsections. In particular, the `bdiffmin` function solves the last problem (the one in section 2.6), thanks to the use of `isf`.

2.1 Signed feasibility of strict inequality systems

The problem solved by `isf` is the following. A *sign vector* is a vector whose components are $+1$ or -1.

Problem 2.1 (signed feasibility of strict inequality systems) Let be given two integers n and $p \in \mathbb{N}^*$ and a matrix V in $\mathbb{R}^{n\times p}$ with nonzero columns. It is requested to determine the following set of sign vectors

$$\mathcal{S} := \{s \in \{\pm 1\}^p : \ s \bullet (V^\mathsf{T} d) > 0 \text{ is feasible for } d \in \mathbb{R}^n\}. \tag{2.1}$$

□

It is known that $\mathcal{S} \neq \varnothing$. If $s \bullet (V^\mathsf{T} d) > 0$ for some $s \in \mathcal{S}$ and $d \in \mathbb{R}^n$, then $(-s) \bullet (V^\mathsf{T}(-d)) > 0$. Therefore, the set $\mathcal{S}$ is *symmetric*, in the sense that

$$-\mathcal{S} = \mathcal{S}. \tag{2.2}$$

One has the following necessary and sufficient condition for the *completeness* of $\mathcal{S}$ (i.e., $\mathcal{S} = \{\pm 1\}^p$):

$$\mathcal{S} = \{\pm 1\}^p \qquad \Longleftrightarrow \qquad V \text{ is injective.} \tag{2.3}$$

The set $\mathcal{S}$ is potentially large. Its cardinality is given by Winder's formula [29, 1966]:

$$|\mathcal{S}| = \sum_{I \subseteq [1\,:\,p]} (-1)^{\mathrm{null}(V_{:,I})}. \tag{2.4}$$

The right-hand side of (2.4) is usually not easy to evaluate numerically but the following lower and upper bounds are available:

$$\max(2p, 2^r) \leqslant 2^r + 2(p-r) \leqslant |\mathcal{S}| \leqslant 2 \sum_{i \in [0\,:\,r-1]} \binom{p-1}{i} \leqslant 2^p, \tag{2.5}$$

where $r := \mathrm{rank}(V)$. Equalities hold in the following cases (in (2.6a), it is assumed that $p \geqslant 2$ and that V has no colinear columns)

$$|\mathcal{S}| = 2p \quad \Longleftrightarrow \quad \mathrm{rank}(V) = 2, \tag{2.6a}$$

$$|\mathcal{S}| = 2 \sum_{i \in [0\,:\,r-1]} \binom{p-1}{i} \quad \Longleftrightarrow \quad \forall\, I \subseteq [1\,:\,p] : \ \mathrm{rank}(V_{:,I}) = \min(|I|, r), \tag{2.6b}$$

$$|\mathcal{S}| = 2^p \quad \Longleftrightarrow \quad V \text{ is injective.} \tag{2.6c}$$

The implication "$\Leftarrow$" in (2.6b), when $r = n$, was established by the Swiss mathematician Ludwig Schläfli [22, p. 174] before 1852; when the condition in the right-hand side of this equivalence holds, the columns of V are said to be *in general position*.

2.2 Orthants encountered by the null space of a matrix

2.2.1 Problem definition

The equivalent form of problem 2.1 introduced in this section is based on a bijection between the *complementary set* of $\mathcal{S}$ in $\{\pm 1\}^p$, denoted $\mathcal{S}^c := \{\pm 1\}^p \setminus \mathcal{S}$, and a collection $\mathcal{I}$ of subsets of $[1\!:\!p]$ (i.e., $\mathcal{I} \subseteq \mathfrak{P}([1\!:\!p])$), which refers to a collection of orthants of $\mathbb{R}^p$, those encountered by the null space of V.

Problem 2.2 (orthants encountered by the null space of a matrix) Let be given two integers n and $p \in \mathbb{N}^*$ and a matrix V in $\mathbb{R}^{n\times p}$ with nonzero columns. Associate with $I \subseteq [1\!:\!p]$ the following orthant of $\mathbb{R}^p$:

$$\mathcal{O}_I^p := \{y \in \mathbb{R}^p : y_I \geqslant 0,\ y_{I^c} \leqslant 0\},$$

where $I^c := [1\!:\!p] \setminus I$. It is requested to determine the set

$$\mathcal{I} := \{I \subseteq [1\!:\!p] : \mathcal{N}(V) \cap \mathcal{O}_I^p \neq \{0\}\}. \qquad \square$$

Note that, if $I \in \mathcal{I}$, then $I^c \in \mathcal{I}$ (because $y \in (\mathcal{N}(V) \cap \mathcal{O}_I^p) \setminus \{0\}$ implies that $-y \in (\mathcal{N}(V) \cap \mathcal{O}_{I^c}^p) \setminus \{0\}$), so that $|\mathcal{I}|$ is even (just like $|\mathcal{S}|$ and $|\mathcal{S}^c|$; see (2.2)).

The equivalence between problems 2.1 and 2.2 is obtained thanks to the following bijection

$$\imath : s \in \{\pm 1\}^p \to \imath(s) := \{i \in [1\!:\!p] : s_i = +1\} \in \mathfrak{P}([1\!:\!p]), \tag{2.7}$$

whose reverse map is $\imath^{-1} : I \in \mathfrak{P}([1\!:\!p]) \to s \in \{\pm 1\}^p$, where $s_i = +1$ if $i \in I$ and $s_i = -1$ if $i \notin I$. One can show that the restriction of $\imath$ to $\mathcal{S}^c$ is in bijection with $\mathcal{I}$ [6, proposition 3.6]:

The map $\imath$ defined by (2.7) is a bijection from $\mathcal{S}^c$ onto $\mathcal{I}$.

Therefore, to get $\mathcal{I}$ from the sign vector set $\mathcal{S}^c$ computed by `isf`, it suffices to apply $\imath$ to the sign vectors $s \in \mathcal{S}^c$.

2.2.2 Stem vector

Recall that the *nullity* of a matrix A, denoted by $\mathrm{null}(A)$, is the dimension of its null space. Let us introduce the following collection of index sets:

$$\mathcal{C} := \{J \subseteq [1\!:\!p] : J \neq \varnothing,\ \mathrm{null}(V_{:,J}) = 1, V_{:,J_0} \text{ is injective for all } J_0 \subsetneq J\}, \tag{2.8}$$

where "$\subsetneq$" is used to denote strict inclusion. In the terminology of the *vector matroid* formed by the columns of V and its subsets made of linearly independent columns [15, proposition 1.1.1], the elements of $\mathcal{C}$ are called the *circuits* of the matroid [15, proposition 1.3.5(*iii*)]. The particular expression (2.8) of the circuit set is interesting in the present context, since it readily yields the following implication:

$$J \in \mathcal{C} \quad \Longrightarrow \quad \text{any nonzero } \alpha \in \mathcal{N}(V_{:,J}) \text{ has none zero component.} \tag{2.9}$$

From (2.8) and (2.9), one can associate with $J \in \mathcal{C}$ a pair of sign vectors $\pm\tilde{s} \in \{\pm 1\}^J$ by $\tilde{s} := \operatorname{sgn}(\alpha)$ for some nonzero $\alpha \in \mathcal{N}(V_{:,J})$; the sign vectors $\pm\tilde{s}$ do not depend on the chosen $\alpha \in \mathcal{N}(V_{:,J}) \setminus \{0\}$ since $\operatorname{null}(V_{:,J}) = 1$. We call such a sign vector a *stem vector*, because of [7, proposition 3.9], which shows that any $s \in \mathcal{S}^c$ can be generated from such a stem vector, in the sense that

$$s \in \mathcal{S}^c \quad \Longleftrightarrow \quad s_J = \tilde{s} \text{ for some } J \subseteq [1\!:\!p] \text{ and some stem vector } \tilde{s}.$$

Definition 2.3 (stem vector) A *stem vector* is a sign vector $\tilde{s} = \operatorname{sgn}(\alpha)$, where $\alpha \in \mathcal{N}(V_{:,J})$ for some $J \in \mathcal{C}$. □

2.3 Pointed cones by vector inversions

A *convex cone* K of $\mathbb{R}^n$ is a convex set verifying $\mathbb{R}_{++}K \subseteq K$ (or, more explicitly, $tx \in K$ when $t > 0$ and $x \in K$). A *closed convex cone* K is said to be *pointed* if $K \cap (-K) = \{0\}$ [4, p. 54], which amounts to saying that K does not contain a line (i.e., an affine subspace of dimension one) or that K has no nonzero direction z such that $-z \in K$. For $P \subseteq \mathbb{R}^n$, we also denote by "cone P" the smallest *convex* cone containing P.

Problem 2.4 (pointed cones by vector inversions) Let be given two integers n and $p \in \mathbb{N}^*$ and p vectors $v_1, \ldots, v_p \in \mathbb{R}^n \setminus \{0\}$. It is requested to determine all the sign vectors $s \in \{\pm 1\}^p$ such that $\operatorname{cone}\{s_i v_i : i \in [1\!:\!p]\}$ is pointed. □

The equivalence between the original problem 2.1 and this problem 2.4 is obtained thanks to the next equivalence ([11, theorem 2.3.29], [4, theorem 3.3.15], [7, proposition 3.12] and many other contributions):

$$\operatorname{cone}\{v_i : i \in [1\!:\!p]\} \text{ is pointed} \quad \Longleftrightarrow \quad \exists\, d \in \mathbb{R}^n,\ \forall i \in [1\!:\!p] : v_i^{\mathsf{T}} d > 0. \tag{2.10}$$

It follows that the set of sign vectors $\mathcal{S}$, determined by `isf`, is precisely the set of sign vectors required by problem 2.4, which reads

$$\mathcal{S} = \{s \in \{\pm 1\}^p :\ \operatorname{cone}\{s_i v_i : i \in [1\!:\!p]\} \text{ is pointed}\}. \tag{2.11}$$

2.4 Linearly separable bipartitions of a finite set

Below, a partition of a set S made of two parts is said to be a *bipartition* of S. It is therefore a pair (I, J) such that $I \cup J = S$ and $I \cap J = \varnothing$. The bipartition (I, J) is said to be *ordered* if (I, J) is considered to be different from (J, I). The set of ordered bipartitions of S is denoted by $\mathfrak{B}(S)$.

Problem 2.5 (linearly separable bipartitioning) Let be given $n \geqslant 2$ in $\mathbb{N}$, $p \in \mathbb{N}^*$ and p vectors $w_1, \ldots, w_p \in \mathbb{R}^{n-1}$. It is requested to find all the $(I, J) \in \mathfrak{B}([1\!:\!p])$, for which there exists a vector $\xi \in \mathbb{R}^{n-1}$ such that

$$\forall\, i \in I,\ \forall\, j \in J : \quad \xi^{\mathsf{T}} w_i < \xi^{\mathsf{T}} w_j. \tag{2.12}$$

Such an ordered bipartition is said to a *linearly separable bipartition* of the set $\{w_i : i \in [1\!:\!p]\}$. □

Like for problem 2.2, note that, if (I, J) is a linearly separable bipartition of $\{w_i : i \in [1:p]\}$, then (J, I) is also a linearly separable bipartition of $\{w_i : i \in [1:p]\}$ (replace ξ by $-\xi$). Therefore, the number of linearly separable bipartitions is even, which refers to the symmetry of $\mathcal{S}$ in (2.2). The problem is illustrated by figure 2.1.

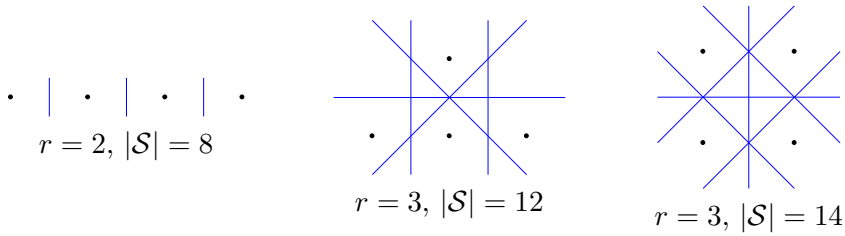

Figure 2.1: Linearly separable bipartitions of a set of $p = 4$ points w_i in $\mathbb{R}^2$ (the dots in the figure). Possible separating hyperplanes are the drawn lines. We have not represented any separating line associated with the bipartition $(\varnothing, [1:p])$ or $([1:p], \varnothing)$, so that the number of linearly separable bipartitions, which is also $|\mathcal{S}|$, is $2(n_s + 1)$, where n_s is the number of represented separating lines. We have set $r := \mathrm{rank}(V) = \dim(\mathrm{vect}\{w_1, \ldots, w_p\}) + 1$ (the matrix $V \in \mathbb{R}^{3\times 4}$ is defined by (2.13)).

Let us now see how the solution to this problem 2.5 can be deduced from the solution to problem 2.1 (the output of the function `isf`). This is done by the following algorithm.

Algorithm 2.6 (linearly separable bipartitioning)
Let be given $n \geqslant 2$ in $\mathbb{N}$, $p \in \mathbb{N}^*$ and p nonzero vectors $w_1, \ldots, w_p$ in $\mathbb{R}^{n-1}$.

1. For $i \in [1:p]$, define $v_i := (w_i, 1) \in \mathbb{R}^n$. Form $V \in \mathbb{R}^{n\times p}$ by
$$V = \begin{pmatrix} v_1 & \cdots & v_p \end{pmatrix}. \tag{2.13}$$
2. Compute $\mathcal{S}$ defined by (2.1) (this set can be computed by the function `isf`).
3. For each $s \in \mathcal{S}$, define
$$I := \{i \in [1:p] : s_i = -1\} \qquad \text{and} \qquad J := \{i \in [1:p] : s_i = +1\}.$$
Then, (I, J) is a linearly separable bipartition of $\{w_i : i \in [1:p]\}$.

The claim made in step 3 of algorithm 2.6 is justified by two facts. First, note that $\mathrm{cone}\{v_i : i \in [1:p]\}$ is pointed (use $d = (0, \ldots, 0, 1)$ in (2.10)). Next, it follows that, for the bipartition (I, J) of $[1:p]$ defined in step 3, one has

$$\begin{aligned} (I, J) \text{ is a linearly separable bipartition} &\iff \mathrm{cone}\{s_i v_i : i \in [1:p]\} \text{ is pointed} \\ &\iff s \in \mathcal{S}. \end{aligned}$$

The first equivalence is justified by [6, proposition 3.16], while the second equivalence is (2.11). Therefore, algorithm 2.6 computes all the linearly separable bipartitions of the set $\{w_i : i \in [1:p]\}$.

2.5 Hyperplane arrangements

The problem examined in this section has a long history, going back at least to the XIXth century [27, 20]. More recently, it appears in *computational discrete geometry* (the discipline has many other names), under the name of *hyperplane arrangements*. Contributions to this problem, or more general versions of it, with a discrete mathematics point of view, has been reviewed in [12, 9, 26, 1, 13]. It has many applications [8, 24, 28].

Problem 2.7 (arrangement of hyperplanes containing the origin) Let be given two integers n and $p \in \mathbb{N}^*$ and p nonzero vectors $v_1, \ldots, v_p \in \mathbb{R}^n$. Consider the hyperplanes containing the origin:

$$\mathcal{H}_i := \{d \in \mathbb{R}^n : v_i^{\mathsf{T}} d = 0\}. \tag{2.14}$$

It is requested to list the regions of $\mathbb{R}^n$ that are separated by these hyperplanes. Such a region is called a *sector* or a *cell* or a *chamber*, depending on the authors [2, 23, 1]. More specifically, let us define the half-spaces

$$\mathcal{H}_i^+ := \{d \in \mathbb{R}^n : v_i^{\mathsf{T}} d > 0\} \qquad \text{and} \qquad \mathcal{H}_i^- := \{d \in \mathbb{R}^n : v_i^{\mathsf{T}} d < 0\}.$$

The problem is to determine the following set of open sectors of $\mathbb{R}^n$, indexed by the ordered bipartitions (I_+, I_-) of $[1\,{:}\,p]$:

$$\mathfrak{C} := \left\{(I_+, I_-) \in \mathfrak{B}([1\,{:}\,p]) : (\cap_{i\in I_+} \mathcal{H}_i^+) \cap (\cap_{i\in I_-} \mathcal{H}_i^-) \neq \varnothing\right\}, \tag{2.15}$$

where $\mathfrak{B}([1\,{:}\,p])$ denotes the set of ordered bipartitions of $[1\,{:}\,p]$. □

Let us now show how to get $\mathfrak{C}$ from $\mathcal{S}$, the latter set being given by the function `isf`, for instance. Observe that, for V given by (2.13), $(I_+, I_-) \in \mathfrak{B}([1\,{:}\,p])$ and $s \in \{\pm 1\}^p$ linked by $s_i = +1$ for $i \in I_+$ and $s_i = -1$ for $I \in I_-$, one has

$$\begin{aligned}
(I_+, I_-) \in \mathfrak{C} \quad &\Longleftrightarrow \quad \exists\, d \in \mathbb{R}^n : \ \begin{cases} v_i^{\mathsf{T}} d > 0 & \text{for } i \in I_+ \\ v_i^{\mathsf{T}} d < 0 & \text{for } i \in I_- \end{cases} \\
&\Longleftrightarrow \quad \exists\, d \in \mathbb{R}^n : \ s \bullet (V^{\mathsf{T}} d) > 0 \\
&\Longleftrightarrow \quad s \in \mathcal{S}.
\end{aligned}$$

Therefore, for $s \in \mathcal{S}$ given by the function `isf`, $(I_+, I_-) \in \mathfrak{B}([1\,{:}\,p])$ defined by

$$I_+ := \{i \in [1\,{:}\,p] : s_i = +1\} \qquad \text{and} \qquad I_- := \{i \in [1\,{:}\,p] : s_i = -1\}$$

is in $\mathfrak{C}$.

2.6 B-differential of the minimum of two affine functions

2.6.1 Problem definition

The *B-differential at* $x \in \mathbb{R}^n$ of a function $H : \mathbb{R}^n \to \mathbb{R}^m$ is the set denoted and defined by

$$\partial_B H(x) := \{J \in \mathcal{L}(\mathbb{R}^n, \mathbb{R}^m) : H'(x_k) \to J \text{ for } \{x_k\} \subseteq \mathcal{D}_H \text{ converging to } x\}, \tag{2.16}$$

where $\mathcal{L}(\mathbb{R}^n, \mathbb{R}^m)$ is the set of linear (continuous) maps from $\mathbb{R}^n$ to $\mathbb{R}^m$, $\{x_k\}$ denotes a sequence and $\mathcal{D}_H$ is the set of points at which H is (Fréchet) differentiable (its derivative

at x is denoted by $H'(x)$, an element of $\mathcal{L}(\mathbb{R}^n, \mathbb{R}^m)$). Hence, when H is continuously differentiable at x, $\partial_B H(x) = \{H'(x)\}$, indicating that this notion is rather adapted to nonsmooth functions. Recall that a locally Lipschitz continuous function is differentiable almost everywhere in the sense of the Lebesgue measure (Rademacher's theorem [19]) and this property has the consequence that the B-differential of a locally Lipschitz function is nonempty and bounded everywhere [5]. The B-differential is an intermediate set used to define the C-differential (C for Clarke [5]) of H at x, which is denoted and defined at $x \in \mathbb{R}^n$ by

$$\partial_C H(x) := \operatorname{co} \partial_B H(x), \tag{2.17}$$

where $\operatorname{co} S$ denotes the convex hull of a set S [21, 14, 4]. Both intervene in the specification of conditions ensuring the local convergence of the semismooth Newton algorithm [16, 17, 25], which can be a motivation for being interested in that concept.

Problem 2.8 (B-differential of the minimum of two affine functions) Let be given two integers n and $m \in \mathbb{N}^*$, two matrices A, $B \in \mathbb{R}^{m \times n}$ and two vectors a, $b \in \mathbb{R}^m$. It is requested to compute the B-differential at some $x \in \mathbb{R}^n$ of the function $H : \mathbb{R}^n \to \mathbb{R}^m$ defined at $x \in \mathbb{R}^n$ by

$$H(x) = \min(Ax + a, Bx + b), \tag{2.18}$$

where the minimum operator "min" acts componentwise (for two vectors u, $v \in \mathbb{R}^m$ and $i \in [1\,{:}\,m]$: $[\min(u, v)]_i := \min(u_i, v_i)$). □

This problem 2.8 can be solved by the Matlab function `bdiffmin` presented in section 4. Let us now see the links between problem 2.8 and problem 2.1.

2.6.2 From $\partial_B H(x)$ to $\mathcal{S}$

We start in this section by showing how a Jacobian $J \in \partial_B H(x)$ can be associated with some $s \in \mathcal{S}$.

Recall the definition 2.16 of the B-differential $\partial_B H(x)$. It is known that [5, proposition 2.6.2(e)]

$$\partial_B H(x) \subseteq \partial_B H_1(x) \times \cdots \times \partial_B H_m(x) =: \overline{\partial}_B H(x), \tag{2.19}$$

but that equality in this inclusion may not hold (see [10, section 7.1.15], [6, counter-example 2.3] and almost all the examples and test-cases in the latter paper). It is also known that for H given by (2.18) (see [6, lemma 2.1], for example)

$$\partial_B H_i(x) = \begin{cases} \{A_{i:}\} & \text{if } (Ax + a)_i < (Bx + b)_i, \\ \{A_{i:}, B_{i:}\} & \text{if } (Ax + a)_i = (Bx + b)_i, \\ \{B_{i:}\} & \text{if } (Ax + a)_i > (Bx + b)_i. \end{cases} \tag{2.20}$$

In view of these cases, it is useful to introduce the following index sets:

$$\begin{aligned} \mathcal{A}(x) &:= \{i \in [1\,{:}\,m] : (Ax + a)_i < (Bx + b)_i\}, \\ \mathcal{B}(x) &:= \{i \in [1\,{:}\,m] : (Ax + a)_i > (Bx + b)_i\}, \\ \mathcal{E}(x) &:= \{i \in [1\,{:}\,m] : (Ax + a)_i = (Bx + b)_i\}, \\ \mathcal{E}^{=}(x) &:= \{i \in \mathcal{E}(x) : A_{i,:} = B_{i,:}\}, \\ \mathcal{E}^{\neq}(x) &:= \{i \in \mathcal{E}(x) : A_{i,:} \neq B_{i,:}\}. \end{aligned}$$

The identities (2.19) and (2.20) readily imply that, a Jacobian $J \in \partial_B H(x)$ satisfies

$$J_{i,:} = \begin{cases} A_{i:} & \text{if } i \in \mathcal{A}(x) \cup \mathcal{E}^{=}(x), \\ B_{i:} & \text{if } i \in \mathcal{B}(x) \cup \mathcal{E}^{=}(x) \end{cases} \tag{2.22}$$

and that

$$\begin{aligned} \overline{\partial}_B H(x) := \{ J \in \mathcal{L}(\mathbb{R}^n, \mathbb{R}^m) : \ & J_{i,:} = A_{i,:}, \text{ if } i \in \mathcal{A}(x), \\ & J_{i,:} = A_{i,:} = B_{i,:}, \text{ if } i \in \mathcal{E}^{=}(x), \\ & J_{i,:} \in \{A_{i,:}, B_{i,:}\}, \text{ if } i \in \mathcal{E}^{\neq}(x), \\ & J_{i,:} = B_{i,:}, \text{ if } i \in \mathcal{B}(x) \}. \end{aligned} \tag{2.23}$$

The values that $J_{i,:}$ can take for $i \in \mathcal{E}^{\neq}(x)$ are more tricky to determine. In some places below, our reasoning is expeditious, but it follows rather closely the meticulous treatment made in [6, section 3.2.1]. To simplify the presentation, we assume in the sequel that

$$\mathcal{E}^{\neq}(x) = [1\,{:}\,p],$$

for some $p \in [1\,{:}\,m]$ (actually, $p = 0$ if and only if $\mathcal{E}^{\neq}(x) = \varnothing$, in which case $\partial_B H(x)$ is determined by (2.22)). For the rest of the discussion, assume that $J \in \partial_B H(x)$. Then, by (2.20):

$$\forall\, i \in [1\,{:}\,p] : \quad J_{i,:} \in \{A_{i:}, B_{i:}\}.$$

However, as said above, the values of $J_{i,:}$ cannot be determined componentwise, but all the components $J_{[1\,:\,p],:}$ must be considered simultaneously. To do so, one has to go back to the definition (2.16) of the B-differential and consider a sequence $\{x_k\} \subseteq \mathcal{D}_H$ such that $x_k \to x$ and $H'(x_k) \to J$. Since $\{x_k\} \subseteq \mathcal{D}_H$ and $A_{i,:} \neq B_{i,:}$, one cannot have $(Ax_k{+}a)_i = (Bx_k{+}b)_i$ for $i \in [1\,{:}\,p]$ [6, lemma 2.1]. Therefore, one can find a subsequence $\mathcal{K}$ of indices k and a partition $(\mathcal{A}_0, \mathcal{B}_0)$ of $\mathcal{E}^{\neq}(x)$ such that for all $k \in \mathcal{K}$:

$$(Ax_k + a)_{\mathcal{A}_0} < (Bx_k + b)_{\mathcal{A}_0} \qquad \text{and} \qquad (Ax_k + a)_{\mathcal{B}_0} > (Bx_k + b)_{\mathcal{B}_0}. \tag{2.24}$$

Therefore, for $k \in \mathcal{K}$, $H_i'(x_k) = A_{i,:}$ if $i \in \mathcal{A}_0$ and $H_i'(x_k) = B_{i,:}$ if $i \in \mathcal{B}_0$. Since $H'(x_k) \to J$, it follows that

$$J_{i,:} = \begin{cases} A_{i,:} & \text{if } i \in \mathcal{A}_0, \\ B_{i,:} & \text{if } i \in \mathcal{B}_0. \end{cases} \tag{2.25}$$

Now, fixing $k \in \mathcal{K}$, setting $d := x_k - x$ and using $(Ax + a)_i = (Bx + b)_i$ for $i \in [1\,{:}\,p]$, one deduces from (2.24) that

$$(B - A)_{\mathcal{A}_0,:} d > 0 \qquad \text{and} \qquad (B - A)_{\mathcal{B}_0,:} d < 0. \tag{2.26}$$

By these inequalities, we see that what matters here is the matrix

$$V := (B - A)^{\mathsf{T}}_{\mathcal{E}^{\neq}(x),:} \in \mathbb{R}^{n \times p}. \tag{2.27}$$

By (2.25), (2.26) and (2.27), we have found a sign vector $s \in \{\pm 1\}^p$ defined by

$$s_i = \begin{cases} +1 & \text{if } i \in \mathcal{A}_0 \text{ or } J_{i,:} = A_{i,:} \\ -1 & \text{if } i \in \mathcal{B}_0 \text{ or } J_{i,:} = B_{i,:} \end{cases} \tag{2.28}$$

and a direction $d \in \mathbb{R}^n$ such that

$$s \bullet (V^{\mathsf{T}} d) > 0. \tag{2.29}$$

In conclusion, for the matrix V given by (2.27), (2.28) and (2.29) show that we have found an $s \in \mathcal{S}$ associated with the given $J \in \partial_B H(x)$.

The established link between $J \in \partial_B H(x)$ and $s \in \mathcal{S}$ can be formalized by the following map

$$\sigma : J \in \partial_B H(x) \mapsto s \in \mathcal{S}, \quad \text{where } s_i = \begin{cases} +1 & \text{if } J_{i,:} = A_{i,:} \\ -1 & \text{if } J_{i,:} = B_{i,:} \end{cases} \tag{2.30}$$

and the following inclusion

$$\sigma(\partial_B H(x)) \subseteq \mathcal{S}. \tag{2.31}$$

An illustration of the problem when $n = 2$ and $p = 3$ is given in figure 2.2. Then,

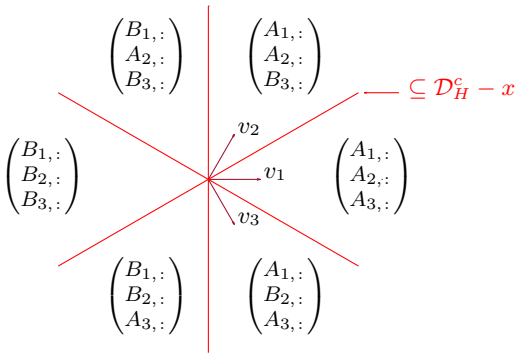

Figure 2.2: The vectors v_i's are given by $v_i = (B-A)_{i,:}^\mathsf{T}$. The red lines are the "hyperplanes" $\mathcal{H}_i := \{d \in \mathbb{R}^n : v_i^\mathsf{T} d = 0\}$. Translated by x, these form the points where H, given by (2.18), is nondifferentiable. The function H has a constant Jacobian in each of the 6 sectors (translated by x), which is the one that is indicated in the figure.

$\text{rank}(V) = 2$ (if the columns of V are not all colinear) and $|\mathcal{S}| = 2p$ by (2.6a). The situation is usually much more complex when $n > 2$ and p is larger.

2.6.3 From $\mathcal{S}$ to $\partial_B H(x)$

In this section, we do the reverse operation of the one that was done in section 2.6.2 and show how a Jacobian $J \in \partial_B H(x))$ can be associated with a given sign vector $s \in \mathcal{S}$. We actually show that equality holds in (2.31). This will result from the inversion of the bijective map σ:

$$\sigma^{-1} : s \in \mathcal{S} \mapsto J \in \partial_B H(x), \text{ where } J_{i,:} = \begin{cases} A_{i,:} & \text{if } i \in [1\!:\!p] \text{ and } s_i = +1, \\ B_{i,:} & \text{if } i \in [1\!:\!p] \text{ and } s_i = -1. \end{cases} \tag{2.32}$$

In (2.32), we have not specified the value of $J_{i,:}$ for $i \notin [1\!:\!p]$, which are actually given by (2.22). It is this association that is used by `bdiffmin`, to compute the B-differential $\partial_B H(x)$ from the set $\mathcal{S}$ computed by `isf`.

Let V be given by (2.27) and $s \in \mathcal{S}$ (this sign vector set is associated with the given V). By (2.1), (2.29) holds for some $d \in \mathbb{R}^n$. Introducing

$$\mathcal{A}_0 := \{i \in [1\!:\!p] : s_i = +1\} \qquad \text{and} \qquad \mathcal{B}_0 := \{i \in [1\!:\!p] : s_i = -1\},$$

we get (2.26). Now, defining $x_k := x+t_k d$, for some $\{t_k\} \downarrow 0$, and using $(Ax+a)_i = (Bx+b)_i$ for $i \in [1:p]$, one deduces (2.24), which shows that $x_k \in \mathcal{D}_H$ and $H'(x_k)$ is independent of k and has the value $H'_i(x_k) = A_{i,:}$ for $i \in \mathcal{A}_0$ and $H'_i(x_k) = B_{i,:}$ for $i \in \mathcal{B}_0$. Therefore J defined by (2.22) and

$$J_{i,:} := \begin{cases} A_{i:} & \text{if } i \in [1:p] \text{ and } s_i = +1 \\ B_{i:} & \text{if } i \in [1:p] \text{ and } s_i = -1 \end{cases}$$

is in $\partial_B H(x)$.

3 The `isf` function

The Matlab function `isf` solves problem 2.1: for a given matrix $V \in \mathbb{R}^{n\times p}$, it computes the sign vector set $\mathcal{S}$ given by (2.1); see section 2.1. The selected sign vectors are the leaves of a tree of sign vectors of increasing size, called the $\mathcal{S}$-tree below. The *binary representation* of a sign vector $s \in \{\pm 1\}^p$ is $(s+1)/2$, which is indeed only formed of elements in $\{0,1\}$. This binary representation is useful, in particular, for printing s in a compact manner.

The name of the `isf` function stands for *Incremental Sign Feasibility* (not *Impôt Sur la Fortune*). The term *Incremental* refers to the fact that the algorithm constructs the $\mathcal{S}$-tree of the sign vectors incrementally (and recursively). The term *Sign* refers to the sign vectors computed by the function. The term *Feasibility* refers to the fact that the feasibility of $s \bullet (V^\mathsf{T} d) > 0$ in $d \in \mathbb{R}^n$ is used to select the appropriate sign vectors. Finally, the juxtaposition of terms that is *Incremental Sign Feasibility* has a weak meaning, which has the merit of bringing together the key concepts that characterize the function.

3.1 Specifications

```
function [info] = isf(V,options)
```

A description of the function can be obtained by entering "`help isf`" in Matlab. We only describe here the main structure fields of input and output variable.

3.1.1 Input variables

`V`: This is the matrix $V \in \mathbb{R}^{n\times p}$ from which the set $\mathcal{S}$ of sign vectors given by (2.1) is computed. The dimensions n and p are deduced from the size of V. The matrix cannot have a zero column; in this case $\mathcal{S} = \varnothing$ and `isf` returns with `info.flag = 3`.

`options`: Structure array allowing the user to tune the behavior of `isf`. All options have default values, so that the `options` argument can be omitted or set to `[]`. The following options are available (they are given in alphabetic order of the field names).

`options.bestv`: integer in $\{0,3\}$ (default 0 if `options.sv == 3`, default 3 if `options.sv < 3`; ignored if `options.rc2018 == true` or if `options.withd == false`). This option can be used to modify the order in which the vectors v_i's (i.e., the columns of V) are considered for constructing the $\mathcal{S}$-tree. By default, this order is that imposed by the

QR factorization of V. When `options.bestv > 0`, a reordering is performed following a heuristics whose aim is to decrease the number of nodes of the $\mathcal{S}$-tree, which has an impact on the number of linear optimization problems (LOP) that must be solved. Only the values 0 and 3 are actually evaluated in [6].

= 0: no modification of the order, except that imposed by the QR factorization;
= 3: a reordering of the vectors is made so that the number of nodes of the $\mathcal{S}$-tree decreases (and therefore the number of LOP to solve).

`options.dvnear0`: logical (default `true`; ignored if `options.rc2018 == true` or if `options.withd == false`). If `true`, the $\mathcal{S}$-tree algorithm constructs two descendants when $v^{\mathsf{T}}d$ is in a specific computed interval surrounding zero, without having to solve a LOP or to use stem vectors if any (v is here the new considered vector and d is the direction intervening in (2.1) at the current node of the $\mathcal{S}$-tree).

`options.fout`: integer, default `1`. First output channel containing all but the printing made during the generation of the $\mathcal{S}$-tree. Set 1 (default) for the standard output (screen). For a specific file, use "`options.fout = fopen(...)`" before calling `isf`.

`options.fout2`: integer, default `1`. Second output channel containing the printing made during the generation of the $\mathcal{S}$-tree. Set 1 (default) for the standard output (screen). For a specific file, use "`options.fout2 = fopen(...)`" before calling `isf`.

`options.rc2018`: logical (default `false`). If `true`, the simulated Rada and Černý algorithm of [18] is run, ignoring most other options.

`options.s`: logical (default `true`). If `true`, half the sign vectors $s \in \mathcal{S}$ are stored in `info.s`, in binary representation. The other half can be obtained by symmetry: `ones(size(info.s))-info.s`.

`options.sc`: logical (default `false`). If `true` and if `options.sv < 3`, half the sign vectors $s \in \mathcal{S}^c$ are stored in `info.sc`, in binary representation. The other half can be obtained by symmetry: `ones(size(info.sc))-info.sc`.

`options.sv`: integer in $[0:3]$ (default 3; ignored if `options.rc2018 == true`). Determine whether and how many stem vectors (definition 2.3) must be used (the i of option D_i in [6, section 5.2.5(D)]).

= 0: do not compute/use stem vectors;
= 1: use the $p - r$ stem vectors that can be computed thanks to the QR factorization of V (r denotes the rank of V); this is not many, but it is inexpensive to compute;
= 2: in addition to the $p - r$ stem vectors obtained with "`options.sv = 1`", use the dual solution to each linear optimization problem having no solution to add a stem vector to the list (see [6, prosition 5.9]); this improves significantly the speed of the algorithm, but requires to solve the LOP with the dual simplex method;
= 3: compute all the stem vectors at the beginning of the run (time consuming operation when p is large).

`options.verb`: integer in $[0:\infty]$ (default 1). Verbosity level of the output channel `options.fout`.

= 0: `isf` works silently on channel `options.fout`;

$\geqslant 1$: `isf` prints error messages, initial setting, final status on channel `options.fout`;
$\geqslant 2$: `isf` also prints more information on channel `options.fout`.

`options.verb2`: integer in $[0:\infty]$ (default 1). Verbosity level of the output channel `options.fout2`.

$= 0$: `isf` works silently on channel `options.fout2`;
$\geqslant 1$: `isf` also prints error messages and the binary representation of the sign vectors;
$\geqslant 2$: `isf` also prints the feasible directions d intervening in (2.1);
$\geqslant 3$: `isf` also prints some information at the intermediate steps of the recursivity process;
$\geqslant 4$: `isf` also checks that the directions d intervening in (2.1) verify indeed $s\bullet(V^{\mathsf{T}}d) > 0$, which is a certificate on the correctness of the computation.

`options.withd`: logical; default `true` if `options.sv < 3`; default `false` if `options.sv == 3`. If `true`, `isf` is required to compute a direction d (the one intervening in (2.1)) at each node of the $\mathcal{S}$-tree; this requires more computation, in particular for solving LOPs.

The table 3.1 below gives the combinations of options used in [6] to get the algorithms named there "simulated RC" (Rada and Černý [18] simulated algorithm), `isf`(A), `isf`(AB), `isf`(ABC), `isf`(ABCD_1), `isf`(ABCD_2), `isf`(ABCD_3) and `isf`(AD_4). As one can see in

options	Simulated RC	isf(A)	isf(AB)	isf(ABC)	isf(ABCD_1)	isf(ABCD_2)	isf(ABCD_3)	isf(AD_4)
rc2018	true							
dvnear0	—	false						
bestv	—	0	0	3	3	3	3	
sv	—	0	0	0	1	2	3	
withd	—						true	

Table 3.1: Options to specify to get the algorithms used in [6]. Default values apply for the unspecified options (these default values can depend on the values of other options). The symbol "—" in the "simulated RC" column means that the option is not used in that algorithm.

table 3.1, the default algorithm is `isf`(AD_4), since this one is obtained by the default values of the options `rc2018`, `dvnear0`, `bestv`, `sv` and `withd`. This does not mean, however, that `isf`(AD_4) is always the best (fastest) algorithm. This one actually depends on the problem data. To this respect, one can give the following guidelines.

G_1. The best (fastest) algorithm is always among the three following ones (those in the blue columns of table 3.1)

$$\texttt{isf}(\text{ABCD}_2), \quad \texttt{isf}(\text{ABCD}_3) \quad \text{and} \quad \texttt{isf}(\text{AD}_4).$$

According to the tests realized in [6], the others are always worse in terms of the number of linear optimization problems to solve and of the computing time (there are reasons for this, see [6]). We mention them to make the link with the algorithms benchmarked in [6], not to recommand their use.

G_2. The algorithm `isf`(AD_4) is particularly efficient when there is not too many stem vectors. The reason is that in the current versioin of `isf`, it is time consuming to

detect the stem vectors and to use them. Since the number of stem vectors rapidly increases with p, the number p of columns of V should not be too large for algorithm `isf`(AD_4) to be efficient.

G$_3$. When p becomes large, the algorithm `isf`(ABCD_2) usually becomes faster than `isf`(AD_4).

G$_4$. The algorithm `isf`(ABCD_3) is somehow intermediate between `isf`(ABCD_2) and `isf`(AD_4). It can be faster than `isf`(ABCD_2), but then `isf`(AD_4) is usually even much faster. It can be faster than `isf`(AD_4), but then `isf`(ABCD_2) is usually even much faster. Therefore, algorithm `isf`(ABCD_3) could be chosen if one has no information on what means a large p for the type of matrix V that one has to deal with.

3.1.2 Output variable

`info`: Structure array giving the output of `isf`, as well as information in the run. The following fields are available.

`info.flag`: integer in $[0:9]$ giving the diagnosis of the run.

- $= 0$: the required job has been realized;
- $= 1$: an input argument is wrong;
- $= 2$: nothing to do since V has no column ($p = 0$);
- $= 3$: one of the vectors $V_{:,i}$ vanishes, implying that $\mathcal{S} = \varnothing$;
- $= 4$: when a direction d intervening in (2.1) is computed (`options.withd == true`) and that the checking of $s \bullet (V^{\mathsf{T}} d) > 0$ is asked (`options.verb2 >= 4`), this diagnosis flag means that the checking failed; this should be due to rounding error;
- $= 5$: one vector is opposite to another one, in which case $\mathcal{S} = \varnothing$;
- $= 6$: the linear optimization solver failed;
- $= 9$: a "technical" problem has been encountered, which requires improvements of the code.

`info.ns`: $= |\mathcal{S}|/2$, half the number of sign vectors in $\mathcal{S}$.

`info.nsc`: $= |\mathcal{S}^c|/2$, half the number of sign vectors in $\mathcal{S}^c := \{\pm 1\}^p \setminus \mathcal{S}$.

`info.s`: matrix of size (`info.ns`)$\times p$, whose rows give half the sign vectors s in $\mathcal{S}$, in binary representation. The other half can be obtained by symmetry: `ones(size(info.s))-info.s`.

`info.sc`: if `options.sc` is `true` and `options.sv < 3` (otherwise the field does not exist), it is a matrix of size (`info.nsc`) $\times\, p$, whose rows give half the infeasible sign vectors s in $\mathcal{S}^c$, in binary representation. The other half can be obtained by symmetry: `ones(size(info.sc))-info.sc`.

3.2 Example of use

Consider the example specified by the matrix

$$V = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

Hence $n = 3$ and $p = 6$. Even though V is "simple", determining $\mathcal{S}$ is not trivial. For this, run `isf` with the given options (for example):

```
options.verb  = 0;
options.verb2 = 0;
options.sv    = 2;
options.sc    = true;                % requires options.sv < 3
[info]        = isf(V,options);
```

Then, `isf` prints nothing on the standard output. At the end of the run, one can explore the output structure `info` to have information on the run and on the computed values.

- Since `info.flag` is `0`, the run is successful.
- Half the number of sign vectors in $\mathcal{S}$ is `info.ns` $= 13$ and half the number of sign vectors in $\mathcal{S}^c$ is `info.nsc` $= 19$. Observe that $13 + 19 = 2^{p-1}$.
- In `info.s`, one finds the binary representation of half of the 26 sign vectors in $\mathcal{S}$:

```
1     1     1     1     1     1
1     0     1     1     1     1
1     0     1     1     0     1
1     0     1     0     1     1
1     0     1     0     0     1
1     1     0     1     1     1
1     1     0     1     1     0
1     1     0     1     0     1
1     1     0     1     0     0
1     0     0     1     0     1
1     0     0     1     0     0
1     0     0     0     0     1
1     0     0     0     0     0
```

  The other half of $\mathcal{S}$ can be obtained by symmetry (`ones(size(info.s))-info.s`):

```
0     0     0     0     0     0
0     1     0     0     0     0
0     1     0     0     1     0
0     1     0     1     0     0
0     1     0     1     1     0
0     0     1     0     0     0
0     0     1     0     0     1
0     0     1     0     1     0
0     0     1     0     1     1
0     1     1     0     1     0
0     1     1     0     1     1
0     1     1     1     1     0
0     1     1     1     1     1
```

- Since `options.sc == true` and `options.sv < 3`, one finds in `info.sc` the binary representation of half of the 38 sign vectors in $\mathcal{S}^c$:

```
1     1     1     1     1     0
1     1     1     1     0     0
1     1     1     1     0     1
1     1     1     0     0     0
1     1     1     0     0     1
```

```
1     1     1     0     1     0
1     1     1     0     1     1
1     0     1     0     0     0
1     0     1     0     0     1
1     0     1     0     1     0
1     0     1     0     1     1
1     1     0     0     0     0
1     1     0     0     0     1
1     1     0     0     1     0
1     1     0     0     1     1
1     0     0     0     0     0
1     0     0     0     0     1
1     0     0     0     1     0
1     0     0     0     1     1
```

The other half of $\mathcal{S}^c$ can be obtained by symmetry (`ones(size(info.sc))-info.sc`):

```
0     0     0     0     0     1
0     0     0     0     1     1
0     0     0     0     1     0
0     0     0     1     1     1
0     0     0     1     1     0
0     0     0     1     0     1
0     0     0     1     0     0
0     1     0     1     1     1
0     1     0     1     1     0
0     1     0     1     0     1
0     1     0     1     0     0
0     0     1     1     1     1
0     0     1     1     1     0
0     0     1     1     0     1
0     0     1     1     0     0
0     1     1     1     1     1
0     1     1     1     1     0
0     1     1     1     0     1
0     1     1     1     0     0
```

4 The `bdiffmin` function

As already mentioned in section 2.6, `bdiffmin` computes the B-differential of the componentwise minimum of two affine vector functions, which is the function H in (2.18). As explained in section 2.6.3, `bdiffmin` computes $\partial_B H(x)$ from the sign vector set $\mathcal{S}$ in (2.1), obtained by running `isf`.

4.1 Specifications

```
function [info] = bdiffmin(A,a,B,b,x,options)
```

A description of the function can be obtained by entering "`help bdiffmin`" in Matlab. We only describe here the main structure fields of input and output variable.

4.1.1 Input variables

`A, a, B, b`: these are the matrix A and $B \in \mathbb{R}^{m\times n}$ and the vectors a and $b \in \mathbb{R}^m$ defining the function H in (2.18). The dimensions m and n of the problem are deduced from the size of these variables.

`x`: the point $x \in \mathbb{R}^n$ at which the B-differential must be computed.

`options`: structure array allowing the user to tune the behavior of `bdiffmin`. All options have default values, so that the `options` argument can be omitted or set to `[]`. The following options are available (they are given in alphabetic order of the field names).

`options.bdiffc`: logical, default `false`. If `true` the matrices that are in the set (4.1), defined below, are listed in `info.bdiffc`.

`options.eqtol`: positive real number, default `1.e-8`. Small value used to detect equality of the components of $Ax + a$ and $Bx + b$. If

$$|(Ax + a)_i - (Bx + b)_i| \leqslant \texttt{options.eqtol},$$

the ith components of $Ax + a$ and $Bx + b$ are considered to be equal.

`options.fout`: integer, default `1`. Output channel containing all the printings made during the run of `bdiffmin` (nothing is printed from `isf`). Set 1 (default) for the standard output (screen). For a specific file, use `options.fout = fopen(...)` before calling `bdiffmin`;

`options.verb`: integer in $[0:\infty]$ (default 1). Verbosity level of the output channel `options.fout`.

$= 0$: `bdiffmin` works silently;
$\geqslant 1$: `bdiffmin` prints error messages, initial setting, final status on the channel `options.fout`.

4.1.2 Output variable

`info`: Structure array giving the output of `bdiffmin`, as well as information on the run. The following fields are specific to `bdiffmin`; other fields are inherited from `isf`.

`info.bdiff`: cell array containing all the Jacobians in $\partial_B H(x) \subseteq \mathbb{R}^{m\times n}$.

`info.bdiffc`: (if `options.bdiffc` is `true`) cell array containing all the matrices in

$$\Big(\partial_B H_1(x) \times \cdots \times \partial_B H_m(x)\Big) \setminus \partial_B H(x). \tag{4.1}$$

`info.flag`: integer in $[0:11]$ giving the diagnosis of the run.

$= 0$: the required job has been realized;
$= 11$: an input argument of `bdiffmin` is wrong.

The other values of `info.flag` are inherited from those of the function `isf`.

4.2 Example of use

Consider the following example that is the one considered in [7, section 5.2.9]:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix} \quad \text{and} \quad a = b = x = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

and run `bdiffmin` with that data and the following options:

```
options.verb   = 0;
options.bdiffc = true;                          % to get the non-Jacobians
[info]         = bdiffmin(A,a,B,b,x,options);
```

Then, `bdiffmin` prints nothing on the standard output. It forms the matrix V in (2.27), which reads (note that $\mathcal{E}^{\neq}(x) = [1:3]$)

$$V = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Next, `bdiffmin` runs `isf` which is asked to work silently (in particular, nothing is printed on the standard output). Let us examine the output structure `info` on return from `bdiffmin` to have information on the run and on the computed values.

- Since `info.flag` is `0`, the run is successful.
- The cell array `info.bdiff` contains all the Jacobians of the B-differential of $x \mapsto \min(Ax + a, Bx + b)$ at the given x. If one executes

  ```
  for i = 1:length(info.bdiff)
    fprintf('Jacobian %i\n',i)
    disp(info.bdiff{i});
  end
  ```

  one gets the 6 Jacobians in $\partial_B H(x)$:

  ```
  Jacobian 1
       1     0     0
       0     1     0
       0     0     1
  Jacobian 2
       1     0     0
       0     2     1
       0     0     1
  Jacobian 3
       1     0     0
       0     2     1
       1     1     2
  Jacobian 4
       2     0     0
       0     1     0
       0     0     1
  Jacobian 5
       2     0     0
  ```

```
     0     1     0
     1     1     2
Jacobian 6
     2     0     0
     0     2     1
     1     1     2
```

- Since `bdiffmin` was run with `options.bdiffc` set to `true`, the cell array `info.bdiffc` contains all the matrices that are in the set (4.1). If one executes

```
for i = 1:length(info.bdiffc)
  fprintf('Matrix %i\n',i)
  disp(info.bdiffc{i});
end
```

one gets the 2 matrices in that set:

```
Matrix 1
     1     0     0
     0     1     0
     1     1     2
Matrix 2
     2     0     0
     0     2     1
     0     0     1
```

References

[1] Marcelo Aguiar, Swapneel Mahajan (2017). *Topics in hyperplane Arrangements*. Mathematical Surveys and Monographs 226. American Mathematical Society, Providence, RI. [doi]. 8

[2] David Avis, Komei Fukuda (1996). Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3), 21–46. [doi]. 3, 8

[3] Pierre Baldi, Roman Vershynin (2019). Polynomial threshold functions, hyperplane arrangements, and random tensors. *SIAM Journal on Mathematics of Data Science*, 1(4), 699–729. [doi]. 3

[4] J.M. Borwein, A.S. Lewis (2006). *Convex Analysis and Nonlinear Optimization – Theory and Examples* (second edition). CMS Books in Mathematics 3. Springer, New York. 6, 9

[5] F.H. Clarke (1983). *Optimization and Nonsmooth Analysis*. John Wiley & Sons, New York. Reprinted in 1990 by SIAM, Classics in Applied Mathematics 5 [doi]. 9

[6] J.-P. Dussault, J.Ch. Gilbert, B. Plaquevent-Jourdain (2023). On the B-differential of the componentwise minimum of two affine vector functions. *Mathematical Programming Computation* (submitted). [hal-04048393]. 3, 5, 7, 9, 10, 13, 14

[7] J.-P. Dussault, J.Ch. Gilbert, B. Plaquevent-Jourdain (2023). On the B-differential of the componentwise minimum of two affine vector functions – The full report. Research report. [hal-03872711]. 6, 19

[8] H. Edelsbrunner, J. O'Rourke, R. Seidel (1986). Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal on Control*, 15(2), 341–363. [doi]. 8

[9] Herbert Edelsbrunner (1987). *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin. [doi]. 8

[10] F. Facchinei, J.-S. Pang (2003). *Finite-Dimensional Variational Inequalities and Complementarity Problems* (two volumes). Springer Series in Operations Research. Springer. 9

[11] Rick Greer (1984). *Trees and Hills: Methodology for Maximizing Functions of Systems of Linear Relations*. Annals of Discrete Mathematics (Mathematical Studies 96) 22. North-Holland. 6

[12] Branko Grünbaum (1972). *Arrangements and Spreads*. Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics 10. AMS, Providence, RI. 8

[13] Dan Halperin, Micha Sharir (2018). *Arrangements*, chapter 28, pages 723–762. Discrete Mathematics its Applications. CRC Press - Taylor & Francis Group. 8

[14] J.-B. Hiriart-Urruty, C. Lemaréchal (2001). *Fundamentals of Convex Analysis*. Springer. 9

[15] James Oxley (2011). *Matroid Theory* (second edition). Oxford Graduate Texts in Mathematics 21. Oxford University Press, Oxford. [doi]. 5

[16] Liqun Qi (1993). Convergence analysis of some algorithms for solving nonsmooth equations. *Mathematics of Operations Research*, 18, 227–244. [doi]. 9

[17] Liqun Qi, Jie Sun (1993). A nonsmooth version of Newton's method. *Mathematical Programming*, 58, 353–367. [doi]. 9

[18] Miroslav Rada, Michal Černý (2018). A new algorithm for enumeration of cells of hyperplane arrangements and a comparison with Avis and Fukuda's reverse search. *SIAM Journal on Discrete Mathematics*, 32(1), 455–473. [doi]. 13, 14

[19] H. Rademacher (1919). Über partielle und totale differenzierbarkeit. *I. Math. Ann.*, 89, 340–359. 9

[20] Samuel Roberts (1887/88). On the figures formed by the intercepts of a system of straight lines in a plane, and on analogous relations in space of three dimensions. *Proc. London Math. Soc.*, 19, 405–422. [doi]. 8

[21] R.T. Rockafellar (1970). *Convex Analysis*. Princeton Mathematics Ser. 28. Princeton University Press, Princeton, New Jersey. 9

[22] L. Schläfli (1950). Theorie der vielfachen Kontinuität (in German). In *Gesammelte mathematische Abhandlugen,* Band 1, pages 168–387. Springer, Basel. [doi]. 5

[23] Nora Helena Sleumer (1998). Output-sensitive cell enumeration in hyperplane arrangements. In *Algorithm theory-SWAT'98 (Stockholm)*, Lecture Notes in Comput. Sci. 1432, pages 300–309. Springer, Berlin. [doi]. 8

[24] Nora Helena Sleumer (2000). *Hyperplane arrangements – Construction, visualization and application*. PhD Thesis, Swiss Federal Institute of Technology, Zurich, Switzerland. [doi]. 8

[25] Marek J. Śmietánski (2019). On a new exponential iterative method for solving nonsmooth equations. *Numerical Linear Algebra with Applications*, 25. [doi]. 9

[26] R.P. Stanley (2007). An introduction to hyperplane arrangements. In Ezra Miller, Victor Reiner, Bernd Sturmfels (editors), *Geometric Combinatorics*, pages 389–496. [doi]. 8

[27] J. Steiner (1826). Einige Gesetze über die Theilung der Ebene und des Raumes. *Journal für die Reine und Angewandte Mathematik*, 1, 349–364. [doi]. 8

[28] Michal Černý, Miroslav Rada, Jaromír Antoch, Milan Hladík (2022). A class of optimization problems motivated by rank estimators in robust regression. *Optimization*, 71(8), 2241–2271. [doi]. 8

[29] Robert O. Winder (1966). Partitions of N-space by hyperplanes. *SIAM Journal on Applied Mathematics*, 14(4), 811–818. [doi]. 3, 4

Index