IncSignFeas.jl (isf): Julia functions for the enumeration of the chambers of hyperplane arrangements

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

(Wednesday 3rd September, 2025, 18:36)

The Julia function `isf` aims at listing the chambers of a hyperplane arrangement. By listing, we mean identifying the *sign vectors* corresponding to each of the (nonempty) chambers. Several variants of a main algorithm are available, defined by different parameters that can be changed and tuned. This code extends the Matlab solver `isf.m`, which is restricted to central arrangements. In this case, this problem has many equivalent reformulations that can thus be also dealt with by processing the output.

Keywords: Duality • Hyperplane arrangement • Matroid circuit • Motzkin's alternative • Stem vector • Strict affine inequality system • Tree algorithm

AMS MSC 2020: 05B35, 05C05, 14N20, 49N15, 52B40, 52C35, 52C40, 90C05.

[†]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, Serena team, 48 rue Barrault, CS 61534, 75647 Paris Cedex, 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, Serena team, 48 rue Barrault, CS 61534, 75647 Paris Cedex, France (e-mail: `Baptiste.Plaquevent-Jourdain@inria.fr`. ORCID 0000-0001-7055-4568).

Table of contents

1 Introduction

This document is a description of the Julia code `isf` available on Github. See section 4 for uses of the code. Its name stands for "incremental signed feasibility" ("`IncSignFeas`" on Github and in Julia). Its goal is to compute the chambers of hyperplane arrangements, by using algorithms based on the tree structure proposed by Rada and Černý [14]. The code described in this notice is highly related to [8] (and [9]) that uses the results produced by `isf`. It follows [6, 7] and their related Matlab code [4], described in [5], which focus on *linear* hyperplane arrangements (all the hyperplanes have the origin in common). Such arrangements are equivalent to various other problems, thus also solvable by the Julia code after processing its output.

After recalling some useful notation, section 2 presents some properties and theoretical elements useful to understand the code. Section 3 presents the main tasks faced by the code and the techniques it uses. Finally, section 4 presents how to use `isf` on some simple examples, while section 5 discusses more advanced information about the code itself.

Notation

One denotes by $\mathbb{N}$ and $\mathbb{R}$ the sets of nonnegative integers and real numbers and one sets $\mathbb{N}^* := \mathbb{N} \setminus \{0\}$ and $\mathbb{R}^* := \mathbb{R} \setminus \{0\}$ ($r \in \mathbb{R}$ is said to be *positive* if $r > 0$ and *nonnegative* if $r \geqslant 0$). For two integers $n_1 \leqslant n_2$, $[n_1 : n_2] := \{n_1, \dots, n_2\}$ is the set of the integers between n_1 and n_2. We denote by $\mathbb{R}^n_+ := \{x \in \mathbb{R}^n : x \geqslant 0\}$ and $\mathbb{R}^n_{++} := \{x \in \mathbb{R}^n : x > 0\}$ the nonnegative and positive orthants, where the inequalities apply componentwise. For a set S, one denotes by $|S|$ its cardinality, by S^c its complement in a set that will be clear from the context and by S^J, for an index set $J \subseteq \mathbb{N}^*$, the set of vectors, whose elements are in S and are indexed by the indices in J. The disjoint union of sets is denoted by $\cup$. The vector e denotes the vector of all ones, whose size depends on the context. The Hadamard product of u and $v \in \mathbb{R}^n$ is the vector $u \bullet v \in \mathbb{R}^n$, whose ith component is $u_i v_i$. The sign function $\mathrm{sgn} : \mathbb{R} \to \mathbb{R}$ is defined by $\mathrm{sgn}(t) = +1$ if $t > 0$, $\mathrm{sgn}(t) = -1$ if $t < 0$ and $\mathrm{sgn}(0) = 0$. The sign of a vector x is defined componentwise: $\mathrm{sgn}(x)_i = \mathrm{sgn}(x_i)$. The dimension of a space $\mathbb{E}$ is denoted by $\dim(\mathbb{E})$, the range space of a matrix $A \in \mathbb{R}^{m \times n}$ by $\mathcal{R}(A)$, its null space by $\mathcal{N}(A)$, its rank by $\mathrm{rank}(A) := \dim \mathcal{R}(A)$ and its nullity by $\mathrm{null}(A) := \dim \mathcal{N}(A) = n - \mathrm{rank}(A)$ thanks to the rank-nullity theorem. The ith row (resp. column) of A is denoted by $A_{i,:}$ (resp. $A_{:,i}$). Transposition operates after a row and/or column selection: $A^\mathsf{T}_{i,:}$ is a short notation for $(A_{i,:})^\mathsf{T}$ for instance. The vertical concatenation of matrices $A \in \mathbb{R}^{n_1 \times m}$ and $B \in \mathbb{R}^{n_2 \times m}$ is denoted by $[A; B] \in \mathbb{R}^{(n_1+n_2) \times m}$. For $u \in \mathbb{R}^n$, $\mathrm{Diag}(u) \in \mathbb{R}^{n \times n}$ is the square diagonal matrix with $\mathrm{Diag}(u)_{i,i} = u_i$.

2 Setting presentation

2.1 Sign vectors

Let $n \in \mathbb{N}^*$. A hyperplane of $\mathbb{R}^n$ is a set of the form $H := \{x \in \mathbb{R}^n : v^\mathsf{T} x = \tau\}$, where $v \in \mathbb{R}^n$ and $\tau \in \mathbb{R}$. A hyperplane H partitions $\mathbb{R}^n$ into three subsets: H itself, its negative and positive open halfspaces, respectively defined by

$$H^- := \{x \in \mathbb{R}^n : v^\mathsf{T} x < \tau\} \qquad \text{and} \qquad H^+ := \{x \in \mathbb{R}^n : v^\mathsf{T} x > \tau\}.$$

In what follows, we assume that all the hyperplanes considered are *proper*, in the sense

that $v \neq 0$; improper hyperplanes do not split the space in three parts[1].

A *hyperplane arrangement* is a collection of $p \in \mathbb{N}^*$ hyperplanes $H_i := \{x \in \mathbb{R}^n : v_i^\mathsf{T} x = \tau_i\}$, for $i \in [1\,:p]$, where $v_1, \ldots, v_p \in \mathbb{R}^n \setminus \{0\}$ and $\tau_1, \ldots, \tau_p \in \mathbb{R}$. It is denoted by $\mathcal{A}(V, \tau)$, where $V := [v_1 \ \cdots \ v_p] \in \mathbb{R}^{n\times p}$ is the matrix made of the vectors v_i's and $\tau := [\tau_1; \ldots; \tau_p] \in \mathbb{R}^p$. The arrangement is said to be *linear* if $\tau = 0$ and *affine* in general (therefore, a linear arrangement is just a particular affine arrangement). The arrangement is said to be *centered* if all the hyperplanes have a point in common [1], which is the case if and only if $\tau \in \mathcal{R}(V^\mathsf{T})$ [8; section 3.2].[2]

Whilst a hyperplane divides $\mathbb{R}^n$ into two nonempty open halfspaces, a hyperplane arrangement splits $\mathbb{R}^n$ into nonempty polyhedral convex open sets, called *chambers*. This is illustrated in figure 2.1 by three elementary examples.

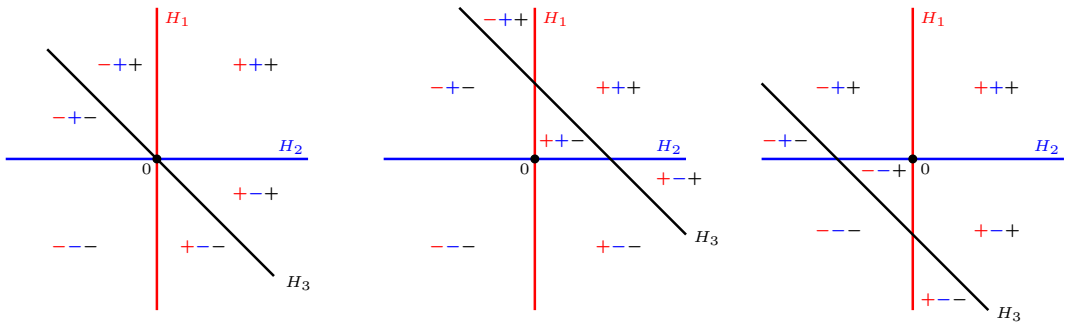

Figure 2.1: Arrangements in $\mathbb{R}^2$ specified by the hyperplanes $H_1 := \{x \in \mathbb{R}^2 : x_1 = 0\}$, $H_2 := \{x \in \mathbb{R}^2 : x_2 = 0\}$, $H_3(\text{left}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = 0\}$, $H_3(\text{middle}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = 1\}$ and $H_3(\text{right}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = -1\}$. The origin is contained in all the hyperplanes but in H_3(middle) and H_3(right), so that the arrangement in the left-hand side is *linear* with 6 chambers and the other ones are *affine* with 7 chambers.

In an arrangement, the chambers can be labelled by a sign vector $s \in \{\pm 1\}^p$, each s_i corresponding to hyperplane H_i: $s_i = +1$ (resp. -1) if the chamber is in H_i^+ (resp. H_i^-). We thus aim at computing a certain subset of $\{-1, +1\}^p$. The problem solved by `isf` is the following one.

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

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

Definitions 2.2 (symmetry in $\{-1, +1\}^p$) Let $p \in \mathbb{N}^*$ and $S \subseteq \{-1, +1\}^p$. An element $s \in S$ is said to be symmetric (in S) if $-s \in S$ and asymmetric otherwise. The set S itself is said to be symmetric if all its elements are symmetric and asymmetric otherwise.

[1] If $v = 0$, then one has the following identities:

$$\begin{array}{lcccc} \text{if } \tau < 0 & : & H^- = \varnothing, & H = \varnothing, & H^+ = \mathbb{R}^n, \\ \text{if } \tau = 0 & : & H^- = \varnothing, & H = \mathbb{R}^n, & H^+ = \varnothing, \\ \text{if } \tau > 0 & : & H^- = \mathbb{R}^n, & H = \varnothing, & H^+ = \varnothing, \end{array}$$

which can be summarized by $H^{-\operatorname{sgn}(\tau)} = \mathbb{R}^n$ (recall that $\operatorname{sgn}(0) = 0$) and the other two sets are empty.

[2] Linear and affine arrangements are not extremely different in nature, see [11; definition 1.15] for instance.

Definition 2.3 (symmetry in $\mathcal{S}(V,\tau)$) The symmetric and asymmetric parts of $\mathcal{S}(V,\tau)$ are denoted and defined by

$$\mathcal{S}_s(V,\tau) := \mathcal{S}(V,\tau) \cap [-\mathcal{S}(V,\tau)] \qquad \text{and} \qquad \mathcal{S}_a(V,\tau) = \mathcal{S}(V,\tau) \setminus \mathcal{S}_s(V,\tau). \tag{2.2}$$

When $\mathcal{S}(V,\tau)$ is symmetric, it suffices to identify half of it and take the opposite of each sign vector to get $\mathcal{S}(V,\tau)$ entirely.

In what follows, unless specified otherwise we always assume that the setting of problem 2.1 holds, i.e., n and p are in $\mathbb{N}^*$, $V \in \mathbb{R}^{n\times p}$ with nonzero columns and $\tau \in \mathbb{R}^p$. An arrangement is said to be in *general position* if the following equivalent properties hold ([9; proposition 3.33]):

$$\forall\ I \subseteq [1:p]: \quad \begin{cases} \cap_{i\in I} H_i \neq \varnothing \text{ and } \dim(\cap_{i\in I} H_i) = n - |I| & \text{if } |I| \leqslant r \\ \cap_{i\in I} H_i = \varnothing & \text{if } |I| \geqslant r+1, \end{cases}$$

$$\forall\ I \subseteq [1:p]: \quad \begin{cases} \operatorname{rank}(V_{:,I}) = |I| & \text{if } |I| \leqslant r \\ \operatorname{rank}([V;\tau^{\mathsf{T}}]_{:,I}) = r+1 & \text{if } |I| \geqslant r+1, \end{cases}$$

$$\forall\ I \subseteq [1:p]: \quad \begin{cases} \operatorname{rank}(V_{:,I}) = \min(|I|, r) \\ \operatorname{rank}([V;\tau^{\mathsf{T}}]_{:,I}) = \min(|I|, r+1). \end{cases}$$

The first characterization is geometric (it uses the hyperplanes) while the second and third are algebraic (they use the pair (V,τ)). The general position of a linear arrangement is defined slightly differently [17] [9; definition 3.28].

We now give a few of the main properties observed on such sign vector sets $\mathcal{S}(V,\tau)$. In the formulation given below, most of them are proven in [9]. The first two formulae of point 7 are rewritings of Winder's formula [17] (see also [18; corollary 5.9, p. 68]) and the last one is taken from Zaslavsky [18; section 2.A, Theorem A].

Proposition 2.4 (some properties of $\mathcal{S}(V,\tau)$) *Let $V \in \mathbb{R}^{n\times p}$ with nonzero columns and $\tau \in \mathbb{R}^p$. The following properties hold.*

1) $\mathcal{S}(V,\tau) \neq \varnothing$.
2) $\mathcal{S}(V,\tau) = \{\pm 1\}^p \Leftrightarrow \operatorname{rank}(V) = p$ *(V is injective).*
3) $\mathcal{S}(V,0)$ *is symmetric, i.e.,* $-\mathcal{S}(V,0) = \mathcal{S}(V,0)$.
4) $\mathcal{S}(V,-\tau) = -\mathcal{S}(V,\tau)$.
5) $\mathcal{S}(V,\tau)$ *is symmetric if and only if* $\tau \in \mathcal{R}(V^{\mathsf{T}})$.
6) $|\mathcal{S}(V,\tau)| \leqslant \sum_{i=0}^{r} \binom{p}{i}$ *where r is the rank of V. This upper bound is attained when the hyperplanes are in general position.*
7) $|\mathcal{S}(V,\tau)| = \sum_{J\subseteq[1:p],\tau_J\in\mathcal{R}(V_{:,J}^{\mathsf{T}})} (-1)^{\operatorname{null}(V_{:,J})} = (|\mathcal{S}(V,0)| + |\mathcal{S}([V;\tau^{\mathsf{T}}],0)|)/2 = (-1)^n \chi(-1)$ *where χ is the caracteristic polynomial of the arrangement.*

As suggested in the last point of this proposition, the matrix $[V;\tau^{\mathsf{T}}]$, of size $(n+1)\times p$, can be useful to compute $\mathcal{S}(V,\tau)$. This is mentioned in section 3.4 and detailed in [8, 9; sections 3.4 and 6], in algorithmic variants called *compact*, that compute only $\mathcal{S}_a(V,\tau)$ and half of $\mathcal{S}_s(V,\tau)$.

2.2 Stem vectors

It was observed in [6, 8] that the determination of $\mathcal{S}(V,\tau)$ can be tackled by duality, that is by using Motzkin's theorem of the alternative [10]. This led to the notion of *stem vector*, identical to the signed circuits [19] when $\tau = 0$.

Let us start by recalling the notion of *circuit* of the *vector matroid* formed by the columns of V and its subsets of linearly independent columns [12]. The set of circuits of V is denoted and defined by

$$\mathcal{C} := \{J \subseteq [1:p] : J \neq \varnothing, \operatorname{null}(V_{:,J}) = 1, V_{:,J_0} \text{ is injective for all } J_0 \subsetneq J\}. \tag{2.3}$$

Definition 2.5 (stem vector) A *stem vector* of the arrangement $\mathcal{A}(V,\tau)$ is a sign vector $\sigma \in \{\pm 1\}^J$ for some $J \in \mathcal{C}(V)$ satisfying

$$\sigma = \operatorname{sgn}(\eta) \text{ for some } \eta \in \mathbb{R}^J \text{ verifying } \eta \in \mathcal{N}(V_{:,J}) \backslash \{0\} \text{ and } \tau_J^\mathsf{T} \eta \geqslant 0.$$

It is said to be symmetric if $\tau_J^\mathsf{T}\eta = 0$ and asymmetric otherwise. The set of stem vectors is denoted by $\mathfrak{S}(V,\tau)$, its subset of symmetric stem vectors by $\mathfrak{S}_s(V,\tau)$ and the one of asymmetric stem vectors by $\mathfrak{S}_a(V,\tau)$.

In particular, if σ is a symmetric stem vector, then $-\sigma$ is also a symmetric stem vector. We mention a few properties, mostly taken from [9].

Proposition 2.6 (some properties of $\mathfrak{S}(V,\tau)$) *Let $V \in \mathbb{R}^{n\times p}$ with nonzero columns, $\tau \in \mathbb{R}^p$ and let $r = \operatorname{rank}(V)$. The following properties hold.*

1) *The circuits are independent of τ but not the stem vectors.*
2) *Let J be a circuit as in definition 2.5. Since the nullity of $V_{:,J}$ is 1, its null space is of the form $\mathbb{R}\eta$ for some $\eta \in \mathbb{R}^J \setminus \{0\}$. By the second part of the definition, $\eta_j \neq 0$ for all $j \in J$ (otherwise, $J_0 := J \setminus \{j\}$ would contradict the definition) so $\sigma_j \in \{-1,+1\}$.*
3) *For a given circuit J, there is a unique asymmetric stem vector σ or a unique pair of opposite symmetric stem vectors $\pm\sigma$, independent of the chosen $\eta \in \mathcal{N}(V_{:,J})$.*
4) *The stem vectors and the circuits may be of different sizes.*
5) *The number of circuits, $|\mathcal{C}(V)|$, which is equal to $|\mathfrak{S}_a(V,\tau)|+|\mathfrak{S}_s(V,\tau)|/2$, is upper bounded by $\binom{p}{r+1}$, which is also attained if and only if $\mathcal{A}(V,\tau)$ is in general position.*
6) *Stem vectors are not elements of groups or vector spaces, so there are no operation on them except taking their opposite, since they are elements with coordinates in $\{-1,+1\}$.*
7) *One has $\mathfrak{S}(V,-\tau) = -\mathfrak{S}(V,\tau)$, $\mathfrak{S}_a(V,-\tau) = -\mathfrak{S}_a(V,\tau)$ and $\mathfrak{S}_s(V,-\tau) = \mathfrak{S}_s(V,\tau)$.*
8) *Let $J = \{j,k\}$ for $j,\ k \in [1:p]$. J is a circuit of size 2 if and only if v_j and v_k are colinear. The circuits of V have size $\geqslant 2$.*[3]

The key property of stem vectors, in the context of identifying $\mathcal{S}(V,\tau)$, is the following [8; proposition 3.9]:

$$s \in \mathcal{S}(V,\tau)^c \quad \Longleftrightarrow \quad s_J = \sigma \text{ for some } \sigma \in \mathfrak{S}(V,\tau). \tag{2.4}$$

Therefore, determining whether s is infeasible (does not belong to $\mathcal{S}(V,\tau)$) can be done as follows.

Definition 2.7 (covering a stem vector) Let $V \in \mathbb{R}^{n\times p}$ with nonzero columns, $\tau \in \mathbb{R}^p$ and $\mathfrak{S}(V,\tau)$ be the set of stem vectors. Let $k \in [1:p]$ and $s \in \{\pm 1\}^k$ be a sign vector of size k and $\sigma \in \mathfrak{S}(V,\tau)$ be a stem vector and J its associated circuit as described in definition 2.5. We say that s covers the stem vector σ if $s_J = \sigma$, or equivalently that σ is covered by s.

Observe that the covering of $\sigma \in \mathfrak{S}(V,\tau)$ with indices in J can occur only if $J \subseteq [1:k]$ for a given $s \in \{\pm 1\}^k$.

[3] Circuits of size 1 would correspond to zero columns – improper hyperplanes, which is excluded.

3 Algorithmic issues

The Julia code solves problem 2.1. It constructs a tree, the $\mathcal{S}$-tree, whose level k corresponds to *feasible* sign vectors s of size k, meaning that $s \in \mathcal{S}(V_{:,[1:k]}, \tau_{[1:k]})$. Thus, the leaves of the $\mathcal{S}$-tree are the sign vectors of $\mathcal{S}(V, \tau)$. This principle is illustrated in [8; figure 4.1] recalled below for convenience. This tree construction leads to the name "Incremental Sign Feasibility" ("IncSignFeas", "ISF") close to "Inc(remental) Enu(meration)" in [14]. For a quick launch of the code, see section 4.

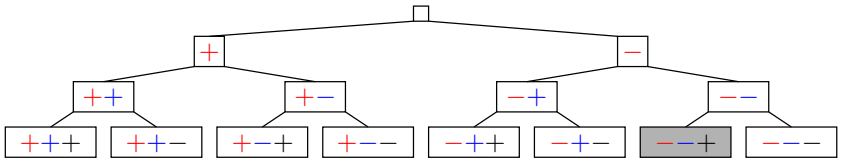

Figure 3.1: $\mathcal{S}$-tree of the arrangement in the middle pane of figure 2.1. The gray node is actually absent from the tree, since there is no chamber associated with $s = (-1, -1, +1)$ (no x such that $\mathrm{Diag}(s)(V^\mathsf{T}x - \tau) > 0$).

The algorithm proceeds recursively by computing the descendants of a node $s \in \{\pm 1\}^k$ (when $k < p$). It proceeds by depth-first search, thus exploring the whole branch starting at $+$ before the one starting at $-$ (and the same, recursively, in each node).

The next sections presents two possibilities to detect the descendants of $s \in \{\pm 1\}^k$.

3.1 Descendants by optimization

The first possibility is to verify whether the systems corresponding to $(s, +)$ and $(s, -)$ have a solution, i.e., if there exists a point x^+ in the chamber of $(s, +)$ and a point x^- in the chamber of $(s, -)$. These systems read

$$\left\{ \begin{array}{ll} s \bullet (V_{:,[1:k]}^\mathsf{T} x - \tau_{[1:k]}) & > 0 \\ +(v_{k+1}^\mathsf{T} x - \tau_{k+1}) & > 0 \end{array} \right. \quad \text{and} \quad \left\{ \begin{array}{ll} s \bullet (V_{:,[1:k]}^\mathsf{T} x - \tau_{[1:k]}) & > 0 \\ -(v_{k+1}^\mathsf{T} x - \tau_{k+1}) & > 0. \end{array} \right. \tag{3.1}$$

Since $s \in \mathcal{S}(V_{:,[1:k]}, \tau_{[1:k]})$, there exists a point x^s verifying the first k inequalities of these systems. Then, define $s_{k+1} := \mathrm{sgn}(v_{k+1}^\mathsf{T} x^s - \tau_{k+1})$. If $s_{k+1} = 0$, then x^s belongs to H_{k+1}, so that small perturbation of x^s verify each of the systems in (3.1). Otherwise, $x^s \in H_{k+1}^{s_{k+1}}$, so x^s verifies one of the two systems in (3.1), and one must check whether the remaining system has a solution. This question can be answered by solving the linear optimization problem (LOP) [14], [8; (4.5)]

$$\begin{array}{rl} \min_{(x,\alpha) \in \mathbb{R}^n \times \mathbb{R}} & \alpha \\ \text{s.t.} & s_i(v_i^\mathsf{T} x - \tau_i) + \alpha \geqslant 0, \quad i \in [1:k] \\ & -s_{k+1}(v_{k+1}^\mathsf{T} x - \tau_{k+1}) + \alpha \geqslant 0 \\ & \alpha \geqslant -1, \end{array} \tag{3.2}$$

where $(s, -s_{k+1}) \in \mathcal{S}(V_{:,[1:k+1]}, \tau_{[1:k+1]})$ when $\alpha < 0$.

By construction, there is one LOP to solve at each node of the tree except for the leaves, which correspond to sign vectors of size p, when all the hyperplanes have been considered. Once the descendants are confirmed in $\mathcal{S}(V_{:,[1:k+1]}, \tau_{[1:k+1]})$, the recursion starts again with these nodes except if $k + 1 = p$, in which case the recursion stops.

The algorithms using linear optimization and points x to verify the systems of inequalities have solutions are called *primal*. The algorithms presented next have some *dual* flavor, which comes from the dual nature of Motzkin's alternative [10].

3.2 Descendants by stem vectors

3.2.1 Detecting descendants

As it is suggested in the end of section 2, the infeasibility of a sign vector can be detected by the stem vectors. This is formalized in the following definition.

Definition 3.1 (covering test) *Let $V \in \mathbb{R}^{n\times p}$ with nonzero columns, $\tau \in \mathbb{R}^p$ and $\mathfrak{S}(V,\tau)$ be the set of stem vectors. Let $k \in [1:p]$ and $s \in \{\pm 1\}^k$ be a sign vector of size k. The covering test for s is the operation that verifies whether there exists some $\sigma \in \mathfrak{S}(V,\tau)$ and a circuit J such that $s_J = \sigma$. It stops as soon as such a stem vector is found.*

In particular, this approach does *not* require to introduce points $x \in \mathbb{R}^n$ verifying the systems. One simply performs a covering test from definition 3.1 for $(s,+)$ then one for $(s,-)$.

It can be shown that, since a sign vector s of size $< p$ always has at least one descendant, if the covering test of $(s,+)$ indicates it is infeasible, then $(s,-)$ is necessarily feasible. Thus there are one or two covering tests to perform per node of the tree (except for the leaves).

3.2.2 Obtaining all the stem vectors

The method of covering tests requires to know the set $\mathfrak{S}(V,\tau)$. By definition 2.5, we compute the circuits and their associated stem vectors. This task is performed by a simple algorithm (which may be slow for large arrangements) given for instance in [15; for $\mathcal{C}(V)$] or [9; algorithm 5.1-5.2].

The algorithm is relatively simple: it explores a tree of subsets of $[1:p]$ where a node corresponds to a subset J and its descendants are $J \cup \{k\}$ for $k > \max(J)$ (to avoid studying the same subsets multiple times), and stops the exploration of a subtree when its subset contains a circuit (which may be larger than the circuit). Let us mention that it can detect the same circuit multiple times, meaning that the duplicates must be removed at the end.

Every time a circuit is found, the corresponding asymmetric stem vector or pair of opposite symmetric stem vectors is computed and stored.

3.3 Primal-dual version

The fully dual version of section 3.2 has two weaknesses: (1) computing the stem vectors may be time-consuming, and (2) when a descendant exists, every stem vector has to be tested in the covering test, which can also require much computing time.

An improvement consists in the primal-dual method, which combines both primal and dual techniques, as described in [8; section 5.2]. It proceeds as follows. At the start, it is a primal method, except that whenever a descendant is declared infeasible, the algorithm obtains a stem vector thanks to a dual solution [2] of the optimization problem (3.2). The algorithm thus collects *some* stem vectors along the way. With a partial list of stem vectors (growing during the run), the algorithm can detect *some* infeasible sign vectors by the covering tests like previously.

Since not all stem vectors are known, once a covering test fails (s may be feasible), a LOP is solved to check whether the sign vector is actually feasible. If not, a new stem vector is obtained.

3.4 Compaction

As mentioned at the end of section 2.1, it is possible to compute $\mathcal{S}_a(V,\tau)$ and half of $\mathcal{S}_s(V,\tau)$ instead of $\mathcal{S}(V,\tau) = \mathcal{S}_a(V,\tau) \uplus \mathcal{S}_s(V,\tau)$.

The details are a bit involved and tedious, so we only give the principles and refer to [9; section 6] (and [13; section A.3.3]) for additional precisions. A compact algorithm only changes the structure of the constructed tree, and can thus be applied to the primal (section 3.1), dual (section 3.2) or primal-dual (section 3.3) methods.

A compact $\mathcal{S}$-tree computes half of the normal $\mathcal{S}$-tree starting at $+$ (or $-$) like when the arrangement is linear, and adds to it the opposites of asymmetric sign vectors that would have been found in the other half of the tree starting at $-$ (or $+$). An example is shown below, for the arrangements in the middle and right panes of figure 2.1.

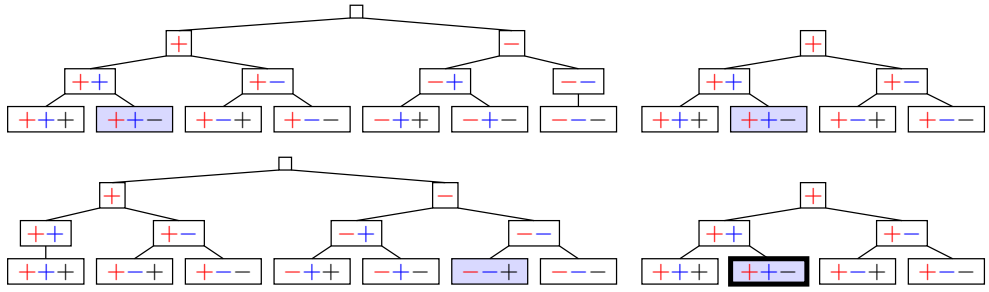

Figure 3.2: Standard $\mathcal{S}$-trees (left) and compact $\mathcal{S}$-trees (right) of the arrangements in the middle pane (above, compare with figure 3.1) and the right-hand side pane (below) of figure 2.1. The sign vectors in the white boxes are in $\mathcal{S}_s(V,\tau)$, those in the blue boxes are in $\mathcal{S}_a(V,\tau)$ and the one in the blue box with bold edges is in $\mathcal{S}_a(V,-\tau)$; this last sign vector must be multiplied by -1 to get a sign vector in $-\mathcal{S}_a(V,-\tau) = \mathcal{S}_a(V,\tau) \subseteq \mathcal{S}(V,\tau)$.

In the bottom right, the node with bold edges was grafted (thus itself and its descendants must be multipled by -1 at the end) to the left subtree of the left half of the bottom left tree, to ensure that the asymmetric part of the arrangement is computed properly. In the upper half of the picture, asymmetry is kept in mind (the light blue of the node) but no need for a graft.

4 Using `isf`

`IncSignFeas` (abbreviated in `isf`) is a Julia package aimed at enumerating the chambers of an hyperplane arrangement (knowing the sign vectors of the chambers).

This package uses a tree structure introduced by Rada and Černý [14], the $\mathcal{S}$-tree of section 3. Several variants of this tree algorithm are proposed (see section 3.1, 3.2, 3.3, 3.4), see here [8] for additional details.

4.1 Package installation

The package can be loaded in Julia with the following instructions.

```
using Pkg
Pkg.add("IncSignFeas")
using IncSignFeas
```

The code uses the Julia packages HiGHS (see here) and JuMP for linear optimization, though some algorithms do not use linear optimization solvers (section 3.2).

4.2 Example of use

Once `IncSignFeas` is active, one must specify the arrangement that will be considered by `isf`. It is defined by a matrix $[V; \tau^\mathsf{T}]$. The various options of the code are detailed in the next section. For now, we focus on a simple example.

Consider the small arrangement in dimension 2 with 4 hyperplanes given by:

$$V = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & -1 \end{bmatrix}, \quad \tau = [0; 0; 1; 0]. \tag{4.1}$$

It is represented in figure 4.1.

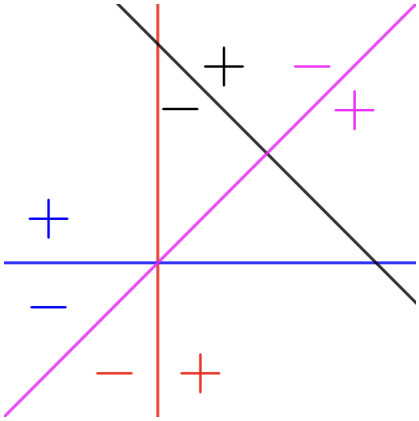

Figure 4.1: The arrangement (4.1). The signs represent the halfspaces.

The chambers of the arrangement (4.1) can be obtained by the following instructions. For the algorithm numbers (the first inputs of `options_from_algo` like 13 here), see table 5.1.

```
Vt = [1 0 1 1 ; 0 1 1 -1 ; 0 0 1 0];     # define the arrangement
options = options_from_algo(13, false);  # suggested algorithm
info = isf(Vt, options);                 # compute the chambers
info.ns
info.s
```

where `false` means $\tau \neq 0$. The last two instructions should have the following outputs

```
info.ns   # the number of chambers
10
info.s    # the set of chambers
10-element Vector{Vector{Int64}}:
[1, 1, 1, 1]
[-1, -1, -1, -1]
[1, 1, 1, -1]
[-1, -1, -1, 1]
[1, 1, -1, 1]
[1, 1, -1, -1]
```

```
[1, -1, 1, 1]
[-1, 1, -1, -1]
[1, -1, -1, 1]
[-1, 1, 1, -1]
```

From the experiments in [8, 9, 13], the best algorithm is the "primal-dual variant with compaction" which is the one used here (algorithm number 13). We see that the symmetric sign vectors come together: indeed, the first and the second vector are opposite to each other, like the third and the fourth, the seventh and the eighth, the ninth and the tenth.

The primal-dual without compaction is called by replacing `13` with `5` in the second line. It provides the same sign vectors in `info.s`, but in a different order:

```
[-1, -1, -1, 1]
[-1, -1, -1, -1]
[1, -1, 1, 1]
[1, -1, -1, 1]
[-1, 1, 1, -1]
[-1, 1, -1, -1]
[1, 1, 1, 1]
[1, 1, 1, -1]
[1, 1, -1, 1]
[1, 1, -1, -1]
```

To consider a linear arrangement, one can input `V` (without a line of zeros for τ) and replace `false` to `true` when calling `options_from_algo`.

Now, we discuss some other variants of the tree algorithm.

4.3 Runs with the primal algorithm and its variants

Now, we compare the outputs of the RČ algorithm from [14], the initial primal algorithm described in section 3.1.

```
options = options_from_algo(0, false);
info = isf(Vt, options);
info.ns
info.s
info.cput_total # total time spent
```

for which the first should be `10` and the second should be the following set

```
[1, 1, 1, 1]
[1, 1, 1, -1]
[1, 1, -1, 1]
[1, 1, -1, -1]
[1, -1, -1, 1]
[1, -1, 1, 1]
[-1, 1, -1, -1]
[-1, 1, 1, -1]
[-1, -1, -1, 1]
[-1, -1, -1, -1]
```

Other interesting fields are the number of linear optimization problems (LOPs) solved during the run

```
info.nb_losolve   # 13
info.nb_feaslop   # 8
info.nb_infeaslop # 5
```

and the time spent on LOPs `info.cput_lp` (likely close to `info.cput_total`). The other fields correspond to values used in a dual algorithm hence irrelevant in this run.

The improvements A, B and C described in [8; section 4.2] can be obtained by

```
options = options_from_algo(3, false);      # standard
info = isf(Vt, options);
info.s
```

while the results with the compact variant are obtained by

```
options = options_from_algo(11, false);     # compact
info = isf(Vt, options);
info.s
```

Note that the orders in the sign vector sets `info.s` vary between algorithms.

4.4 Runs with the dual algorithm

Finally, we consider the dual variant from section 3.2, which uses only stem vectors. One can use the following instructions

```
Vt = [1 0 1 1 ; 0 1 1 -1 ; 0 0 1 0]
options = options_from_algo(7, false)
info = isf(Vt, options)
```

The results are a bit different, though `info.ns` is equal and `info.s` is the same up to a different order.

```
[-1, -1, -1, -1]
[-1, -1, -1, 1]
[1, -1, -1, 1]
[1, -1, 1, 1]
[-1, 1, -1, -1]
[-1, 1, 1, -1]
[1, 1, -1, -1]
[1, 1, -1, 1]
[1, 1, 1, -1]
[1, 1, 1, 1]
```

The number of LOPs are zero, which is expected. However, the values related to stem vectors are now nonzero. We see

```
info.nb_stems_sym   # 1
info.nb_stems_asym  # 3
info.nb_sv_checks   # 20
info.nb_sv_detects  # 5
```

So during the run, 20 covering tests were performed and 5 detected an infeasible descendant, by looking at 1 pair of symmetric stem vectors and 3 asymmetric ones. The symmetric stem vector (completed with zeros) is `[1; -1; 0; -1]` and the asymmetric ones are the columns of

```
[-1 -1  0
 -1  0 -1
  1  1  1
  0  1 -1]
```

The algorithm to compute all the circuits verifies subsets in this order: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$ (circuit yielding the first asymmetric stem vector), $\{1,2,4\}$ (circuit yielding the symmetric stem vector), $\{1,3\}$, $\{1,3,4\}$ (circuit yielding the second asymmetric stem vector), $\{1,4\}$, $\{2\}$, $\{2,3\}$, $\{2,3,4\}$ (circuit yielding the third asymmetric stem vector), $\{3\}$, $\{3,4\}$, $\{4\}$.

- $J = \{1,2,3\}$: $\mathcal{N}(V_{:,J}) = \mathbb{R}[-1;-1;+1]$, $\tau_J^\mathsf{T}\eta = [0\ 0\ 1][-1;-1;+1] = 1$, so asymmetric;
- $J = \{1,2,4\}$: $\mathcal{N}(V_{:,J}) = \mathbb{R}[+1;-1;-1]$, $\tau_J = 0$, so symmetric;
- $J = \{1,3,4\}$: $\mathcal{N}(V_{:,J}) = \mathbb{R}[-1;+1;+1]$, $\tau_J^\mathsf{T}\eta = [0\ 1\ 0][-1;+1;+1] = 1$, so asymmetric;
- $J = \{2,3,4\}$: $\mathcal{N}(V_{:,J}) = \mathbb{R}[-1;+1;-1]$, $\tau_J^\mathsf{T}\eta = [0\ 1\ 0][-1;+1;-1] = 1$, so asymmetric.

Concerning computation times, no time was spent on linear optimization. Instead, the values `info.cput_sv` (computing the stem vectors) and `info.cput_cover` (covering tests) are nonzero.

The dual algorithm with compaction may be called with

```
options = options_from_algo(15, false)
info = isf(Vt, options)
```

where the symmetric sign vectors come together: the first and the second vector are opposite to each other, like the third and the fourth, the seventh and the eighth, the ninth and the tenth.

```
[1, 1, 1, 1]
[-1, -1, -1, -1]
[1, 1, 1, -1]
[-1, -1, -1, 1]
[1, 1, -1, 1]
[1, 1, -1, -1]
[1, -1, 1, 1]
[-1, 1, -1, -1]
[1, -1, -1, 1]
[-1, 1, 1, -1]
```

Additional experiments are proposed in the folder `test`. In particular, the tables in [8] can be reproduced with some outputs of the file `isf_benchmark_launches_aff.jl` (though the experiments take some time).

5 The `isf.jl` code

5.1 Specifications

The code is split into several files, some of which contain a few functions (most often, one for linear arrangements – `H` for homogeneous, one for affine arrangements – `nH` for non-homogeneous – and one for the compact versions – `HnH`). The documentation of each function is available by typing "`?`" followed by the name of the function in a Julia REPL. Additional technical implementation details can be found in the function files.

Let `Vt` $:= [V; \tau^{\mathsf{T}}]$, a call to the `isf` code is made as follows:

```
info = isf(Vt, options)
```

where `options` is a set of parameters that details for instance which algorithm must be used or how to compute certain tests and `info` is a returned object containing various values relevant to the run. Both `options` and `info` are of the `mutable struct` type.

5.1.1 Input variables

We detail the `struct` named `Options` (types start with a capital in Julia). The main field of `Options` is `algorithm`, which defines how the code is going to proceed.

- `algorithm`: integer in $[0 : 15]$, which specifies the algorithm that the code must use. Table 5.1 describes the algorithms by their number `algorithm`. The main values of

`algorithm`	name [6]	meaning	stem vectors	compact?
0	RČ	primal	none (`sv = 0`)	no
1	A		none (`sv = 0`)	no
2	AB		none (`sv = 0`)	no
3	ABC	primal improved	none (`sv = 0`)	no
4	ABCD1		few (`sv = 1`)	no
5	ABCD2	primal-dual	some (`sv = 2`)	no
6	ABCD3		all (`sv = 3`)	no
7	AD4	dual	all (`sv = 3`)	no
8	RČ/C	primal compact	none (`sv = 0`)	yes
9	A/C		none (`sv = 0`)	yes
10	AB/C		none (`sv = 0`)	yes
11	ABC/C	improved compact	none (`sv = 0`)	yes
12	ABCD1/C		few (`sv = 1`)	yes
13	ABCD2/C	primal-dual compact	some (`sv = 2`)	yes
14	ABCD3/C		all (`sv = 3`)	yes
15	AD4/C	dual compact	all (`sv = 3`)	yes

Table 5.1: Correspondence between `options.algorithm` and the algorithm features.

`algorithm` are: `0` for the RČ (primal) algorithm [14] (section 3.1), `3` for the primal algorithm with the improvements in [8; section 4.2], `5` for the primal-dual algorithm (section 3.3), `7` for the dual algorithm (section 3.2). Add 8 to each to get the compact variants (thus 8, 11, 13 and 15).

- `bestv:` integer in $\{0, 1, 3\}$ that specifies the order in which the hyperplanes are considered. The value 0 means this option is unused, 1 means the next chosen index/hyperplane is the one maximizing the distance to the current witness point x^s (see section 3.1). When it equals 3, the chosen index also excludes the indices ensuring two descendants (this is modification C in [6,8]). It is used when `algorithm` $\in$ `{3,4,5,6,11,12,13,14}`.
- `dvnear0:` boolean. If `true`, when the current point is "close enough" to the new hyperplane, two descendants are obtained without major computation (this is modification B in [6,8]). It is used when `algorithm` $\in$ `{2,3,4,5,6,10,11,12,13,14}`.
- `optimizer: DataType`. Linear optimization solver, default to `HiGHS.Optimizer` (which is licence-free in Julia).
- `rational:` boolean. If `true` (default is `false`), computation is done in exact arithmetics using rational numbers, thanks to the LinearAlgebraX package. It is assumed that the user provides rational data for `Vt`.
- `s:` boolean. By default equal to `true`, but can be put to `false` to avoid storing the sign vectors (so they are just counted; see [3; third paragraph of the introduction]).
- `sv:` integer in $[0 : 4]$ standing for "stem vectors". When it equals 0, no duality is used (primal method of section 3.1). When it equals 1, only $p - r$ stem vectors are computed from the QR factorization (D1 in table 5.1, see [8; section 4.2.1. A]). When it equals 2, in addition to the $p - r$ starting ones, the code uses the primal-dual method of section 3.3 which collects some stem vectors during the run. When it equals 3, all stem vectors are computed (section 3.2). When it equals 4, the "crude dual algorithm" [8; algorithm 5.1], which does not use the $\mathcal{S}$-tree, is used. As a warning, we mention last this method is likely to be extremely slow unless p is small ($\leqslant 15$ for instance).
- `svsprod:` boolean. If `true`, the covering test is done recursively (`svsprod` stands for "stem vector sign product", see [13; section A.4.2]). By default, this value is equal to `false`.
- `symmetry:` boolean. If `true`, then only half the tree is computed since the arrangement is symmetric ($\tau = 0$, which is assumed to have been checked by the user).
- `tol_coordinates:` floating value expected to be very small. Tolerance under which the coordinates of a vector are assumed to be zero. Default value is 2.22×10^{-11}.
- `tol_nonzero_q:` floating value expected to be very small. Tolerance under which the absolute value of a quantity is assumed to be zero. Default value is 2.22×10^{-13}.
- `wechelon:` boolean standing for "with echelon(ned form)", which is an alternative way of computing (all) the stem vectors.
- `wechelon_svsprod_choice:` boolean. If `false` (default value), the code suggests the values of `wechelon` and `svsprod` to use.
- `withd:` boolean standing for "with directions". If `true` (default value), witness points are used (so the algorithm is not fully dual). It must be set to `false` to use a fully dual algorithm (where one must have `options.sv > 2`).

Some combinations of options and inputs may be incoherent, for instance if the matrix-vector pair (V, τ) is not rational but `options.rational == true`. Among them, the code detects some of such combinations (see section 5.2.1).

The file `isf_benchmark_test_aff.jl` compares the outputs of the various algorithms on the different instances used in [8]. The file `isf_benchmark_launches_aff.jl` contains numerical experiments including outputs reproducing the tables in [8].

algorithm	0/8	1/9	2/10	3/11	4/12	5/13	6/14	7/15
bestv	0	0	0	3	3	3	3	0
dvnear0	false	false	true	true	true	true	true	false
sv	0	0	0	0	1	2	3	3
withd	true	true	true	true	true	true	true	false

Table 5.2: Correspondence of algorithms and the most important options.

5.1.2 Output variables

The output of `isf` is a large `struct` which contains various elements about the run. In particular, some of the available benchmarking aggregates the more relevant output into a `DataFrame`, a structure which can be easily used to produce curves or other results.

All the fields related to stem vectors and covering tests are used only if `options.sv` is nonzero.

- `flag`: integer equal to 0 if the code worked as intended and > 0 if an error occured.
- `ns`: integer giving the number of sign vectors.
- `r`: integer giving the rank of V (sometimes unused).
- `s`: vector of vectors of integers containing the sign vectors, each `info.s[i]` is a sign vector.
- `sclst`: `Set` of vector of booleans (using $\{0, 1\}$ instead of $\{-1, +1\}$) used for $\mathcal{S}(V, \tau)^c$. It is only used when `sv = 4` and uses the `JuliaSet` structure to prevent duplications.
- `nb_losolve`: integer giving the number of linear problems solved.
- `nb_feaslop`: integer giving the number of feasible linear problems solved.
- `nb_infeaslop`: integer giving the number of infeasible linear problems solved.
- `nb_stems_sym`: integer giving (half) the number symmetric stem vectors ($|\mathfrak{S}_s(V, \tau)|/2$).
- `nb_stems_asym`: integer giving the number asymmetric stem vectors ($|\mathfrak{S}_a(V, \tau)|$).
- `nb_sv_checks`: integer giving the number of covering tests.
- `nb_sv_detect`: integer giving the number of covering tests that found a covered stem vector.
- `nb_duplicated_stems_sym`: integer giving the number of duplicated symmetric stem vectors found.
- `nb_duplicated_stems_asym`: integer giving the number of duplicated asymmetric stem vectors found.
- `svsprod_sym`: vector of integers giving the product between the matrix of symmetric stem vectors and the partial sign vector.
- `svsprod_asym`: vector of integers giving the product between the matrix of asymmetirc stem vectors and the partial sign vector.
- `cput_lp`: float giving the time spent on LOPs (including the minor fraction of preparing the data that intervene in the LOPs).
- `cput_sv`: float giving the time spent on the computation of the stem vectors.
- `cput_cover`: float giving the time spent on the covering tests.
- `cput_total`: float giving the total time.
- `stems_sym`: matrix of integers containing the symmetric stem vectors.
- `stems_asym`: matrix of integers containing the asymmetric stem vectors.

- `stems_sym_init`: used during the computation of the symmetric stem vectors.
- `stems_asym_init`: used during the computation of the asymmetric stem vectors.
- `stem_sizes_sym`: vector of integers giving the sizes (number of nonzero components, i.e., size of the associated circuits) of the symmetric stem vectors.
- `stem_sizes_asym`: vector of integers giving the sizes (number of nonzero components, i.e., size of the associated circuits) of the asymmetric stem vectors.
- `stem_zero_indices_sym`: vector of vectors of integers that preselects a subset of the symmetric stem vectors that may be covered by the sign vectors of size k.
- `stem_zero_indices_asym`: vector of vectors of integers that preselects a subset of the asymmetric stem vectors that may be covered by the sign vectors of size k.

5.2 Parts of the code

Many parts of the code are split into three (rather) similar parts: one for the case of linear arrangements, i.e., the symmetric case with $\tau = 0$, named "H" (for Homogeneous), one for affine arrangements named "nH" (for nonHomogeneous), one for affine arrangements dealt with by compact algorithms named "HnH" (for Homogeneous-nonHomogeneous). It is assumed that if $\tau = 0$, the user knows it and sets `symmetry` as `true`. In this case, only half the sign vectors are computed (each sign vector at the end can be multiplied by -1 to obtain another feasible sign vector).

When precising which of the three versions is not important, we shall use `*` to generalize to all versions. If the location of a function is not specified, it is in the file of the same name. Let us mention a convention in Julia, which states that functions with a `!` at the end of their names modify their parameters instead of returning objects (essentially, modifying fields of `info`).

5.2.1 General functions

The file `isf_tools.jl` defines the two structures in `isf`: `options` and `info`, initialized by `isf_get_options` and `isf_get_info`. The call `options_from_algo(algorithm, symmetry)` creates the appropriate options from a value of `algorithm` and `symmetry`. The three files `isf_tools.jl`, `isf_binary_tree_update.jl` and `isf_bin_operations.jl` also contain various elementary functions that are useful in the code. The file `isf_bounds.jl` contains the two upper bounds on $|\mathcal{S}(V,\tau)|$ and $|\mathcal{S}(V,0)|$ [16].

The hat function `isf` is located in `IncSignFeas.jl`: it first verifies the conformity of the `options` with `isf_check` and that the specified options are not contradictory (for instance, if the arrangement is indicated as symmetric but the algorithm is compact) with `isf_coherence`. Finally, it verifies if there are some identical hyperplanes with `isf_noncolin.jl`; these can be removed from the recursion (the sign vectors are modified once the $\mathcal{S}$-tree has been fully explored). After this, it launches the main algorithms.

5.2.2 Stem vectors

The stem vectors represent an important part of `IncSignFeas.jl`. In the variant that uses a sprinkle of duality (D1 or `options.sv = 1`), the `isf_somesv_*` functions compute a few stem vectors. For the primal-dual variant (D2 or `options.sv = 2`), it is done by the function `isf_sv_add`: from the dual variables, one only needs to extract the nonzero components, and verify if the stem vector is symmetric or asymmetric.

For the algorithms computing all the circuits, there are two groups of functions/files. The classic computation of circuits is done by the functions `isf_allsv_*`, which verify the potential subsets of interest, and the functions `isf_allsv_from_indices_*`, which test the subsets. The computation with the echelon form suggested in [15; section 9] is realized by the functions `isf_allsv_*_Wechelon`, which prepare a call to the recursive functions `isf_allsv_*_Wechelon_rec` and then post-process the obtained stem vectors. From the experiments, this technique is better if the stem vectors have many nonzero coordinates (general position, highly random instances, ...). They use the file `isf_echelonning.jl` (a slight adaptation from the related Julia package).

Let us recall that the symmetric stem vectors come by opposite pairs. Up to a slight modification of the test in definition 3.1, one only needs to keep one of each pair in memory; this reduces the complexity of the covering tests.

The covering tests are performed by the `isf_cover_stem_*` functions, which essentially return the boolean (`'` is the transpose in Julia)

```
b = 0 in (info.stems_asym)' * [s] - info.stems_sizes_asym
```

where `[s]` denotes the current sign vector `s` completed by zeros into a vector of size p and is tested against the asymmetric stem vectors. For the symmetric ones, one may use

```
b = 0 in (abs.(info.stems_sym' * [s]) - info.stem_sizes_sym)
```

where the absolute value is here to deal with the fact we only take one symmetric stem vector in each opposite pair. Other computations of that boolean were tested but this one seemed slightly faster. The recursive computation of the covering test consists in observing that (see [13; section A.4.2]), between the covering test of s and say $(s,+)$, one has (denoting `i[k+1]` the index of the new hyperplane):

```
info.stems_sym' * [(s, +)] = info.stems_sym' * [s] + info.stems_sym[:,i[k+1]].
```

Said briefly, instead of doing a matrix-vector product every time, we update a vector. However, the length of this vector to keep in memory (and multiple of them even with a depth-first search) is the number of stem vectors, which may be large. Unsurprisingly, it appeared to be relevant for instances with many stem vectors, especially when they have many nonzero coordinates (general position, highly random instances, ...).

In general, it may not be very clear whether these two options should be used or not. The file `isf_choose_hard_options.jl` proposes a test for each option suggesting a value to choose. The test is designed from what performed best for the instances in [6].

5.2.3 Recursive process

The hat function `isf` calls one of the starting functions, `isf_first_*` (or `isf_first_rc` for the original algorithm of [14]). The recursive processes, `isf_rec_*`, construct the $\mathcal{S}$-tree. There is a slight difference for the fully dual algorithms (`options.withd` is `false`): if `options.sv = 3`, the file `isf_rec_nod.jl` ("no d(irections)") intervenes, which still employs the $\mathcal{S}$-tree. When `options.sv = 4`, the algorithm forks to `isf_nod_lazy.jl` and `isf_nod_lazy_completion.jl`, which generates all infeasible sign vectors from each stem vectors (with *many* duplicates), then computes $\{\pm 1\} \setminus \mathcal{S}(V,\tau)^c$. In particular, up to a simple modification of the code, one could compute $\mathcal{S}(V,\tau)^c$. Let us mention this algorithm is in general quite slow and unlikely to be effective (we suggest limiting it to $p \leqslant 15$).

The primal algorithms essentially verifies the feasibility of linear systems via `isf_feas.jl`, which employs the optimization solver HiGHS to solve (3.2). The experiments in [8,9,13] were done with Gurobi, however GLPK was also tested. Rather minor differences were observed. The heuristics B and C, i.e., when `options.dvnear0` is `true` and `options.bestv` is > 0, are implemented in the file `isf_nytpe.jl`.

5.2.4 End of the recursion

Once the current sign vector has size p, the recursion stops. If `options.s` if `true`, the sign vector is stored (`isf_storing_*`).

5.2.5 Numerical experiments

The files for numerical experiments are located in the `test` folder. The instances tested were generated by the functions from `isf_generation_linear.jl` and `isf_generation_affine.jl`.

The files `isf_benchmark_test_lin.jl` and `isf_benchmark_test_aff.jl` were created to verify the multiple variants (16 for linear instances and 32 for affine instances, not testing the rational computation), to ensure they returned the same sign vector sets. Up to numerical precision, no error was found.[4]

The recursive covering was tested in `isf_benchmark_recursive_covering.jl`. The two most important files, `isf_benchmark_launches_lin.jl` and `isf_benchmark_launches_aff.jl` contain the benchmarking of the linear [6] and affine [8] instances respectively. They utilize `isf_benchmark_values.jl`.

Some experiments about data structures can be found in the folder `TESTS_method`. A brute force method, testing all the 2^p systems, may be found in the folder `brute_force`. Additional details on some code results can be found in [13; Appendix A].

6 References

[1] Christos A. Athanasiadis (1996, September). Characteristic Polynomials of Subspace Arrangements and Finite Fields. *Advances in Mathematics*, 122(2), 193–233. 4

[2] J. Frédéric Bonnans, Jean Charles Gilbert, Claude Lemaréchal, Claudia A. Sagastizábal (2006). *Numerical Optimization: Theoretical and Practical Aspects*. Universitext. Springer-Verlag, Berlin, second edition. 8

[3] Taylor Brysiewicz, Holger Eble, Lukas Kühne (2023, December). Computing Characteristic Polynomials of Hyperplane Arrangements with Symmetries. *Discrete & Computational Geometry*, 70(4), 1356–1377. 15

[4] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2023). ISF and BDIFFMIN. 3

[5] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2023). ISF and BDIFFMIN - MATLAB functions for central hyperplane arrangements and the computation of the B-differential of the componentwise minimum of two affine vector functions. Technical report, Inria Paris, Université de Sherbrooke. 3

[6] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2025). On the B-differential of the componentwise minimum of two affine vector functions. *Mathematical Programming Computation*. 3, 5, 14, 15, 18, 19

[4] In (3.2), if the optimal α is very close to zero, depending on the algorithm, sometimes a chamber may be detected and sometimes not. This occurred, though rarely, in the experiments.

[7] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2025). On the B-differential of the componentwise minimum of two affine vector functions - The full report. Technical report, Inria Paris, Université de Sherbrooke. 3

[8] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2025). Primal and dual approaches for the chamber enumeration of real hyperplane arrangements. *(submitted)*. 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 19

[9] Jean-Pierre Dussault, Jean Charles Gilbert, Baptiste Plaquevent-Jourdain (2025). Primal and dual approaches for the chamber enumeration of real hyperplane arrangements - The full report. Technical report (in preparation), Inria Paris, Université de Sherbrooke. 3, 5, 6, 8, 9, 11, 19

[10] Theodore S. Motzkin (1936). Beiträge zur Theorie der linearen Ungleichungen. Technical report, University Basel, Jerusalem, Israel. 5, 7

[11] Peter Orlik, Hiroaki Terao (1992). *Arrangement of Hyperplanes*, volume 300 of *Grundlehren Der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag Berlin Heidelberg GmbH, Madison. 4

[12] James G. Oxley (2011). *Matroid Theory*. Oxford Graduate Texts in Mathematics 21. Oxford University Press, Oxford New York, NY, second edition edition. 6

[13] Baptiste Plaquevent-Jourdain (2025). *A Robust Linearization Method for Complementarity Problem – A Detour Through Hyperplane Arrangements*. PhD Thesis, Sorbonne Université, Université de Sherbrooke. 9, 11, 15, 18, 19

[14] Miroslav Rada, Michal Černý (2018, January). 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. 3, 7, 9, 11, 14, 18

[15] Jörg Rambau (2023). Symmetric lexicographic subset reverse search for the enumeration of circuits, cocircuits, and triangulations up to symmetry. pages 1–41. 8, 18

[16] Ludwig Schläfli (1950). *Gesammelte mathematische Abhandlugen*. Birkhäuser, Springer, Basel. 17

[17] Robert Owen Winder (1966, July). Partitions of N -Space by Hyperplanes. *SIAM Journal on Applied Mathematics*, 14(4), 811–818. 5

[18] Thomas Zaslavsky (1975). Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes. *Memoirs of the American Mathematical Society*, 1(154), 1–109 (?). 5

[19] Günter M. Ziegler (2007). *Lectures on Polytopes*, volume 152 of *Graduate Texts in Mathematics*. Springer New York, New York, NY, 7th edition. 5