ISF.jl: Julia functions for the identification of the chambers of hyperplane arrangements

Jean-Pierre DUSSAULT [†], Jean Charles GILBERT [‡] and Baptiste PLAQUEVENT-JOURDAIN [§]

(Thursday 7th August, 2025, 10:22)

The Julia function `isf` aims at computing the chambers of a hyperplane arrangement. By computing, we mean identifying the *sign vectors*, i.e., vectors with coordinates equal to -1 or $+1$, corresponding to each of the (nonempty) chambers. Several versions of the algorithms are available, as well as parameters to adapt depending on the user's motivations. In a related paper by the authors, it was observed that when hyperplanes have a point in common, i.e., have a nonempty intersection, this problem has many equivalent reformulations that can thus be also treated after a manipulation of the output of `isf`.

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` ref ?(du package/code en ligne), which stands for "incremental signed feasibility". It is aimed at computing the chambers of hyperplane arrangements, by using algorithms based on the tree structure developed for that purpose in an article of Rada and Černý [10]. The code described here is highly related to [7] (and [8]) that uses the results produced by `isf`. It follows [5,6] and their related Matlab code [3], described in [4], who focus on *centered* hyperplane arrangements, which are equivalent to various other problems, thus also solvable by the Julia code described here after a manipulation of its main output.

Notation

One denotes by $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{R}$ the sets of integers, 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 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 $u \cdot v \in \mathbb{R}^n$, whose ith component is $u_i v_i$. The sign function $\operatorname{sgn} : \mathbb{R} \to \mathbb{R}$ is defined by $\operatorname{sgn}(t) = +1$ if $t > 0$, $\operatorname{sgn}(t) = -1$ if $t < 0$ and $\operatorname{sgn}(0) = 0$. The sign of a vector x or a matrix M is defined componentwise: $\operatorname{sgn}(x)_i = \operatorname{sgn}(x_i)$ and $[\operatorname{sgn}(M)]_{i,j} = \operatorname{sgn}(M_{i,j})$ for i and j. For $u \in \mathbb{R}^n$, $|u| \in \mathbb{R}^n$ is the vector defined by $|u|_i = |u_i|$ for all $i \in [1:n]$. 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 $\operatorname{rank}(A) := \dim \mathcal{R}(A)$ and its nullity by $\operatorname{null}(A) := \dim \mathcal{N}(A) = n - \operatorname{rank}(A)$ 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: $A_{i,:}^{\mathsf{T}}$ 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 $u \in \mathbb{R}^n$, $\operatorname{Diag}(u) \in \mathbb{R}^{n \times n}$ is the square diagonal matrix with $\operatorname{Diag}(u)_{i,i} = u_i$. The orthogonal of a subspace $Z \subseteq \mathbb{R}^n$ is denoted by $Z^\perp := \{x \in \mathbb{R}^n : x^{\mathsf{T}} z = 0, \text{ for all } z \in Z\}$.

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

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

If $v = 0$, then one has the following identities (recall that $\operatorname{sgn}(0) = 0$):

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

which can be summarized into $H^{-\operatorname{sgn}(\tau)} = \mathbb{R}^n$ and the other two sets are empty. In what follows, we assume all the hyperplanes considered are *proper*, in the sense that $v \neq 0$; improper hyperplanes do not split the space in three.

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 $v_1, \ldots, v_p \in \mathbb{R}^n$ 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* or *central* 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 $\tau \in \mathcal{R}(V^\mathsf{T})$ ([7; section 3]).

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.

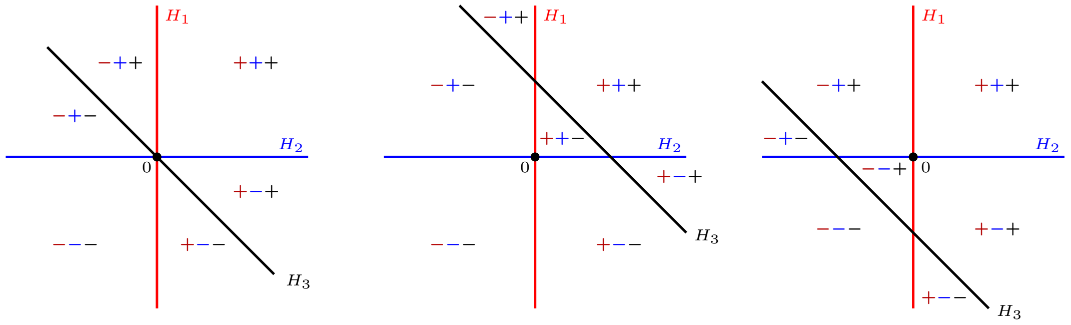

Figure 2.1: Arrangements in $\mathbb{R}^2$ specified by the hyperplanes $H_1 := \{x \in \mathbb{R}^2 : x_1 = 0\}$, $H_2 := \{x \in \mathbb{R}^2 : x_2 = 0\}$, $H_3(\text{left}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = 0\}$, $H_3(\text{middle}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = 1\}$ and $H_3(\text{right}) := \{x \in \mathbb{R}^2 : x_1 + x_2 = -1\}$. 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.

When considering all hyperplanes, the intersections of half-spaces are labelled by one sign per hyperplane. Since we only consider the open subsets, i.e., the intersections of the open half-spaces, the signs take value -1 or $+1$. We now consider the following problem.

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*

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

In what follows, unless specified we always assume the setting of problem 2.1 holds, i.e., n and p in $\mathbb{N}^*$, $V \in \mathbb{R}^{n\times p}$ with nonzero columns and $\tau \in \mathbb{R}^p$. We now give a few of the main properties observed on such sign vector sets $\mathcal{S}$. They are mostly taken from [7].

Proposition 2.2 (some properties of $\mathcal{S}$) *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, in the sense that* $-\mathcal{S}(V,0) = \mathcal{S}(V,0)$.
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 bound is attained when the hyperplanes are in general position.*
7) *$|\mathcal{S}(V,\tau)| = (-1)^r \chi(-1) = \sum_{J\subseteq[1:p],\tau_J\in\mathcal{R}(V_{:,J}^{\mathsf{T}})}(-1) = (|\mathcal{S}(V,0)| + |\mathcal{S}([V;\tau^{\mathsf{T}}],0)|)/2$ where the first formula is from Zaslavsky [13], and the two others are rewritings of formulas from Winder [12].*

General position states as follows ([8; proposition 3.33]), where the first characterization is geometric and the second is algebraic. There exists a slightly different variant for when $\tau = 0$.

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

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

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

2.2 Stem vectors

It was observed in the aforementioned papers that the determination of $\mathcal{S}$ can be tackled "dually", by using the stem vectors (also called signed circuits for instance in [14]). Let us introduce the circuits, the collection of subsets of $[1:p]$ such that

$$\mathcal{C} := \{J \subseteq [1:p] : J \neq \varnothing, \operatorname{null}(V_{:,J}) = 1, V_{:,J_0} \text{ is injective for all } J_0 \subsetneq J\}. \tag{2.2}$$

The set $\mathcal{C}$ is the set of *circuits* of the *vector matroid* formed by the columns of V and its subsets of linearly independent columns [9].

Definition 2.3 (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 $\mathfrak{S}_s$ and the one of asymmetric stem vectors by $\mathfrak{S}_a$.

In particular, if σ is a symmetric sign vector, then $-\sigma$ also is one. We mention a few properties, mostly taken from [7].

Proposition 2.4 (some properties of $\mathfrak{S}$) *The following properties hold.*

1) *The circuits are independent from τ but not the stem vectors.*
2) *Since the nullity of $V_{:,J}$ is 1, its null space is of the form $\mathbb{R}\eta$ for some $\eta \in \mathbb{R}^J$. By the second part of the definition, $\eta_j \neq 0$ for all $j \in J$ (otherwise, $J_0 := J\backslash\{j\}$ would contradict the definition).*
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 in general position.*
6) *One has $\mathfrak{S}(V,-\tau) = -\mathfrak{S}(V,\tau)$, $\mathfrak{S}_a(V,-\tau) = -\mathfrak{S}_a(V,\tau)$ and $\mathfrak{S}_s(V,-\tau) = \mathfrak{S}_s(V,\tau)$.*
7) *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 $\{-1,+1\}$.*
8) *A circuit of size 1 would mean some $v_i = 0$. Thus, circuits are of size $\geqslant 2$; circuits of size exactly 2 correspond to parallel (or identical) hyperplanes.*

The main property of stem vectors, in the context of identifying $\mathcal{S}$, is the following:

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

In what follows, we say that a sign vector s *covers* a stem vector σ in this case.

2.3 Comments on arrangement types

This section introduced linear and affine arrangements, though these two types are related in various ways. Indeed, one can swap between affine and linear by the following: let $t \in \mathbb{R}$ and

$$\mathcal{V}_t := \begin{bmatrix} V & 0 \\ \tau^\mathsf{T} & t \end{bmatrix}, \quad \mathcal{S}(\mathcal{V}_t, 0) = [\mathcal{S}(V,\tau) \times \{-\operatorname{sgn}(t)\}] \cup [\mathcal{S}(V,\tau) \times \{-\operatorname{sgn}(t)\}].$$

The matrix $\mathcal{V}_t$ and the linear arrangement $\mathcal{A}(\mathcal{V}_t, 0)$ have one more dimension and one more hyperplanes. It is clear that it suffices to compute half of a linear (or centered) arrangement, since the opposite half can be obtained from it by a simple multiplication; this amounts at looking at only what happens in the halfspace of one of the hyperplanes (say, the first one).

Thus, the above relation tells us that affine arrangements are "half" of linear ones – which is a reformulation of saying one only needs to compute half of a linear arrangement. To summarize,

$$\begin{aligned} \mathcal{S}(V,\tau) &:= \text{affine } (n,p) \simeq \text{half of linear } (n+1,p+1), \\ \mathcal{S}(V,0) &:= \text{linear } (n,p) \simeq \text{two opposite affine } (n-1,p-1), \end{aligned}$$

which confirms that in general, an affine arrangement is a little bit more complicated and difficult to compute than a linear one.

In [7,8], it was observed that the matrix $[V;\tau^\mathsf{T}]$ could be used to compute, for an affine arrangement, the asymmetric part of $\mathcal{S}(V,\tau)$ and only half of the symmetric part of $\mathcal{S}(V,\tau)$.

3 The `isf.jl` code

The Julia function `isf` solves problem 2.1. It constructs a tree where level k of the tree corresponds to feasible sign vectors of size k. Thus, its leaves are the sign vectors of $\mathcal{S}(V,\tau)$. This tree construction lead to the name "Incremental Sign Feasibility" close to "Inc(remental) Enu(meration)" in [10].

3.1 Specifications

The main function, `isf`, as well as most others, has a relatively detailed documentation available by typing "`?`" then the name of the function in a Julia REPL. Furthermore, many details are available in the function files (often of the form `function_name.jl`), and in the code of the function themselves, to precise some technical elements. Let `Vt` $:= [V;\tau^\mathsf{T}]$, it is used by calling

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

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

3.1.1 Input variables

We detail the `struct` named `options`. Some fields intersect, in the sense that some values are supposed to be coherent, see the comments after the descriptions. In the code and files, HnH which stands for Homogeneous - nonHomogeneous, denotes the method computing only half of the symmetric part of $\mathcal{S}$.

- `algorithm:` integer in $[0:15]$, which corresponds to the possible algorithms the code is able to use. The algorithm names come from [5]. Table 3.2 details which algorithm corresponds to which options.

`algorithm =`	0	1	2	3	4	5	6	7
algorithm	`RC`	A	AB	ABC	ABCD1	ABCD2	ABCD3	AD4
`algorithm =`	8	9	10	11	12	13	14	15
compact version	`RC`	A	AB	ABC	ABCD1	ABCD2	ABCD3	AD4

Table 3.1: Correspondence between `options.algorithm` and the algorithm. The difference between 0/8 and 1/9 comes from a structure in the tree (not a parameter value).

- `bestv:` integer in $\{0, 1, 3\}$ (2 is an old unused value). The value 0 means this option is unused, 1 means the next index/hyperplane chosen is the one maximizing the distance of the current point to the hyperplane, while 3 means choosing one that has a high chance of producing only one descendant (this is modification C in [5, 7]).
- `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 [5, 7]).
- `rational:` boolean. If `true`, computation is exact since made in rational numbers, with the LinearAlgebraX package. It is assumed that the user provided rational data `Vt`.
- `s:` boolean. By default, `true`, but can be put to `false` to avoid storing the sign vectors (so they are just counted; in this case, see [2; third paragraph of the introduction]).
- `sv:` integer in $[0:4]$, corresponding to the use of stem vectors. The value 0 means no stem vectors, 1 means only $p-r$ stem vectors computed from the QR factorization, 2 means infeasible LOPs add a stem vector to the list, in addition to the $p-r$ starting ones. The value 3 means all stem vectors are computed, while 4 means *no* tree algorithm is used, and instead one uses the "crude dual algorithm" in [7]; this value technically overrides the value of `algorithm` (which is implemented but not seriously considered).
- `svsprod:` boolean that must be manually specified (`options.svsprod = true`); if `true`, means the covering test is done recursively (`svsprod` stands for stem vector sign product).
- `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}.

`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 3.2: Correspondance of algorithms and main options.

- `wechelon`: boolean standing for "with echelon(ned form)", which is an alternate way of computing (all) the stem vectors.
- `withd`: boolean standing for "with directions", meaning that witness points are used (so the algorithm is not fully dual) when `true`.

These fields assume the user has not provided incoherent inputs, such as a matrix-vector pair (V, τ) with non-rational data while `options.rational == true`.

The file `isf_benchmark_launches_aff.jl` possess all main combinations of options that were tested. The other options, `rational`, `svsprod`, `wechelon`, can be added.

To conclude, let us mention that the best algorithm seems to be, overall, algorithm ABCD2, i.e., `algorithm = 5 (or 13)` (and with `svsprod = false`).

3.1.2 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 relevant information into a `DataFrame`, which can be easily manipulated to output 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 everything went well and > 0 if an error occured.
- `ns`: integer being the number of sign vectors.
- `r`: rank of V (sometimes unused).
- `s`: vector of vector of integers containing the sign vectors.
- `sclst`: `Set` of vector of booleans (using $\{0, 1\}$ instead of $\{-1, +1\}$); this structure is only used when `sv = 4` since the crude dual algorithm computes the infeasible set and the JuliaSet structure prevent duplication.
- `nb_losolve`: integer representing the number of linear problems solved.
- `nb_feaslop`: integer representing the number of feasible linear problems solved.
- `nb_infeaslop`: integer representing the number of infeasible linear problems solved.
- `nb_stems_sym`: integer representing (half) the number symmetric stem vectors ($|\mathfrak{S}_s|/2$).
- `nb_stems_asym`: integer representing the number asymmetric stem vectors.
- `nb_sv_checks`: integer representing the number of covering tests.
- `nb_sv_detect`: integer representing the number of successful covering tests.
- `nb_duplicated_stems_sym`: integer representing the number of duplicated symmetric stem vectors found.
- `nb_duplicated_stems_asym`: integer representing the number of duplicated asymmetric stem vectors found.

- `svsprod_sym`: vector of integers representing the stem vectors – sign product for the symmetric stem vectors.
- `svsprod_asym`: vector of integers representing the stem vectors – sign product for the asymmetric stem vectors.
- `cput_lp`: float representing the time spent on LOPs (including the minor fraction of preparing the data that intervene in the LOPs).
- `cput_sv`: float representing the time spent on the computation of the stem vectors.
- `cput_cover`: float representing the time spent on the covering tests.
- `cput_total`: float representing 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`: matrix of integers for the precomputation of the symmetric stem vectors.
- `stems_asym_init`: matrix of integers for the precomputation of the asymmetric stem vectors.
- `stem_sizes_sym`: vector of integers representing the sizes of the symmetric stem vectors.
- `stem_sizes_asym`: vector of integers representing the sizes of the asymmetric stem vectors.
- `stem_zero_indices_sym`: vector of vector of integers a certain ordering of the symmetric stem vectors so preselect those that may be relevant.
- `stem_zero_indices_asym`: vector of vector of integers a certain ordering of the asymmetric stem vectors so preselect those that may be relevant.

3.2 Parts of the code

Many parts of the code are split into three (rather) similar parts: one for the case of central 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 inputs `symmetry` as `true`. Furthermore, only half the sign vectors were 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 write use `*` to generalize to all three. If the location of a function is not precised, 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`).

3.2.1 General functions

The file `isf_tools.jl` defines the two main structures used: `options` and `info` (the `values` object correspond to the success or reason of failure), which are initialized by `isf_get_...`. A few elementary tools are contained in it and in `isf_bounds.jl`, `isf_binary_tree_update.jl` and `isf_bin_operations.jl`. Recall that for a given algorithm (see table 3.1), the appropriate values of `options` are initialized by function `options_from_algo`.

The main function `isf` is located in `isf_main.jl`: it first verifies the conformity of the `options` with function `isf_check` and that options do not contradict with `isf_coherence` (not exhaustively), for instance if the arrangement is indicated as symmetric but the algorithm is compact. Finally, it verifies if there are some identical hyperplanes with `isf_noncolin.jl`, which can be removed from the recursion (the sign vectors are corrected once the tree is finished). After this, it launches the main algorithms.

3.2.2 Stem vectors

The stem vectors represent an important part of `ISF.jl`. In the variant that uses a sprinkle of duality (D1 or `sv = 1`), the `isf_somesv_*` functions compute a few stem vectors. For the primal-dual variant (D2 or `sv = 2`), it is done by function `isf_sv_add`: from the dual variables, one only need 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 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 [11; section 9] is realized by the functions `isf_allsv_*_Wechelon`, preparing a call to recursive functions `isf_allsv_*_Wechelon_rec` and post-treat 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 mention that the symmetric stem vectors are "paired" and only one of the pair is used, to avoid storing two elements having the same role. In short, the algorithm to compute the circuits (and stem vectors) verify subsets and stops the exploration of a subtree when a circuit is detected.

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 and `'` denotes the transposition in Julia. For the symmetric ones, one may use

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

Other computations of that boolean were tested but this one seemed slightly faster. The recursive computation of the covering test consists in observing that, 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, essentially based on the observations for the instances tested in [5] on which version is better for which instance.

3.2.3 Recursive process

The main function `isf` calls one of the starting functions, `isf_first_*` (or `isf_first_rc` for the original algorithm of [10]). 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 $\mathcal{S}^c$ (it is unlikely to be fast).

The primal algorithms essentially verifies the feasibility of linear systems via `isf_feas.jl`, which employs the optimization solver Gurobi. GLPK was also tested, but with rather minor differences observed. The file `isf_nytpe.jl` serves for heuristics B and C, i.e., when `options.dvnear0` is `true` and `options.bestv` is > 0.

3.2.4 End of the recursion

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

3.2.5 Numerical experiments

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 (which occurred for 1-2 chambers in a few of the instances), no error was found.

The recursive covering was tested in `isf_benchmark_recursive_covering.jl`. The two main files, `isf_benchmark_launches_lin.jl` and `isf_benchmark_launches_aff.jl` contain the benchmarking of the linear ([5]) and affine ([7]) instances respectively. They employ `isf_benchmark_values.jl`.

The file `archive_pb_lin_bugs_proper.jl` contains the details about issues of "bad conditioning", where depending on whether the data is normed or not, a chamber is deemed nonempty or not. In general, this should not be possible, but it may happen when some hyperplanes are defined by large coordinates and some are not.

4 Example

Consider the small arrangement in dimension 2 given by:

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

The code is loaded with `include("isf_main.jl")` – note that a Gurobi license [1] is required to use linear optimization. After defining V and τ, the options for the RC algorithm can for instance be used with `options = options_from_algo(0, false)`. The code is launched with `info = isf(Vt, options_test)`.

[1]Free in academia and reasonably easy to obtain.

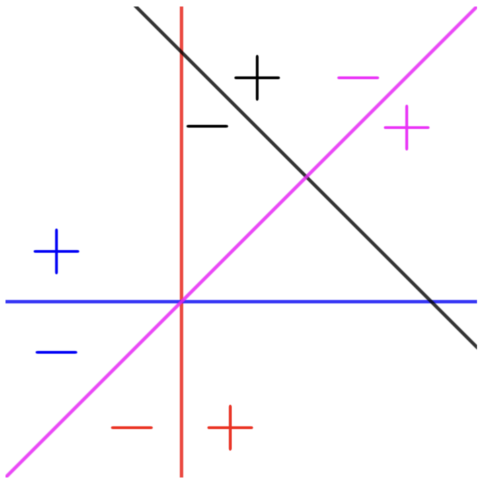

Figure 4.1: The four hyperplanes in $\mathbb{R}^2$. The signs represent the halfspaces.

The return starts as follows: `Info(0, 10, 0, 0, ...)`: these four values indicate that the code finished well, that there are 10 sign vectors – the complementary set (`SetVectorBool()`) and the rank are not computed. The sign vectors are (writing `+1` instead of `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,-1]]
```

The following values `13, 8, 5, 0, 0, 0, 0, 0, 0, Int64[], Int64[]` correspond to the number or LOPs – 8 feasible (the code starts with + and −, so 8 feasible ones to add 8 feasible sign vectors) and 5 infeasible. The six zeros correspond to informations on the stem vectors (unused here).

The four floating values `0.014656001, 0.0, 0.0, 0.016024292` correspond to the time spent on the major parts of the code – tow zeros for times related to stem vectors, and the major part of the time related to solving LOPs. The eight remaining quantities are also related to stem vectors, and in a purely primal algorithm, read

```
Matrix{Int64}(undef, 0, 0), Matrix{Int64}(undef, 0, 0),
0×0 ElasticMatrix{Int64, Vector{Int64}},
0×0 ElasticMatrix{Int64, Vector{Int64}},
Int64[], Int64[], Vector{Int64}[], Vector{Int64}[]
```

If one uses `options = options_from_algo(7, false)`, i.e., the fully dual method, the results are a bit different. The rank, equal to 2, is computed `Info(0, 10, 0, 2, ...)`. The sign vectors are returned in a different order but are the same (similarly, the complementary set is not computed `SetVectorBool()`).

```
[[-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 following values `0, 0, 0, 1, 3, 20, 8, 0, 0` confirm that LO was not used, that there is 1 symmetric stem vector and 3 asymmetric ones, 20 covering tests and 8 detecting infeasible descendants, and no duplicates in the computation, since the arrangement is very small (the `Int64[], Int64[]` are nonempty only with recursive covering). Indeed, it is rather clear that there are no circuits of size 2 since no hyperplanes are parallel, and none of size 4 since the dimension is 2. Now, the algorithm to compute circuits will verify subsets in this order: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$ (circuit detected, so no further exploration), $\{1,2,4\}$ (circuit detected and no more indices to be considered), $\{1,3\}$, $\{1,3,4\}$ (circuit detected and no more indices to be considered), $\{1,4\}$, $\{2\}$, $\{2,3\}$, $\{2,3,4\}$ (circuit detected and no more indices to be considered), $\{3\}$, $\{3,4\}$, $\{4\}$.

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

Indeed, the matrices of stem vectors (completed with zeros) are `[1; -1; 0; -1;;]` and

```
[-1 -1  0;
 -1  0 -1;
 +1 +1 +1;
  0 +1 -1],
```

The computing times, `0.0, 0.231294208, 0.043574586, 0.277206917`, indicate no time spent on LO, and a major part of the time for the computation of stem vectors, a minor part for the covering tests (on larger instances, it is not always the case). The last fields are additional information on the stem vectors.

5 References

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

[2] Taylor Brysiewicz, Holger Eble, Lukas Kühne (2023, December). Computing Characteristic Polynomials of Hyperplane Arrangements with Symmetries. *Discrete & Computational Geometry*, 70(4), 1356–1377. 7

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

[4] 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

[5] 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, 7, 10, 11

[6] 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

[7] 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, 11

[8] 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

[9] James G. Oxley (2011). *Matroid Theory*. Oxford Graduate Texts in Mathematics 21. Oxford University Press, Oxford New York, NY, second edition edition. 5

[10] 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, 6, 11

[11] Jörg Rambau (2023). Symmetric lexicographic subset reverse search for the enumeration of circuits, cocircuits, and triangulations up to symmetry. pages 1–41. 10

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

[13] 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

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