Journal of Software for Algebra and Geometry

The OrientedMatroids package for SageMath

Aram Dermenjian, Elizabeth Flight and Tudor Tanasa

vol 15 2025

JSAG **15** (2025), 81–91
https://doi.org/10.2140/jsag.2025.15.81

The Journal of Software for
Algebra and Geometry

The `OrientedMatroids` package for SageMath

ARAM DERMENJIAN, ELIZABETH FLIGHT AND TUDOR TANASA

ABSTRACT: We introduce the `OrientedMatroids` package for SageMath. It provides an interface for constructing oriented matroids using one of various cryptomorphic definitions. It also provides methods for transferring between the various definitions and implements some basic properties for oriented matroids.

1. INTRODUCTION. Matroids are a generalisation of linear independence and can be defined using one of many cryptomorphic definitions, such as using graphs, rank functions, closure operators, etc., where two definitions are said to be cryptomorphic if they are equivalent, but not obviously equivalent. The mathematical software system SageMath [5] already contains classes for constructing matroids using any one of many cryptomorphic definitions. Oriented matroids are a generalisation of matroids obtained by adding some orientation. They are combinatorial generalisations of directed graphs, (real) hyperplane arrangements, (real) point configurations and more. Although SageMath contains methods for constructing matroids, no such constructions exist for oriented matroids in SageMath. With recent developments in matroid theory, it became worthwhile to create the `OrientedMatroids` package in order to allow experimentation and working with oriented matroids from a programming point of view.

The `OrientedMatroids` package allows for constructing an oriented matroid using one of its cryptomorphic definitions with a system in place to allow the easy addition of more definitions in the future. In addition, the package allows the user to choose which cryptomorphic definition is being used, allowing the user to change between different points of view when working with oriented matroids. Finally, the package allows the output of basic properties and structures that a particular oriented matroid might possess, contain or be related to.

Outline. In Section 2 we give the necessary background for oriented matroids. In Section 3 we discuss the structure of the package itself and the methods that currently exist. Finally, we conclude in Section 4 by giving some examples of using the `OrientedMatroids` package to work with oriented matroids. The package can be found on Github and will eventually be integrated into SageMath.

Dermenjian author was sponsored by a research grant from the Heilbronn Institute for Mathematical Research. Flight and Tanasa were sponsored by summer internship grant from the University of Manchester.
MSC2020: 52C40.
Keywords: oriented matroids, SageMath, Python.

Acknowledgements. We would like to thank the anonymous reviewer for many helpful comments and suggestions for improvements. We would also like to thank Travis Scrimshaw and Bryan Gillespie for helpful conversations on implementing oriented matroids into SageMath.

2. BACKGROUND FOR ORIENTED MATROIDS. In this section we review some background for oriented matroids. For a more thorough introduction to oriented matroids, the interested reader is directed to [1].

2.1. ***Cryptomorphic definitions.*** Recall that cryptomorphic definitions are two (or more) definitions that are equivalent, but not obviously equivalent. In the `OrientedMatroids` package, the user is able to construct oriented matroids using one of three cryptomorphic definitions which we will discuss in this section. Before going through the definitions, we introduce the notions of signed subsets and signed vectors as they will be the tools used to define oriented matroids.

Let E be some set. A *signed set* is a pair $X = (X^+, X^-) \in E \times E$ such that $X^+ \cap X^- = \varnothing$ and $X^+ \cup X^- = E$. The elements of X^+ are known as the *positive elements of* X and the elements of X^- are called the *negative elements of* X. A *signed subset* is then a signed set (X^+, X^-) such that $X^+ \cup X^- \subseteq E$. For ease of notation we let $\underline{X} = X^+ \cup X^-$ and call it the *support of* X. This allows for interpreting a signed subset as a signed set X such that $\underline{X} \subseteq E$. The set $X^0 = E \setminus \underline{X}$ is known as the *zero set of* X. By letting $-X = (X^-, X^+)$, we obtain a signed set whose orientation is the reverse of X. For ease of notation, for two signed subsets X and Y, we let $X = \pm Y$ mean that either $X = Y$ or $X = -Y$. For $e \in E$ we let $X(e)$ denote the sign of e in X, in other words, we let $X(e) = +$ if $e \in X^+$, $X(e) = -$ if $e \in X^-$ and $X(e) = 0$ if $e \in X^0$.

By ordering E we may encode signed subsets as signed vectors. For an ordering $i_1 < \ldots < i_n$ of E, we let a *signed vector* X be a vector in $\{+, 0, -\}^E$ where the j-th component is equal to $X(i_j)$. By an abuse of notation we let X represent both the signed subset and the signed vector. For a signed vector X, let X_e denote the e-th component, *i.e.*, $X_e = X(e)$.

Example 1. Suppose that $E = \{1, 2, 3, 4, 5, 6, 7\}$. Let $X = \{\{2, 6, 7\}, \{1, 5\}\}$ be a signed subset. The positive elements of X are 2, 6, and 7 and the negative elements are 1 and 5. The signed vector associated to X is the vector $(-, +, 0, 0, -, +, +)$. The signed vector associated to $-X$ is the vector $(+, -, 0, 0, +, -, -)$.

Circuit oriented matroids. The first definition of an oriented matroid is a generalisation of directed graphs and encodes the structure of a directed graph using the circuits. For a set of signed subsets $\mathcal{C}$ we let $-\mathcal{C} = \{-X \mid X \in \mathcal{C}\}$.

Definition 1. An oriented matroid $\mathcal{M}$ is a pair $(E, \mathcal{C})$ where E is a set and $\mathcal{C}$ is a collection of signed subsets of E such that the following hold:

(C1) $\varnothing \notin \mathcal{C}$,

(C2) $\mathcal{C} = -\mathcal{C}$,

(C3) for all $X, Y \in \mathcal{C}$, if $\underline{X} \subseteq \underline{Y}$ then $X = \pm Y$, and

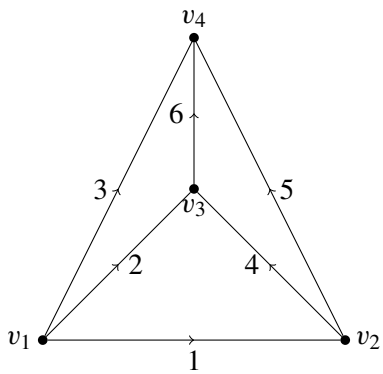
(C4) for all $X, Y \in \mathcal{C}$, $X \neq -Y$ and $e \in X^+ \cap Y^-$ there exists $Z \in \mathcal{C}$ such that

$$Z^+ \subseteq \left(X^+ \cup Y^+\right) \setminus \{e\} \text{ and } Z^- \subseteq \left(X^- \cup Y^-\right) \setminus \{e\}$$

The set $\mathcal{C}$ is called the *set of signed circuits of* $\mathcal{M}$.

Note that the four properties encode precisely the circuit structure of an arbitrary directed graph. The first property (C1) states that the empty circuit is not included in our set of (signed) circuits. The second property (C2) states that for every circuit X in $\mathcal{C}$, the reverse of the circuit X is also a circuit and should be included in our set $\mathcal{C}$. The third property (C3) states that our circuits are simple (no circuit contains some other circuit). The fourth property (C4) is normally known as the (weak) elimination axiom and states that if we have two circuits X and Y with an edge e in common (but with opposite signs) then there is another circuit Z which is a subset of the edges of X and Y which does not contain the edge e. These properties can be verified in the following example.

Example 2. Let G be the directed graph:



Denoting by $\overline{x}$ the reversal of a directed edge, there are 14 (simple) cycles given by the following 7 cycles and their reversals.

$$15\overline{3} \qquad 3\overline{62} \qquad 46\overline{5} \qquad 14\overline{2} \qquad 15\overline{62} \qquad 3\overline{641} \qquad 3\overline{5}4\overline{2}$$

These 14 cycles are precisely the 14 circuits which define the set of signed circuits of an oriented matroid on the set of 6 edges.

To verify (C4), one can take the circuits $X = 14\overline{2}$ and $Y = \overline{3}26$. Since the edge labelled 2 is negative in X and positive in Y it satisfies the conditions of (C4), *i.e.*, $2 \in X^- \cup Y^+$. Then taking Z^+ to be $(X^+ \cup Y^+) \setminus \{2\} = \{1, 4, 6\}$ and Z^- to be $(X^- \cup Y^-) \setminus \{2\} = \{\overline{}\}$ gives us the circuit $Z = \overline{3}641$.

Vector Oriented Matroids. To define an oriented matroid through vectors, we must introduce the composition operation. We define composition using sign vectors. Given two signed vectors X and Y, the *composition* $X \circ Y$ is given component-wise:

$$(X \circ Y)_e = \begin{cases} X_e & \text{if } X_e \neq 0 \\ Y_e & \text{otherwise.} \end{cases}$$

A *vector* of an oriented matroid is some composition of circuits.

Definition 2. An oriented matroid $\mathcal{M}$ is a pair $(E, \mathcal{V})$ where E is a set and $\mathcal{V}$ is a collection of signed subsets of E such that the following hold:

(V1) $\varnothing \in \mathcal{V}$,

(V2) $\mathcal{V} = -\mathcal{V}$,

(V3) for all $X, Y \in \mathcal{V}$, then $X \circ Y \in \mathcal{V}$,

(V4) for all $X, Y \in \mathcal{V}$, $e \in X^+ \cap Y^-$ and $f \in (\underline{X} \setminus \underline{Y}) \cup (\underline{Y} \setminus \underline{X}) \cup (X^+ \cap Y^+) \cup (X^- \cap Y^-)$, there is $Z \in \mathcal{V}$ such that

$$Z^+ \subseteq \left(X^+ \cup Y^+\right) \setminus \{e\}\,,\ Z^- \subseteq \left(X^- \cup Y^-\right) \setminus \{e\}\,, \text{ and } f \in (Z^+ \cup Z^-)$$

The set $\mathcal{V}$ is called the *set of vectors of* $\mathcal{M}$.

The equivalence of this definition with the circuit axiom definition is due to [2] and [4].

Example 3. Continuing with Example 2, taking the composition of all the circuits will give us the set of vectors. As an example of a composition let us take the composition of $1\overline{53}$ and $3\overline{62}$. The sign vectors for these two cycles are given by $(+, 0, -, 0, +, 0)$ and $(0, -, +, 0, 0, -)$ respectively. Then

$$(+, 0, -, 0, +, 0) \circ (0, -, +, 0, 0, -) = (+, -, -, 0, +, -), \text{ and}$$
$$(0, -, +, 0, 0, -) \circ (+, 0, -, 0, +, 0) = (+, -, +, 0, +, -)$$

giving us two different signed vectors. The associated signed subsets are given by $1\overline{2}3\overline{56}$ and $1\overline{2}35\overline{6}$. Taking all such vectors using composition gives us the set of vectors of an oriented matroid (the same oriented matroid found in Example 2).

Covector Oriented Matroids. Before we define a covector, we must first define what the dual of an oriented matroid is. We say two signed subsets X and Y are *orthogonal*, denoted $X \perp Y$, if $\underline{X} \cap \underline{Y} = \varnothing$ or there exist $e, f \in \underline{X} \cap \underline{Y}$ such that $X(e) \cdot Y(e) = -X(f) \cdot Y(f)$. Let $\mathcal{M} = (E, \mathcal{C})$ be an oriented matroid with circuits $\mathcal{C}$. Then there is a unique set $\mathcal{C}^*$ of signed subsets of E such that $X \perp Y$ for all $X \in \mathcal{C}$ and $Y \in \mathcal{C}^*$. Then $\mathcal{C}^*$ are the circuits of another oriented matroid on E called the *dual of* $\mathcal{M}$, denoted $\mathcal{M}^*$. The circuits of $\mathcal{M}^*$ are called the *cocircuits* of $\mathcal{M}$. Similarly, the vectors of $\mathcal{M}^*$ are called the *covectors* of $\mathcal{M}$. As covectors are signed vectors themselves, this allows for another definition for oriented matroids.

Definition 3. An oriented matroid $\mathcal{M}$ is a pair $(E, \mathcal{L})$ where E is a set and $\mathcal{L}$ is a collection of signed vectors of E, such that the following hold:

(CV1) $(0, \ldots, 0) \in \mathcal{L}$,

(CV2) $\mathcal{L} = -\mathcal{L}$,

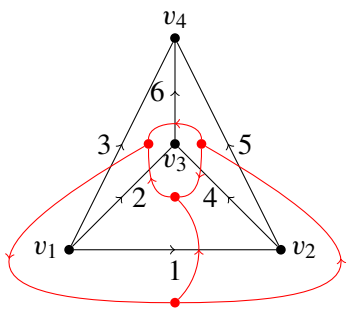
(CV3) $X, Y \in \mathcal{L}$ implies $X \circ Y \in \mathcal{L}$,

(CV4) if $X, Y \in \mathcal{L}$ and $e \in S(X, Y)$ then there exists $Z \in \mathcal{L}$ such that $Z_e = 0$ and $Z_f = (X \circ Y)_f = (Y \circ X)_f$ for all $f \notin S(X, Y)$,

where $S(X, Y) = \{e \in E \mid X_e = -Y_e \neq 0\}$ is known as the *separation set* of X and Y. The set $\mathcal{L}$ is called the *set of covectors of* $\mathcal{M}$.

The equivalence of this definition with the circuit axioms comes from the fact that these axioms are (essentially) the same axioms as the vector definition axioms. The main difference is the final axiom which has been replaced by an equivalent axiom (shown to be equivalent by [3] and [4]).

Example 4. The dual of the oriented matroid in Example 2 is the oriented matroid associated to the dual (di)graph of G. To obtain the dual digraph $G^\star$ of G, we first draw both G and the dual graph $G^\star$ of G in the plane. We then direct each edge of $G^\star$ so that there is a counter-clockwise turn of the corresponding arc in G. The dual digraph is the digraph given in red:



2.2. ***Properties and associated objects.*** In the `OrientedMatroids` package, we introduce a number of different properties and associated objects to oriented matroids. We define these properties and associated objects next.

In this section, let $\mathcal{M}$ be an oriented matroid on a set E using any of the (equivalent) definitions. We start by some definitions on E itself.

Definitions 1. The set E is known as the *ground set* of $\mathcal{M}$. An element $e \in E$ is known as a *loop* if for all signed vector (or signed subset) X in the oriented matroid one has $X_e = 0$ ($X(e) = 0$). Two elements e and f in E are *parallel to one another* if for all signed vector (or signed subset) X in the oriented matroid, then $X_e = \pm X_f$ ($X(e) = \pm X(f)$).

As with many mathematical objects, it is useful to use one oriented matroid to obtain another. Two well-known oriented matroids obtained from a given oriented matroid are the deletion and restriction of an oriented matroid. We describe these two methods next.

Definitions 2. The *deletion* of an oriented matroid is the oriented matroid whose signed subsets/vectors are restricted to the set $E \setminus A$ for some subset $A \subseteq E$. The set A is known as the *change set* in our methods. The *restriction* (more commonly known as a *contraction*) of an oriented matroid is the oriented matroid whose signed subsets/vectors are restricted to the set $E \setminus A$ for some subset $A \subseteq E$ with the additional requirement that only signed subsets/vectors whose zero set is a subset of A are included.

Oriented matroids have certain partial orders associated with them which allow us to better understand their structures. For an oriented matroid $\mathcal{M} = (E, \mathcal{V})$ defined using the covector axioms, the *face poset* of $\mathcal{M}$ is the partial order on the set of covectors $\mathcal{V}$ ordered component-wise by $0 < +$ and $0 < -$. The all-zero vector is the bottom element and the join of two elements X and Y is equal to $X \circ Y = Y \circ X$ if the separation set of X and Y is empty. If we adjoin a top element then the face poset is known as the *face lattice*. The maximal elements of a face poset are known as *topes*. There is an additional partial order one may place on the topes called the *tope poset* which fixes some tope B and lets $T \leq T'$ if and only if $S(B, T) \subseteq S(B, T')$ (this poset was first described in [4]).

We end this section by defining some properties which an oriented matroid might possess.

Definitions 3. The *rank* of an oriented matroid is the rank of its underlying matroid. If an oriented matroid does not contain any loops or (distinct) parallel elements, then we say it is *simple*. An oriented matroid is said to be *acyclic* if none of its circuits are an all positive (all elements are positive and none are negative) or if, equivalently, there exists a positive tope. A tope T is called *simplicial* if the interval $[0, T]$ in the face poset is isomorphic to a Boolean lattice. If every tope is simplicial then we call the oriented matroid *simplicial*.

3. The OrientedMatroids package.

The `OrientedMatroids` package allows us to instantiate oriented matroids and retrieve various properties. The package is designed in such a way to allow the implementation of additional definitions in the future. We break the package down into the essential components and talk about each class individually.

3.1. *The `AbstractOrientedMatroid` class and children.*

The `AbstractOrientedMatroid` class is the abstract class that all other types of oriented matroids are based on. Any new type of oriented matroid coming from a cryptomorphic definition will extend this class so that all functionality of oriented matroids are present

When extending the `AbstractOrientedMatroid` class, there is only one required method that must be present. This method is the `is_valid(self)` method and returns a boolean whether or not the given data satisfies the properties of that particular oriented matroid definition. There are currently four classes which extend the `AbstractOrientedMatroid` class. These are:

(1) `CircuitOrientedMatroid` class, which uses the circuit definition of oriented matroids,

(2) `VectorOrientedMatroid` class, which uses the vector definition of oriented matroids,

(3) `CovectorOrientedMatroid` class, which uses the covector definition of oriented matroids, and

(4) `RealHyperplaneArrangementOrientedMatroid` class, which extends the `CovectorOriented Matroid` class for faster computation in the case of hyperplane arrangements.

As there are multiple cryptomorphic definitions, the `AbstractOrientedMatroid` class has a method `convert_to(self, new_type=None)` which allows you to convert from one type to another type. This has the effect of changing the `elements()` method (see Section 3.1) to the new defining type. An example

usage of this is the following where we first define an oriented matroid using the vector axioms and then convert to the same oriented matroid using circuit axioms:

```
sage: M = OrientedMatroid([[1],[-1],[0]], key='vector')
sage: M.convert_to('circuit')
Circuit oriented matroid of rank 0
```

In this case, before converting, the method `M.elements()` would return the vectors of the oriented matroid, but after converting `M.elements()` would return the circuits of the oriented matroid.

In addition to the above, our class has the following additional methods.

Element methods. The following methods are present in regard to getting the elements of an oriented matroid:

- `groundset(self)` – Return the ground set E that the oriented matroid is defined over.
- `elements(self)` – Return the (defining) elements of the oriented matroid.
- `an_element(self)` – Return an arbitrary element of the oriented matroid.
- `circuits(self)` – Return the circuits of the oriented matroid.
- `cocircuits(self)` – Return the cocircuits of the oriented matroid.
- `vectors(self)` – Return the vectors of the oriented matroid.
- `covectors(self)` – Return the covectors of the oriented matroid.
- `loops(self)` – Return the elements which are loops of the oriented matroid.
- `are_parallel(self, e, f)` – Return whether or not the two elements are parallel in the oriented matroid.

In the above, by *defining element* we mean the collection used to define the oriented matroid. For example, if the oriented matroid `M` was constructed using the circuit axioms, then `M.elements()` would return the set of circuits.

Associated object methods. The following methods are present in regard to objects associated to an oriented matroid:

- `deletion(self, change_set)` – Return the deletion of an oriented matroid based on a change set.
- `dual(self)` – Return the dual of the oriented matroid.
- `face_poset(self, facade=False)` – Return the face poset of an oriented matroid.
- `face_lattice(self, facade=False)` – Return the face lattice of an oriented matroid.
- `matroid(self)` – Return the underlying matroid of the oriented matroid.
- `restriction(self, change_set)` – Return the restriction of an oriented matroid based on a change set.

- `topes(self)` – Return the topes of the oriented matroid.
- `tope_poset(self, base_tope, facade=False)` – Return the tope poset based off a particular `base_tope`.

Property methods. The following methods are present in regards to properties an oriented matroid might have:

- `rank(self)` – Return the rank of an oriented matroid.
- `is_acyclic(self)` – Return whether or not the oriented matroid is acyclic.
- `is_simple(self)` – Return whether or not the oriented matroid is simple.
- `is_simplicial(self)` – Return whether or not the oriented matroid is simplicial.

3.2. ***Speed and efficiency of package.*** The `OrientedMatroids` package has a lot of room for improvement, in particular in the construction and verification of whether an object is an oriented matroid or not; *i.e.*, the `is_valid` methods. Testing various examples on computers, the `OrientedMatroids` package can handle oriented matroids of rank up to 8 within a reasonable amount of time. For example, the following piece of code on the first author's personal computer took roughly 49 seconds using an Intel i9-13900HX processor on Windows 11 (using Windows Subsystem for Linux and SageMath 10.6):

```
sage: G = Graph({1:[2,4],2:[3,4,5],3:[4,6,8],4:[7],5:[8]})
sage: A = hyperplane_arrangements.graphical(G)
sage: M = OrientedMatroid(A); M
Hyperplane arrangement oriented matroid of rank 7
```

In this instance, the conversion from a digraph to circuits takes roughly $O(n\log(n))$ time and the time complexity for the `is_valid` method is roughly $O(n^4)$. A future goal is to improve efficiency and speed to allow for the handling of oriented matroids of rank greater than 8.

4. EXAMPLES.

4.1. ***Instantiation and use.*** As with matroids in SageMath, the `OrientedMatroids` package uses a helper function to instantiate an oriented matroid based off input data. To instantiate an oriented matroid we have the following function:

```
def OrientedMatroid(data=None, groundset=None, key=None, **kwds):
```

The input is defined as follows:

- `data` – The data will be used to define the oriented matroid itself. It can either be a list/tuple of `SignedSubsetElement` objects from this package or tuples with positive, negative and zero sets. Alternatively, it can be a hyperplane arrangement, point configuration or digraph object from SageMath.
- `groundset` – The ground set E that the oriented matroid will be based off of. If this is not provided, the package attempts to figure out the groundset based off the data provided.

- key – The key is the type of oriented matroid your data is. This can be either covector, vector, or circuit. If nothing is passed in, then the package will attempt to figure out the key based off the data.

The following code is a use case for using the package on various objects.

Digraphs: The example in Example 2 can be encoded in the following way:

```
sage: d = {'v1':{'v2':1,'v3':2,'v4':3},'v2':{'v3':4,'v4':5},'v3':{'v4':6}}
sage: M = OrientedMatroid(DiGraph(d),key="circuit"); M
Circuit oriented matroid of rank 3
sage: len(M.circuits())
14
```

Hyperplane arrangements:

```
sage: A = hyperplane_arrangements.braid(3)
sage: M = OrientedMatroid(A); M
Hyperplane arrangement oriented matroid of rank 2
sage: M.groundset()
(Hyperplane 0*t0 + t1 - t2 + 0,
 Hyperplane t0 - t1 + 0*t2 + 0,
 Hyperplane t0 + 0*t1 - t2 + 0)
sage: AbstractOrientedMatroid.options.display='vector'
sage: M.elements()
[(0,0,0),
 (0,1,1),
 (0,-1,-1),
 (1,0,1),
 (1,1,1),
 (1,-1,0),
 (1,-1,1),
 (1,-1,-1),
 (-1,0,-1),
 (-1,1,0),
 (-1,1,1),
 (-1,1,-1),
 (-1,-1,-1)]
sage: M.is_acyclic()
True
```

Point configurations:

```
sage: PC = [[1,0,0],[0,1,0],[0,0,1],[1/2,1/2,0],[0,1/2,1/2],[1/3,1/3,1/3]]
sage: M = OrientedMatroid(PointConfiguration(PC)); M
Circuit oriented matroid of rank 3
sage: M.matroid()
Matroid of rank 3 on 6 elements with 16 bases
```

Non-example: If the user attempts to construct an oriented matroid that is invalid, the package will automatically mention that an error occurred, and in particular which error. In terms of the covector axioms (see Definition 3), the following gives (non-)examples for three of the axioms.

```
sage: CV2 = [ [0,0],[1,1]]
sage: OrientedMatroid(CV2, key='covector')
Traceback (most recent call last):
...
ValueError: Every element needs an opposite

sage: CV3 = [[1,1],[-1,-1],[0,1],[1,0],[-1,0],[0,-1]]
sage: OrientedMatroid(CV3, key='covector')
Traceback (most recent call last):
...
ValueError: Composition must be in vectors

sage: CV4 = [ [0,0],[1,1],[-1,-1],[1,-1],[-1,1]]
sage: M = OrientedMatroid(CV4, key='covector'); M
Traceback (most recent call last):
...
ValueError: weak elimination failed
```

4.2. ***Testing of conjectures.*** The OrientedMatroids package can also handle testing more complex examples and can help with trying to find counterexamples/examples of conjectures. An example of a conjecture that the OrientedMatroids package can currently help with is the search for counterexamples in the following conjecture by Las Vergnas. We first define the tope graph before stating the conjecture. The *tope graph* $\mathcal{T}$ of an oriented matroid is a graph whose vertices are the topes and an edge between two topes if they cover the same element in the face poset. Alternatively, one can think of the tope graph as the underlying graph of the Hasse diagram of the tope poset. It is known that for an oriented matroid of rank r, then a tope T is simplicial if and only if T has exactly r neighbours in the tope graph $\mathcal{T}$.

Conjecture 4.1. *Every* (*simple*) *oriented matroid of rank* r *contains a simplicial tope.*

The following code shows that the conjecture is true for oriented matroids coming from graphical arrangements where the graph has 5 nodes. Note that it requires the `nauty` package to be installed into Sage.

```
sage: n = 5
sage: for G in graphs.nauty_geng('-c ' + str(n)):
sage:     A = hyperplane_arrangements.graphical(G)
sage:     M = OrientedMatroid(A)
sage:     if not any([i.is_simplicial() for i in M.topes()]):
sage:         print("Found graph without simplicial region")
sage: print("Done")
Done
```

Depending on the ability of the computer used, one can construct oriented matroids to test whether Las Vergnas' conjecture is true for more complicated oriented matroids; something that would currently be done by hand.

SUPPLEMENT. The online supplement contains the distribution directory for `OrientedMatroid`, version 0.2.

REFERENCES.

[1] A. Björner, M. L. Vergnas, B. Sturmfels, N. White, and G. M. Ziegler, *Oriented matroids*, 2nd ed., Encyclopedia of mathematics and its applications **46**, Cambridge University Press, 1999.

[2] R. G. Bland and M. Las Vergnas, "Orientability of matroids", *J. Comb. Th. B* **24**:1 (1978), 94–123.

[3] K. Fukuda, *Oriented matroid programming*, PhD thesis, University of Waterloo, Waterloo, ON, Canada, 1982, available at https://people.inf.ethz.ch/~fukudak/Doc_pub/fukuda1982thesis.pdf.

[4] A. Mandel, *Topology of oriented matroids*, PhD thesis, University of Waterloo, Waterloo, ON, Canada, 1981, available at https://www.researchgate.net/profile/Arnaldo-Mandel/publication/244517387_Topology_of_Oriented_Matroids/links/55ea5ccd08aeb6516265e4d5/Topology-of-Oriented-Matroids.pdf.

[5] *SageMath, the Sage Mathematics Software System* (*Version* 10.6.0), 2025. `https://www.sagemath.org`.

RECEIVED: 21 Nov 2024 REVISED: 25 Jun 2025 ACCEPTED: 28 Jul 2025

ARAM DERMENJIAN:

aram.dermenjian.math@gmail.com
University of Manchester, Manchester, M13 9PL, United Kingdom

ELIZABETH FLIGHT:

lizzyflight@outlook.com
University of Manchester, Manchester, M13 9PL, United Kingdom

TUDOR TANASA:

tudor.tanasa@student.manchester.ac.uk
University of Manchester, Manchester, M13 9PL, United Kingdom