Grassmannian as Continuous Abstract Data Type with Computable Semantics*

Seokbin Lee**, Donghyun Lim, Sewon Park, and Martin Ziegler

KAIST, School of Computing

Abstract. The Grassmannian (over d-dimensional Euclidean space) is the collection of all homogeneous subspaces of $\mathbb{R}^d$, equipped with the operations Minkowski sum (aka join), intersection (aka meet), orthogonal complement, and projection. From dimension $d = 2$ on, these operations are discontinuous and therefore uncomputable. We show that, when the integral dimension of the join/meet is given as promise, they do become computable: by devising devise algorithms in the *Exact Real Computation* paradigm.

1 Introduction

1.1 Computable Analysis

In classical theory of algorithms and computation, the choice of computation model is not crucial since it does not affect computability and little affects computational complexity. However, this model-agnosticism does not hold if the problems of interest are of continuous nature, as opposed to discrete.

There are two major models of computation that deals with continuous data. One is the BSS machine, an algebraic model where basic arithmetic operations over real numbers are taken to be primitive, being computable in a single time step [BSS89]. BSS machine model has one significant limitation: it is not implementable. No Turing machine can simulate a BSS machine. Another weakness is that the exponential map is not computable by any BSS machine [Bra03].

The other model, which we assume as *the model*, is computable analysis. It is an analytic model, where computation over real numbers are defined as arbitrary approximation [Wei00]. Unlike BSS machine model, it has the virtue of being implementable.

Computable analysis defines that a function $f : \mathbb{R}^m \to \mathbb{R}^n$ is *computable* if there exists a Turing machine that, for every $x \in \mathbb{R}^m$, given any sequence $(a_n) \subseteq \mathbb{Q}$ satisfying $d(x, a_n) \leq 2^{-n}$, produces a sequence $(b_n) \subseteq \mathbb{Q}$ such that $d(f(x), b_n) \leq 2^{-n}$.

According to this definition, however, the inequality test $\neq: \mathbb{R}^2 \to \{0, 1\}$ is not decidable but only semideciable [Wei00]. In other words, when testing equality of two real numbers x and y:

* Supported by the National Research Foundation of Korea (grant NRF-2017R1E1A1A03071032) and by the International Research & Development Program of the Korean Ministry of Science and ICT (grant NRF-2016K1A3A7A03950702).

** Corresponding author

– if actually $x = y$, then the test will get stuck;
– if actually $x \neq y$, then the test will return 0.

Similarly, when testing $x < y$,

– if actually $x < y$, then the test will return 1;
– if actually $x > y$, then the test will return 0;
– if actually $x = y$, then the test will get stuck.

These are theoretical limitations in the same sense that undecidability of halting problem is a theoretical limitation. There is no way to avoid this phenomenon by adding more layers to abstractions. A programmer should be well aware of this issue when writing a program that deals with real numbers. It is the programmer's responsibility to accept this limitation and still somehow write a program.

1.2 Exact Real Computation

Exact Real Computation (ERC) is an imperative language based on the philosophy of computable analysis. The language provides the set of real numbers as its primitive datatype R. The datatype is equipped with the operators $(+, -, \times, /)$ which represent the field operators of reals. We say that the datatype R represents the exact real numbers in the sense that the operators come with no rounding errors in their evaluations.

ERC provides also a datatype for integers and instructions for conditional branching and while loop. The detailed construction of the language can be found in [BCK+16]. We illustrate in this section only two crucial differences to other common languages:

– All operations are exact and devoid of rounding errors.
– Comparisons "x`>0`" of real numbers are *partial*, in that they return `true` in the case $x > 0$, `false` in the case $x < 0$, but do not return ('freeze', get 'stuck') in the case $x = 0$, since this is undecidable as mentioned above.
– To write total programs, the multivalued/non-extensional lazy `choose` operation is essential: $\mathbf{select}(x_0 > y_0, \ldots, x_d > y_d)$ will return *any* integer $j \in \{0, \ldots, d\}$ satisfying $x_j > y_j$, provided such exists — not necessarily reproducibly.

By combining both, we can express various multivalued operators:

Example 1 (Soft Sign). Given a real number x and a positive integer k, $\mathbf{select}(x < 2^{-k}, x > -2^{-k})$ returns 0 when $x \leq -2^{-k}$, 1 when $x \geq 2^{-k}$, and 0 or 1 arbitrarily when $-2^{-k} < x < 2^{-k}$. We may interpret k as a tolerance factor that when $|x| \geq 2^{-k}$, the semantic is exactly the sign of x; but, when $|x|$ is small relative to 2^{-k}, the semantic contain both 0 and 1. See also [YSS13].

Another example, consider a given finite list of real numbers $x_1, \cdots, x_d$ with a promise that at least one of them is not zero. Then we can find *some* j such that $x_j \neq 0$. This provides a computable relaxation of classical but uncomputable pivot search to be used in Section 2.3.

1.3 Grassmannian

The Grassmannian $\mathcal{G}_k^d$ is the collection of all k-dimensional homogeneous subspaces of $\mathbb{R}^d$. $\mathcal{G}^d$ is the collection of all homogeneous subspaces, equipped with operations Minkowski sum, intersection, orthogonal complement, and projection. We consider each subspace of $\mathbb{R}^d$ represented by some basis. Computation of operations on the Grassmannian is defined with respect to such a basis representation. For example, an algorithm computing the Minkowski sum has the following specification:

INPUT 1 m linearly independent vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m \in \mathbb{R}^d$
INPUT 2 n linearly independent vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d$
OUTPUT ℓ linearly independent vectors $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_\ell \in \mathbb{R}^d$ satisfying

$$\mathrm{lspan}\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_\ell\} = \mathrm{lspan}\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$$

Other operations on the Grassmannian have similar specifications.

These operations, however, as naively specified above, are not computable. We can reduce equality testing, which is undecidable, to those operations. For $a, b \in \mathbb{R}$, we have $a = b$ if and only if the Minkowski sum of $\{(1, a, 0, 0, \cdots, 0)\}$ and $\{(1, b, 0, 0, \cdots, 0)\}$ is of dimension 1.

One way to make those operations computable is by changing the specification so that algorithms be given the dimension of output space as part of input.

INPUT 1 m linearly independent vectors $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m \in \mathbb{R}^d$
INPUT 2 n linearly independent vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d$
INPUT 3 a number $l = \dim(\mathrm{lspan}\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\})$
OUTPUT ℓ linearly independent vectors $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_\ell \in \mathbb{R}^d$ satisfying

$$\mathrm{lspan}\{\boldsymbol{w}_1 \cdots \boldsymbol{w}_\ell\} = \mathrm{span}\{\boldsymbol{u}_1, \cdots, \boldsymbol{u}_m, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$$

This paper presents algorithms in ERC that compute operations on the Grassmannian with respect to the latter specification.

2 Algorithms in ERC

Class `GRASSMANN` consists of the following variables:

- integer d for ambient dimension
- integer m for subspace dimension
- $d \times m$-dimensional array of reals with basis of the subspace

Furthermore it consists of the following methods, specified and realized by algorithms as described in the sequel: constructor, join (Minkowski sum), meet (intersection), orthogonal complement, and projection.

Algorithm 1 Modified Gaussian Elimination

procedure GAUSSIAN(A, k) ▷ A has dimensions $m \times n$
  for i from 1 to k **do** ▷ k is the number of iterations
    $j \leftarrow$ **select**$(|A_{i,i}| > 0, \ldots, |A_{i,n}| > 0)$
    Swap c_i and c_j of A ▷ c_i denotes the i-th column of A
    for p from $i+1$ to d **do**
      $c_p \leftarrow c_p - \frac{A_{i,p}}{A_{i,i}} c_i$
  return A

The algorithms avoid undecidability of inequality by cleverly exploiting the information of output dimension provided as part of input. They will also heavily utilize a modified form of columnwise Gaussian Elimination, described below.

We remark that, by "swap c_1 and c_j," we mean that the entries in the respective columns should be swapped one by one; in actual implementation this would require an iteration of n times. For simplicity, we omitted this detail in the pseudocode.

2.1 Constructor

The constructor receives ambient dimension d and non-zero vector $\boldsymbol{u} \in \mathbb{R}^d$, verifies the latter to be non-zero, and returns $\mathrm{lspan}(\boldsymbol{u}) \in \mathcal{G}_1^d$.

2.2 Complement

This method receives a subspace (object of class `GRASSMANNIAN`), computes a basis of its orthogonal complement using a modification of Gaussian Elimination, and returns that subspace.

Specifically, let $A \in \mathbb{R}^{d \times m}$ denote the given matrix whose columns form a basis of the subspace under consideration. Classical linear algebra augments it to an $(m+d) \times d$ matrix $\begin{bmatrix} A^T \\ I_d \end{bmatrix}$, where I_d denotes the $d \times d$ identity matrix; transform the latter to column echelon form $\begin{bmatrix} B \\ C \end{bmatrix}$: Then those columns of C whose counterparts in B vanish are classically known to constitute a basis of the kernel of A, that is, of the orthogonal complement of A's range.

Algorithm 2 Complement Pseudocode

procedure COMPLEMENT(A, m, d)
  $M \leftarrow \begin{bmatrix} A^T \\ I_d \end{bmatrix}$ ▷ I_d is the $d \times d$ identity matrix
  $M \leftarrow \mathrm{Gaussian}(M, m)$
  return $M[m+1 : m+d, m+1 : d]$ ▷ return the last d rows and $d-m$ columns of M

We make some remarks. We note that, since it is enough to make the first m rows of A^T in echelon form, we need only loop the column index i from 1 to

m. Also, we are guaranteed that the **select** method used in the pseudocode will terminate and return correctly, because of the promise that the rows of A are linearly independent and hence A being full rank.

2.3 Join

This method receives two subspaces (objects of class `GRASSMANNIAN`) and the integer dimension ℓ of their Minkowski sum. It verifies the subspaces to have same ambient dimension, then computes a basis of their Minkowski sum using a modification of Zassenhaus' Algorithm, and returns that subspace.

Specifically, let $A \in \mathbb{R}^{d\times m}$ denote the given matrix whose columns form a basis of the first subspace under consideration; and $B \in \mathbb{R}^{d\times n}$ similarly for the second subspace. Then transform the $2d \times (m+n)$ matrix $\left[\begin{array}{c|c} A & B \\ \hline A & 0 \end{array}\right]$ to column echelon form $\left[\begin{array}{c|c|c} C & 0 & 0 \\ \hline * & D & 0 \end{array}\right]$ where C is of format $d \times \ell$ with all columns non-zero. Then the columns of C form a basis of the Minkowski sum as sought.

Algorithm 3 Join Pseudocode

```
procedure JOIN(A, B, l)
    M ← [A|B; A|0]
    M ← Gaussian(M, m + n)
    return M[1 : d, 1 : l]
```

As in Subsection 2.2, pivot search in Gaussian Elimination can be adjusted based on the promise that C has the known number ℓ of columns all non-zero.

2.4 Meet

Similarly to Subsection 2.3, but now the matrix D of format $d\times k$ with all columns non-zero constitutes a basis of the intersection of known dimension k of the two subspaces. Note that the dimension of the Minkowski sum $\ell = m+n-k$ is known as well.

Algorithm 4 Meet Pseudocode

```
procedure MEET(A, B, k)
    M ← [A|B; A|0]
    M ← Gaussian(M, m + n)
    return M[d + 1 : 2d, m + n − k + 1 : m + n]
```

2.5 Projection

This method receives a subspace (object of class `GRASSMANNIAN`), represented as a matrix $A \in \mathbb{R}^{d \times m}$ and another subspace $B \in \mathbb{R}^{d \times n}$ to be projected onto A. It verifies that d coincides with the ambient dimension of the subspace. The resulting dimension of the projected subspace, l, is also provided.

Algorithm 5 Projection Pseudocode

procedure PROJECTION(A, B, l)
  $P \leftarrow A(A^TA)^{-1}A^T$ ▷ We are guaranteed the existence of $(A^TA)^{-1}$ because A^TA is regular
return Gaussian$(PB, l)[1 : d, 1 : l]$

To elaborate, we know that A has full column rank, therefore so does the matrix A^TA, which implies that it is invertible and therefore an inverse can be computed via Gaussian Elimination. Once again, we omit the details for simplicity.

2.6 Implementation in `iRRAM`

The source code is available at
https://github.com/realcomputation/irramplus/tree/master/GRASSMANN

References

[BCK+16] Franz Braußе, Pieter Collins, Johannes Kanig, SunYoung Kim, Michal Konecny, Gyesik Lee, Norbert Müller, Eike Neumann, Sewon Park, Norbert Preining, and Martin Ziegler. Semantics, logic, and verification of "exact real computation", 2016.

[Bra03] Vasco Brattka. The emperor's new recursiveness: The epigraph of the exponential function in two models of computability. In Masami Ito and Teruo Imaoka, editors, *Words, Languages & Combinatorics III*, pages 63–72, Singapore, 2003. World Scientific Publishing. ICWLC 2000, Kyoto, Japan, March 14–18, 2000.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. On a theory of computation and complexity over the real numbers: *np*- completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc. (N.S.)*, 21(1):1–46, 07 1989.

[Wei00] Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

[YSS13] Chee Yap, Michael Sagraloff, and Vikram Sharma. Analytic root clustering: A complete algorithm using soft zero tests. In *Conference on Computability in Europe*, pages 434–444. Springer, 2013.