Adv. Appl. Clifford Algebras (2025) 35:51

https://doi.org/10.1007/s00006-025-01403-9

Advances in
Applied Clifford Algebras

Clifford Algebra in R: Introducing the `Clifford` Package

Robin K. S. Hankin*

Communicated by Dietmar Hildenbrand

Abstract. Here I present the `clifford` package for working with Clifford algebras in the R programming language. Algebras of arbitrary dimension and signature can be manipulated, and a range of different multiplication operators is provided. The algebra is described and package idiom is given; it obeys `disordR` discipline. A case-study of conformal algebra is presented. The package is available on CRAN and development versions are hosted at github.

Mathematics Subject Classification. Primary 15A66, 16W50, 20C05, 20C40, 20D15; Secondary 68W30.

Keywords. Class file, HTML, Journal.

Contents

*Corresponding author.

Published online: 13 October 2025

1. Introduction

Clifford algebras [24] are interesting and instructive mathematical objects, with a rich structure that has wide-ranging applications to physics. Clifford algebra has attracted much computational support from a very wide range of perspectives [7]. Examples would include purely symbolic support for well-established proprietary systems such as Mathematica [3] or Maple [1]; or open-source systems such as sage [33,36] and *sympy* [29]. Computational support for Clifford algebras can take many forms: Gaigen [10] and Gaalop [26], for example, generate C++ implementations of geometric algebra; CliffoSor [11] offers direct hardware support for Clifford operators.

Here I introduce the `clifford` package, written in the R computing language [32], which furnishes functionality for working with Clifford algebras. The R language is easily augmented to manipulate symbolic mathematics and examples would include finite group theory [14], Weyl algebra [17], and knot theory [22]. As of 2024, no other R functionality for Clifford algebra is available. Advantages of working with the R language include extensive statistical capabilities, platform independence, and support for reproducible research. The package is available on CRAN at `https://CRAN.R-project.org/package=clifford` and development versions are hosted at github at `https://github.com/RobinHankin/clifford`.

Notation follows Snygg [34]. We consider $V = \mathbb{R}^n$, an n-dimensional vector space spanned by basis vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$. Then an arbitrary vector in this space will be $a^1\mathbf{e}_1 + \cdots + a^n\mathbf{e}_n$. The associated Clifford algebra will be of dimension 2^n, spanned by formal products $\mathbf{e}_{i_1}\mathbf{e}_{i_2}\ldots\mathbf{e}_{i_r}$, $1 \leqslant i_1 < \cdots < i_r \leqslant n$. We write this as $\mathbf{e}_{i_1\ldots i_r}$ for brevity. The defining relations would be $\mathbf{e}_i\mathbf{e}_j = -\mathbf{e}_j\mathbf{e}_i$ for $i \neq j$ and

$$\mathbf{e}_i\mathbf{e}_i = \begin{cases} +1, & 1 \leqslant i \leqslant p \\ -1, & p < i \leqslant p+q \\ 0 & p+q < i \leqslant n. \end{cases} \tag{1.1}$$

The Clifford algebra $\mathcal{C}_{p,q}$ (other notations include $Cl(p,q)$) is then the algebra formed by $\mathbb{R}_{p,q}$ together with formal products of basis vectors. The value of n is suppressed in this notation in much the same way as the length of a numeric vector is ignorable in base R; but is implemented—by way of option `maxdim`—as a controllable option. One special case would be $p = q = 0$, implying that $\mathbf{e}_i\mathbf{e}_i = 0$ for $1 \leqslant i \leqslant n$; geometric products become wedge products.

2. Computational Implementation

In standard Clifford terminology, a *blade* is a product of one-vectors (that is, members of V), and a *basis blade* is a product of basis vectors.

The package represents basis blades using dynamic bitset objects from the `boost` library [28]. A bitset emulates an array of Boolean elements, but is optimized for space allocation and access/modification times. The set bits specify the basis blades present in a term; using bitsets allows products to

use fast Boolean operators. An object such as $\mathbf{e}_2\mathbf{e}_5\mathbf{e}_7$ [or $\mathbf{e}_{257}$] will be a bitset with bits 2, 5, and 7 set [note the off-by-one issue]. More detail on the implementation of products is given in Table 1. Dynamic objects are needed here as the number of bits in the set is specified at runtime. A `clifford` object is an element of a Clifford algebra; this is set of basis blades, each with a real coefficient. The `stl` map class [30] is used:

```
typedef boost::dynamic_bitset<> blade;
typedef map<blade, long double> clifford;
```

A "map" is a sorted associative container that contains key-value pairs with unique keys [30]. A `clifford` object thus maps dynamic bitsets (basis blades) to coefficients, here long doubles. The `STL map` class is used for efficiency: search, and insertion/deletion operations are consistently $\mathcal{O}(\log m)$, where m is the number of nonzero coefficients. For example, elements of any clifford algebra on $V = \mathbb{R}^n$ will have at most 2^n non-zero coefficients, searchable and manipulable in time $\mathcal{O}(n)$; further, any k-vector will have $\binom{n}{k} \lesssim 2^n/\sqrt{n}$ possible coefficients, so operations will again be $\mathcal{O}(n)$. The various products presented in Table 1 are more complicated but the geometric product of two k-vectors has complexity $\mathcal{O}(4^n)$ [4], and the outer product $\mathcal{O}(3^n)$ [5]. Similar techniques are used in the `spray` and `mvp` packages [16,18] which furnish functionality for multivariate polynomials.

3. Outline of the Computational Method

A Clifford object is the formal sum of terms; a term is a basis blade with an an associated coefficient; sums are managed by an STL map. Scalar multiplication is distributive; addition is performed elementwise via a standard container iterator.

The six "ordinary" products (that is, the first six products of Table 1) are implemented using a standardised system. Distributivity means we only need consider the product of basis blades.

To calculate the product of, say, $\mathbf{e}_{257}$ by $\mathbf{e}_{34}$, we first of all count the intersection of the set bits in the ranges $1 \leqslant i \leqslant p$, $p < i \leqslant p+q$, and $i > p+q$ to determine whether a sign flip is required or to set the product to zero. The different types of product have different rules for setting the product to zero (this is a little difficult to see at first glance; but taking the outer product as an example, $A \wedge B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{s+r}$, we can see that any common set bits in the basis blades of A and B will reduce the grade of their product, thus by definition having non-empty intersection zeros the product). We then write the product by juxtaposing the set bits to arrive at $\mathbf{e}_{25734}$; and then count the number of transpositions required to sort the set bits and arrive at $\mathbf{e}_{23457}$. The boost library performs all these operations quickly and efficiently. The coefficient of the product is just the product of the coefficient, modulo the sign.

4. The Package in Use

Suppose we specify that $\mathbf{e}_i\mathbf{e}_i = +1$, $i \geqslant 1$ (the default in the package) and want to work with Clifford object $1 + 2\mathbf{e}_1 + 3\mathbf{e}_2 + 4\mathbf{e}_2\mathbf{e}_3$. In R idiom this would be

```
> (x <- 1 + 2*e(1) + 3*e(2) + 4*e(2:3))
Element of a Clifford algebra, equal to
+ 1 + 2e_1 + 3e_2 + 4e_23
```

Here we have used function `e()` which takes an integer vector that specifies the term. Note that the `*` operator is "overloaded" [35]: that is, it is sensitive to the class of object on either side, and in this case `e(i)` is a Clifford object. Addition and subtraction work as expected:

```
> y <- 2*e(1) + 55*e(1:5)
> x-y
Element of a Clifford algebra, equal to
+ 1 + 3e_2 + 4e_23 - 55e_12345
```

In the above, see how the $\mathbf{e}_1$ term has vanished. We can multiply Clifford elements using natural R idiom:

```
> x*x
Element of a Clifford algebra, equal to
- 2 + 4e_1 + 6e_2 + 8e_23 + 16e_123
```

(Multiplication that Snygg denotes by juxtaposition is here indicated with a `*`). We can consider arbitrarily high dimensional data:

```
> (z <- as.1vector(1:7))
Element of a Clifford algebra, equal to
+ 1e_1 + 2e_2 + 3e_3 + 4e_4 + 5e_5 + 6e_6 + 7e_7
> z*x
Element of a Clifford algebra, equal to
+ 8 + 1e_1 - 10e_2 - 1e_12 + 11e_3 - 6e_13 - 9e_23 + 4e_123 +
4e_4 - 8e_14 - 12e_24 + 16e_234 + 5e_5 - 10e_15 - 15e_25 + 20e_235 +
6e_6 - 12e_16 - 18e_26 + 24e_236 + 7e_7 - 14e_17 - 21e_27 + 28e_237
```

In the above, we coerce a vector to a Clifford 1-vector. The package includes many functions to create elements of Clifford algebras, including `rcliff()`,

a routine for generating simple random Clifford objects with small integer coefficients:

```
> rcliff()
Element of a Clifford algebra, equal to
+ 5 - 5e_1 - 1e_4 + 4e_24 + 3e_5 + 6e_6 - 2e_26 - 4e_36 - 3e_1246
```

The defaults for `rcliff()` specify that the object is a sum of grade-4 terms but this can be altered:

```
> (x <- rcliff(d=7, g=5, include.fewer=TRUE))
Element of a Clifford algebra, equal to
+ 7 - 9e_3 - 7e_4 - 3e_1234 + 2e_6 + 1e_136 + 9e_1246 + 5e_456
+ 7e_1237 + 8e_13567
> elements(grades(x))
 [1] 0 1 1 4 1 3 4 3 4 5
```

Above, argument `include.fewer` directs `rcliff()` to return a Clifford object that includes terms with grades lower than the specified `g=5`; on the next line, `elements()` is needed to coerce the disord-compliant output of `grades()` to a normal R vector; full details are given in [15].

5. Pseudo-Euclidean Spaces

The signature of the metric may be altered. Starting with the Euclidean case we have:

```
> e1 <- e(1)
> e2 <- e(2)
> e1*e1
Element of a Clifford algebra, equal to
scalar ( 1 )
> e2*e2
Element of a Clifford algebra, equal to
scalar ( 1 )
```

However, if we wish to consider $p = 1$, $q = 1$ [thus $\mathbf{e}_1^2 = +1$, $\mathbf{e}_2^2 = -1$], package idiom is to use the `signature()` function:

```
> signature(1, 1)  # signature +-
> e1*e1 # as before, returns +1
Element of a Clifford algebra, equal to
scalar ( 1 )
> e2*e2 # should return -1
Element of a Clifford algebra, equal to
scalar ( -1 )
```

Suppose we wish to use a signature $+++-$, corresponding to the Minkowski metric in special relativity; this would be indicated in package idiom by `signature(3,1)`. Note that the clifford objects themselves do not store the signature; it is used only by the product operation `*`.

```
> x <- rcliff(d=4, g=3, include.fewer=TRUE)
> y <- rcliff(d=4, g=3, include.fewer=TRUE)
```

Then we may multiply these two clifford objects using either the default positive-definite inner product, or the Minkowski metric:

```
> x*y
Element of a Clifford algebra, equal to
+ 20 + 36e_1 - 92e_2 - 90e_12 + 37e_3 + 6e_13 + 92e_23 - 30e_123 -
10e_4 - 94e_24 - 22e_124 + 87e_34 + 29e_134 - 21e_234 - 20e_1234
> signature(3,1)  # switch to signature +++-
> x*y
Element of a Clifford algebra, equal to
+ 20 + 90e_1 - 56e_2 - 50e_12 - 38e_3 - 51e_13 + 106e_23 + 30e_123 +
2e_4 - 136e_24 - 40e_124 - 15e_34 + 15e_134 - 21e_234 - 20e_1234
```

In the above, see how the products are different using the two inner products.

6. Grassmann Algebra

A Grassmann algebra corresponds to a Clifford algebra with identically zero inner product. Package idiom is to use a zero signature:

```
> signature(0, 0)  # specify null inner product
> is.zero(e(5)^2)     # should be TRUE
[1] TRUE
```

This is a somewhat clunky way of reproducing the functionality of the `stokes` package [19]. To create example objects illustrating this feature of the package, we use `clifford()`, the *formal* creation method for Clifford objects:

```
> (x <- clifford(list(1:3, c(2,3,7)), coeffs=3:4))
Element of a Clifford algebra, equal to
+ 3e_123 + 4e_237
```

(above we see how `clifford()` takes a list of basis blades and a vector of coefficients). We may take the wedge product of `x` with another object `y`, created in the same way:

```
> y <- clifford(list(1:3, c(1,4,5), c(4,5,6)), coeffs=1:3)
> x
Element of a Clifford algebra, equal to
+ 9e_123456 - 8e_123457 - 12e_234567
```

then the `stokes` idiom for this would be:

```
> (x <- as.kform(rbind(1:3, c(2,3,7)), 3:4))
          val
 2 3 7  =   4
 1 2 3  =   3
> (y <- as.kform(rbind(1:3, c(1,4,5), 4:6), 1:3))
          val
 1 2 3  =   1
 1 4 5  =   2
 4 5 6  =   3
> x
                val
 1 2 3 4 5 6  =   9
 2 3 4 5 6 7  = -12
 1 2 3 4 5 7  =  -8
```

7. Positive-definite inner product

Function `signature()` takes an infinite argument to make the inner product positive-definite:

```
> signature(Inf)
```

(internally the package sets the signature to `.Machine$integer.max`, a near-infinite integer). With this, $\mathbf{e}_i\mathbf{e}_i = +1$ for any i. For example, if $i = 53$ we have:

```
> e(53)^2
Element of a Clifford algebra, equal to
scalar ( 1 )
```

8. Left and Right Contractions

Dorst [9] defines the left contraction $A \rfloor B$ and right contraction $A \lfloor B$ ([8] calls these left and right inner products) as follows:

$$A \rfloor B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{s-r} \tag{8.1}$$

$$A \lfloor B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{r-s} \tag{8.2}$$

(it is understood that $n < 0$ implies $\langle X \rangle_n = 0$ for any Clifford object). Package idiom for these would be `A%_|%B` and `A%|_%B` —or `lefttick(A,B)` and `righttick(A,B)`—respectively. Thus:

```
> (A <- rcliff())
Element of a Clifford algebra, equal to
+ 7 - 9e_1 + 7e_2 - 5e_23 + 9e_123 + 5e_5 - 7e_15 + 2e_1456
> (B <- rcliff())
Element of a Clifford algebra, equal to
+ 6 - 2e_2 + 6e_3 - 7e_123 + 7e_14 - 3e_5 + 2e_26 + 1e_46 +
9e_2346 - 9e_256
> A %_|%B
Element of a Clifford algebra, equal to
+ 76 - 35e_1 - 14e_2 + 42e_3 + 49e_13 + 63e_23 - 49e_123 - 63e_4 +
49e_14 - 21e_5 + 14e_6 + 59e_26 + 52e_46 + 63e_346 + 63e_2346 -
63e_56 - 63e_256
> A %_|%B
Element of a Clifford algebra, equal to
+ 76 - 33e_1 + 12e_2 + 54e_12 - 10e_3 + 18e_13 - 30e_23 + 54e_123 +
30e_5 - 40e_15 + 6e_146 - 14e_56 + 12e_1456
```

One thing to be wary of is the order of operations. Thus $\mathbf{e}_2 \rfloor \mathbf{e}_{12} = -\mathbf{e}_1$ (in a positive-definite space) but

```
> e(2) %_|% e(1)

Element of a Clifford algebra, equal to
the zero clifford element (0)
```

because this is parsed as $(\mathbf{e}_2 \rfloor \mathbf{e}_1)\mathbf{e}_2 = 0\mathbf{e}_2 = 0$. To evaluate this as intended we need to include brackets:

```
>e(2) %_|% (e(1)*e(2))

Element of a Clifford algebra, equal to
- 1e_1
```

although in this case it might be preferable to create the terms directly:

```
>e(2) %_|% e(1:2)

Element of a Clifford algebra, equal to
- 1e_1
```

8.1. Numerical Verification of Left and Right Inner Product Identities

Chisolm gives a number of identities for these products including

$$A\rfloor(B\lfloor C) = (A\rfloor B)\lfloor C \tag{8.3}$$
$$A\rfloor(B\rfloor C) = (A \wedge B)\rfloor C \tag{8.4}$$
$$A\lfloor(B \wedge C) = (A\lfloor B)\lfloor C \tag{8.5}$$

In package idiom:

```
> A <- rcliff();  B <- rcliff();  C <- rcliff()
> A %_|% (B %|_% C) == (A %_|% B) %|_% C

[1] TRUE

> A %_|% (B %_|% C) == (A %^% B) %_|% C

[1] TRUE

> A %|_% (B %^% C) == (A %|_% B) %|_% C

[1] TRUE
```

9. Higher Dimensional Spaces

Abłamowicz and Fauser [2] consider high-dimensional Clifford algebras and consider the following two elements of the 1024-dimensional Clifford algebra which we may treat as $\mathcal{C}_{7,3}$ spanned by $\mathbf{e}_1, \ldots, \mathbf{e}_{10}$ and perform a calculation which I reproduce below (although these authors exploited Bott periodicity, a feature not considered here).

Firstly we change the default print method slightly:

```
> options("basissep" = ",")
```

(this separates the subscripts of the basis vectors with a comma, which is useful for clarity if $n > 9$). We then define clifford elements x, y:

```
> (x <- clifford(list(1:3, c(1,5,7,8,10)), c(4,-10)) + 2)
Element of a Clifford algebra, equal to
+ 2 + 4e_1,2,3 - 10e_1,5,7,8,10
> (y <- clifford(list(c(1,2,3,7), c(1,5,6,8),
c(1,4,6,7)), c(4,1,-3)) - 1)
Element of a Clifford algebra, equal to
- 1 + 4e_1,2,3,7 - 3e_1,4,6,7 + 1e_1,5,6,8
```

Their geometric product is given in the package as

```
> signature(7)
> x*y
Element of a Clifford algebra, equal to
- 2 - 4e_1,2,3 - 16e_7 + 8e_1,2,3,7 - 6e_1,4,6,7 - 12e_2,3,4,6,7 +
2e_1,5,6,8 + 4e_2,3,5,6,8 - 40e_2,3,5,8,10 - 30e_4,5,6,8,10 +
10e_1,5,7,8,10
```

in agreement with [2], although the terms appear in a different order.

10. Case Study: Spheres in Conformal Geometric Algebra

Perwass [31] shows that geometric entities such as circles and spheres in $\mathbb{R}^3$ may be represented as points in a conformal geometric algebra. To use the package, we consider the three Euclidean basis vectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ together with two additional basis vectors $\mathbf{e}_+$ and $\mathbf{e}_-$ obeying $\mathbf{e}_+^2 = 1, \mathbf{e}_-^2 = -1, \mathbf{e}_+ \cdot \mathbf{e}_- = 0$. Hildenbrand [25] defines $\mathbf{e}_0 = (\mathbf{e}_- - \mathbf{e}_+)/2$ and $\mathbf{e}_\infty = \mathbf{e}_- + \mathbf{e}_+$, representing "the origin" and "the point at infinity" respectively. It is straightforward to implement these objects using the package:

```
> options("maxdim" = 5)
> signature(4, 1)
> eplus <- e(4)
> eminus <- e(5)
> e0 <- (eminus - eplus)/2
> einf <- eminus + eplus
```

Above we use package idiom to specify $C\ell(4,1)$ with Euclidean basis vectors `e(1)`, `e(2)`, `e(3)`, and define conformal objects e_+, e_-, e_0, e_∞ in terms of `e(4)` and `e(5)`. Perwass shows that a point $\mathbf{x} = (x_1, x_2, x_3)$ may be projectively extended to $P \in C\ell(4,1)$ by the expression $P = \mathbf{x} + (\mathbf{x}\cdot\mathbf{x})\mathbf{e}_\infty/2 + \mathbf{e}_0$. Package idiom for this is straightforward:

```
> point <- function(x){as.1vector(x) + sum(x^2)*einf/2 + e0}
> point(c(5, -4, 7))

Element of a Clifford algebra, equal to
+ 5e_1 - 4e_2 + 7e_3 + 44.5e_4 + 45.5e_5
```

Perwass goes on to show that a sphere S may be specified in terms of four (conformal) points that lie on it:

$$S = P_1 \wedge P_2 \wedge P_3 \wedge P_4 / I \tag{10.1}$$

where the P_i are points that lie on the sphere and I is the unit pseudoscalar. Here I demonstrate the package's capabilities by finding the unique sphere that passes through four points. This is possible, but difficult, using standard Euclidean methods; but is a one-liner in conformal geometry. Below, I specify the center and radius of a sphere in $\mathbb{R}^3$, choose four points on the sphere, and numerically reconstruct the center and radius from the resulting conformal sphere:

```
> center <- c(3, -5, 6)
> radius <- 3
> point_on_sphere  <- function(center, radius, vec){
  point(center + radius*vec/sqrt(sum(vec^2)))
}

> p1 <- point_on_sphere(center, radius, c(1, 4, 2))
> p2 <- point_on_sphere(center, radius, c(6,-5, 3))
> p3 <- point_on_sphere(center, radius, c(0, 4, 0))
> p4 <- point_on_sphere(center, radius, c(1, 0, 0))

> S <- p1 ^ p2 ^ p3 ^ p4 / pseudoscalar()
```

Then we extract the radius and center and compare with the known values of 3 and $(3, -5, 6)$ respectively. The radius r is given by $r^2 = \left(\frac{S}{-S\cdot\mathbf{e}_\infty}\right)^2$ and the center by scaling the sandwich product $S\mathbf{e}_\infty S$.

```
> sqrt(drop((S / drop(S
[1]  3
> S_sandwich <- S * einf * S
> scaling_factor <- -zap(S_sandwich
> getcoeffs(zap(S_sandwich / scaling_factor), 1:3)
[1]  3 -5 6
```

Above, we use function `zap()` which cancels small coefficients which appear due to numerical error; apart from that, we see that the radius and center of the original sphere are reconstituted accurately and efficiently, using natural R idiom.

11. Conclusions and Further Work

The `clifford` package furnishes a consistent and documented suite of reasonably efficient `R`-centric functionality. The package is available on CRAN at `https://CRAN.R-project.org/package=clifford` and development versions are hosted at github at `https://github.com/RobinHankin/clifford`. Further work might include implementation of spinor algebra (via existing R-centric symbolic matrix functionality [12,21]); representation of quadric surfaces, as in [6]; or perhaps a more complete implementation of Clifford inverses as introduced in [27].

TABLE 1. The Clifford products implemented in the package

Geometric product	`A*B = geoprod(A,B)`	$AB = \sum_{r,s} \langle A \rangle_r \langle B \rangle_s$
Inner product	`A %.% B = cliffdotprod(A,B)`	$A \cdot B = \sum_{\substack{r\neq 0 \\ s\neq 0}} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{\lvert s-r \rvert}$
Outer product	`A %^% B = wedge(A,B)`	$A \wedge B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{s+r}$
Fat dot product	`A %o% B = fatdot(A,B)`	$A \bullet B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{\lvert s-r \rvert}$
Left contraction	`A %_|% B = lefttick(A,B)`	$A \rfloor B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{s-r}$
Right contraction	`A %|_% B = righttick(A,B)`	$A \lfloor B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_{r-s}$
Cross product	`A %X% B = cross(A,B)`	$A \times B = \frac{1}{2}(AB - BA)$
Scalar product	`A %star% B = star(A,B)`	$A * B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle_0$
Euclidean product	`A %euc% B = eucprod(A,B)`	$A \star B = A * B^\dagger$

Appendix A. Products and Conjugations Implemented in the Package

The package implements a number of products and involutions. The products are given in Table 1. An involution is a function that is its own inverse, or equivalently $f(f(x)) = x$. There are several important involutions on Clifford objects; these commute past the grade operator with $f(\langle A\rangle_r) = \langle f(A)\rangle_r$ and are linear: $f(\alpha A + \beta B) = \alpha f(A) + \beta f(B)$. The dual is documented here for convenience, even though it is not an involution (applying the dual *four* times is the identity).

- The **reverse** $A^\sim$ is given by `rev()` (both Perwass and Dorst use a tilde, as in $\tilde{A}$ or $A^\sim$. However, both Hestenes and Chisolm use a dagger, as in $A^\dagger$. Here, I use Perwass's notation). The reverse of a term written as a product of basis vectors is simply the product of the same basis vectors but written in reverse order. This changes the sign of the term if the number of basis vectors is 2 or 3 (modulo 4). Thus, for example, $(e_1e_2e_3)^\sim = e_3e_2e_1 = -e_1e_2e_3$ and $(e_1e_2e_3e_4)^\sim = e_4e_3e_2e_1 = +e_1e_2e_3e_4$. Formally, if $X = e_{i_1}\dots e_{i_k}$, then $\tilde{X} = e_{i_k}\dots e_{i_1}$.
$$\langle A^\sim\rangle_r = \widetilde{\langle A\rangle_r} = (-1)^{r(r-1)/2}\langle A\rangle_r$$
Perwass shows that $\langle AB\rangle_r = (-1)^{r(r-1)/2}\left\langle \tilde{B}\tilde{A}\right\rangle_r$.
- The **Conjugate** $A^\dagger$ is given by `Conj()` (we use Perwass's notation, def 2.9 p59). This depends on the signature of the Clifford algebra; see `grade.Rd` for notation. Given a basis blade $e_{\mathbb{A}}$ with $\mathbb{A} \subseteq \{1,\dots,p+q\}$, then we have $e_{\mathbb{A}}^\dagger = (-1)^m e_{\mathbb{A}}{}^\sim$, where $m = \mathrm{gr}_-(\mathbb{A})$. Alternatively, we might say
$$(\langle A\rangle_r)^\dagger = (-1)^m(-1)^{r(r-1)/2}\langle A\rangle_r$$
where $m = \mathrm{gr}_-(\langle A\rangle_r)$ [NB I have changed Perwass's notation].
- The **main (grade) involution** or **grade involution** $\widehat{A}$ is given by `gradeinv()`. This changes the sign of any term with odd grade:
$$\widehat{\langle A\rangle_r} = (-1)^r\langle A\rangle_r$$
(I don't see this in Perwass or Hestenes; notation follows Hitzer and Sangwine). It is a special case of grade negation.
- The bf grade r-negation $A_{\overline{r}}$ is given by `neg()`. This changes the sign of the grade r component of A. It is formally defined as $A - 2\langle A\rangle_r$ but function `neg()` uses a more efficient method. It is possible to negate all terms with specified grades, so for example we might have $\langle A\rangle_{\overline{\{1,2,5\}}} = A - 2(\langle A\rangle_1 + \langle A\rangle_2 + \langle A\rangle_5)$ and the R idiom would be `neg(A,c(1,2,5))`. Note that Hestenes uses "$A_{\overline{r}}$" to mean the same as $\langle A\rangle_r$.
- The **Clifford conjugate** $\overline{A}$ is given by `cliffconj()`. It is distinct from conjugation $A^\dagger$, and is defined in Hitzer and Sangwine as
$$\overline{\langle A\rangle_r} = (-1)^{r(r+1)/2}\langle A\rangle_r\,.$$
- The **dual** C^* of a clifford object C is given by `dual(C,n)`; argument `n` is the dimension of the underlying vector space. Perwass gives $C^* = CI^{-1}$

where $I = e_1 e_2 \dots e_n$ is the unit pseudoscalar [note that Hestenes uses I to mean something different]. The dual is sensitive to the signature of the Clifford algebra *and* the dimension of the underlying vector space.

Appendix B. Note on disordR Discipline

Clifford objects are considered to be the sum of a finite number of blades, each multiplied by a coefficient. As discussed above, the mapping itself is implemented using the STL map class [30]; one reason why the this is fast is that the order in which the key-value pairs are stored is undefined: the compiler may store them in the order which it regards as most propitious. The order of storage is immaterial algebraically, as addition is commutative and associative. In the clifford package, the map is from blades to their coefficient, a real number.

However, one issue arising from the use of the map class in computational algebra is that the order of coefficients is implementation-specific. This can be problematic if one attempts to access coefficients using standard vector notation, for the result is not well-defined. An example might be Clifford object $3 + 4\mathbf{e}_1 - 5\mathbf{e}_{123}$ and its representation in the following R session:

```
> p <- 3 + 4*e(1) - 5*e(1:3)
> p
Element of a Clifford algebra, equal to
+ 3 + 4e_1 - 5e_123
```

We observe that p is algebraically equivalent to $4\mathbf{e}_1 - 5\mathbf{e}_{123} + 3$ [the terms are written in a different order] and the internal map equivalent leverages this ambiguity by storing the key-value pairs in whatever order the compiler finds most propitious. We might wish to extract the coefficients:

```
> coeffs(p)
  A disord object with hash cb6600d951d6a0e1a666fb9217a896b7706
ee3b5 and elements
[1]  3  4 -5
(in some order)
```

We see that the coefficients are $3, 4, -5$, but the returned value—an object of class disord—declines to specify an order for these coefficients. This is because the order is implementation-specific, and disordR discipline will not "guess" which order is actually employed. For example, if we try to extract the first element, an error is returned:

```
> p[1]
Error in .local(x, i, j = j, ..., drop) :
   if using a regular index to extract, must extract each
   element once and once only (or none of them)
```

We see that the package refuses to declare which entry is the first, and returns an error message. Even though the order of coefficients is not well defined, various statistics can be obtained:

```
> sum(coeffs(p))
[1]  2
```

Here we observe that the sum of the coefficients *is* well defined, whatever order is used for the summation: addition is commutative and associative. The `disordR` package manages these issues, essentially ensuring that meaningful questions with a well-defined answer (such as the sum of coefficients) are executed transparently and efficiently, while ill-posed questions with implementation-specific results (such as the "first" element) return an error. The map class is used in a wide range of algebraic contexts and `disordR` discipline is enforced in a number of packages; examples would include [13,20,23]. Full details and further motivating examples are given in [15].

References

[1] Ablamowicz, R., Fauser, B.: Mathematics of clifford–a maple package for clifford and grassmann algebras. Adv. Appl. Clifford Algebras **15**(2), 157–181 (2005)

[2] Abłamowicz, R., Fauser, B.: Symbolic computations in higher dimensional clifford algebras, (2012)

[3] Aragon-Camarasa, G., Aragon-Gonzalez, G., Aragon, J. L., Rodriguez-Andrade, M. A.: Clifford algebra with mathematica, (2018)

[4] Breuils, S., Nozick, V., Fuchs, L.: A geometric algebra implementation using binary tree. Adv. Appl. Clifford Algebras **27**, 2133–2151 (2017)

[5] Breuils, S., Nozick, V., Sugimoto, A.: Computational aspects of geometric algebra products of two homogeneous multivectors. Advances in Applied Clifford Algebras, 33(4), (2023)

[6] Breuils, S., Nozick, V., Sugimoto, A., Hitzer, E.: Quadric conformal geometric algebra of $\mathbb{R}^{9,6}$. Adv. Appl. Clifford Algebras **28**(35), 1–16 (2018)

[7] Burns, L.: Awesome geometric algebra. Github repo, https://github.com/ga/awesome-geometric-algebra, (2020)

[8] Eric Chisolm. Geometric algebra, (2012)

[9] Dorst, L.: Applications of geometric algebra in computer science and engineering, chapter 2, pages 35–46. Birkhäuser, (2002)

[10] Fontijne, D.: Gaigen 2: a geometric algebra implementation generator. In Proceedings of the 5th international conference on Generative programming and component engineering, pages 141–150, (2006)

[11] Franchini, S., Gentile, A., Sorbello, F., Vassallo, G., Vitabile, S.: An embedded, FPGA-based computer graphics coprocessor with native geometric algebra support. Integr. VLSI J. **42**(3), 346–355 (2008)

[12] Hankin, R.: Normed division algebras with R: introducing the onion package. R News **6**(2), 49–52 (2006)

[13] Hankin, R.: Partial rank data with the hyper2 package: likelihood functions for generalized Bradley-Terry models. The R Journal **9**(2), 429–439 (2017)

[14] Hankin, R. K. S.: Introducing the permutations R package. SoftwareX, 11, (2020)

[15] Hankin, R. K. S.: Disordered vectors in R: introducing the `disordR` package. arXiv:2210.03856, (2022)

[16] Hankin,R. K. S.: Fast multivariate polynomials in R: the `mvp` package. arXiv:2210.15991, (2022)

[17] Hankin, R. K. S.: Quantum algebra in R: the weyl package. arXiv:2212.09230, (2022)

[18] Hankin, R. K. S.: Sparse arrays in R: the `spray` package. arXiv:2210.03856, (2022)

[19] Hankin, R. K. S.: Stokes's theorem in R. arXiv:2210.17008, (2022)

[20] Hankin, R. K. S.: The free abelian group in R: the `frab` package. arXiv:2307.13184, (2023)

[21] Hankin, R. K. S.: Jordan algebra in R. arXiv:2303.06062, (2023)

[22] Hankin, R. K. S.: Visually pleasing knot projections. Journal of Mathematics and the Arts, (2023)

[23] Hankin, R. K. S.: Antiassociative algebra in R: introducing the `evitaicossa` package. arXiv:2412.16161, (2024)

[24] Hestenes, D., Sobczyk,G.: Clifford algebra to geometric calculus. Kluwer, (1987)

[25] Hildenbrand,D.: Foundations of geometric algebra computing. Springer, (2013)

[26] Hildenbrand, D., Pitt, J., Koch,A.: Geometric Algebra Computing, chapter Gaalop - high performance parallel computing based on conformal geometric algebra. Springer, (2010)

[27] Hitzer, E., Sangwine, S.: Multivector and multivector matrix inverses in real Clifford algebras. Appl. Math. Comput. **311**, 375–389 (2017)

[28] Karlsson, B.: Beyond the C++ Standard Library: An Introduction to Boost. Addison-Wesley, (2005)

[29] Meurer, A., et al.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, e103 (2017)

[30] Musser, David R., Derge, Gillmer J., Saini, A.: STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. Addison-Wesley Professional, 3rd edition, (2009)

[31] Perwass, C.: Geometric algebra with applications in engineering. Springer, (2009)

[32] R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, (2022)

[33] Scrimshaw,T., Karn, T. K.: Clifford algebras, 2022. Subsection of [36]

[34] Snygg, J.: A new approach to differential geometry using Clifford's geometric algebra. Birkhäuser, (2010)

[35] Stroustrup, B.: The C++ programming language. Addison-Wesley, fourth edition, (2013)

[36] The Sage Developers. SageMath, the Sage Mathematics Software System (Version 8.6), (2019)

Robin K. S. Hankin
University of Stirling
Stirling FK9 4LH
UK
e-mail: hankin.robin@gmail.com

Received: July 17, 2024.
Accepted: July 17, 2025.