Package ‘clifford’

March 8, 2020

Type Package
Title Arbitrary Dimensional Clifford Algebras
Version 1.0-2
Maintainer Robin K. S. Hankin <hankin.robin@gmail.com>
Description A suite of routines for Clifford algebras, using the 'Map' class of the Standard Template Library. Canonical reference: Hestenes (1987, ISBN 90-277-1673-0, "Clifford algebra to geometric calculus"). Special cases including Lorentz transforms, quaternion multiplication, and Grassman algebra, are discussed. Conformal geometric algebra theory is implemented.
License GPL (>= 2)
LazyData yes
Suggests knitr,testthat,onion,lorentz
VignetteBuilder knitr
Imports Rcpp (>= 0.12.5)
LinkingTo Rcpp,BH
SystemRequirements C++11
URL https://github.com/RobinHankin/clifford.git
BugReports https://github.com/RobinHankin/clifford/issues
NeedsCompilation yes
Author Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)
Repository CRAN
Date/Publication 2020-03-08 20:20:02 UTC

R topics documented:

clifford-package .......................................................... 2
allcliff ................................................................. 4
antivector ............................................................... 5
A suite of routines for Clifford algebras, using the 'Map' class of the Standard Template Library. Canonical reference: Hestenes (1987, ISBN 90-277-1673-0, "Clifford algebra to geometric calculus"). Special cases including Lorentz transforms, quaternion multiplication, and Grassman algebra, are discussed. Conformal geometric algebra theory is implemented.

Details

The DESCRIPTION file:

Package: clifford
Type: Package
Title: Arbitrary Dimensional Clifford Algebras
Version: 1.0-2
Authors@R: person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gmail.com")
Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>
Description: A suite of routines for Clifford algebras, using the 'Map' class of the Standard Template Library. Canonical reference: Hestenes (1987, ISBN 90-277-1673-0, "Clifford algebra to geometric calculus"). Special cases including Lorentz transforms, quaternion multiplication, and Grassman algebra, are discussed. Conformal geometric algebra theory is implemented.
License: GPL (>= 2)
LazyData: yes
Suggests: knitr,testthat,onion,lorentz
VignetteBuilder: knitr
Imports: Rcpp (>= 0.12.5)
LinkingTo: Rcpp,BH
SystemRequirements: C++11
URL: https://github.com/RobinHankin/clifford.git
BugReports: https://github.com/RobinHankin/clifford/issues
Author: Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

Conj
Ops.clifford
[.clifford
allcliff
antivector
as.vector
c_identity
clifford
clifford-package
drop
dual
even
getcoeffs
grade
homog
magnitude
neg
numeric_to_clifford
print.clifford
quaternion
rcliff
rev
signature
summary.clifford
term
zap
zero

Author(s)
NA

Maintainer: Robin K. S. Hankin <hankin.robin@gmail.com>
References


See Also

clifford

Examples

```r
as.1vector(1:4)

as.1vector(1:4) * rcliff()

# Following from Ablamowicz and Fauser (see vignette):
x <- clifford(list(1:3,c(1,5,7,8,10)),c(4,-10)) + 2
y <- clifford(list(c(1,2,3,7),c(1,5,6,8),c(1,4,6,7)),c(4,1,-3)) - 1
x*y # signature irrelevant
```

---

**allcliff**

*Clifford object containing all possible terms*

Description

The Clifford algebra on basis vectors $e_1, e_2, ..., e_n$ has $2^n$ independent multivectors. Function `allcliff()` generates a clifford object with a nonzero coefficient for each multivector.

Usage

```r
allcliff(n)
```

Arguments

- `n` Integer

Author(s)

Robin K. S. Hankin
antivector

Examples

allcliff(6)

a <- allcliff(5)
a[] <- rcliff()*100

antivector  Antivectors or pseudovectors

Description

Antivectors or pseudovectors

Usage

antivector(v, n = length(v))
is.antivector(C, include.pseudoscalar=FALSE)

Arguments

v       Numeric vector
n       Integer
C       Clifford object
include.pseudoscalar
        Boolean: should the pseudoscalar be considered an antivector?

Details

An antivector is an $n$-dimensional Clifford object of all of whose terms are of grade $n - 1$.
The pseudoscalar is a peculiar edge case. Consider:

A <- clifford(list(c(1,2,3)))
B <- A + clifford(list(c(1,2,4)))

> is.antivector(A)
[1] FALSE
> is.antivector(B)
[1] TRUE
> is.antivector(A,include.pseudoscalar=TRUE)
[1] TRUE
> is.antivector(B,include.pseudoscalar=TRUE)
[1] TRUE

One could argue that A should be an antivector as it is a term in B, which is definitely an antivector.
Use include.pseudoscalar=TRUE to ensure consistency in this case.
Note
An antivector is always a blade.

Author(s)
Robin K. S. Hankin

References

Examples
antivector(1:5)

Description
Given a clifford object with all terms of grade 1, return the corresponding numeric vector

Usage
## S3 method for class 'clifford'
as.vector(x,mode = "any")

Arguments
x Object of class clifford
mode ignored

Note
The awkward R idiom of this function is because the terms may be stored in any order; see the examples

Author(s)
Robin K. S. Hankin

See Also
numeric_to_clifford
Examples

x <- clifford(list(6,2,9),1:3)
as.vector(x)

as.1vector(as.vector(x)) == x  # should be TRUE

clifford

Create, coerce, and test for clifford objects

Description

A clifford object is a member of a Clifford algebra. These objects may be added and multiplied, and have various applications in physics and mathematics.

Usage

clifford(terms, coeffs=1)
is.ok.clifford(terms, coeffs)
as.clifford(x)
is.clifford(x)
nbits(x)
nterms(x)

Arguments

terms A list of integer vectors with strictly increasing entries corresponding to the basis vectors of the underlying vector space
coeffs Numeric vector of coefficients
x Object of class clifford

Details

- Function clifford() is the formal creation mechanism for clifford objects
- Function as.clifford() is much more user-friendly and attempts to coerce a range of input arguments to clifford form
- Function nbits() returns the number of bits required in the low-level C routines to store the terms (this is the largest entry in the list of terms)
- Function nterms() returns the number of terms in the expression
- Function is.ok.clifford() is a helper function that checks for consistency of its arguments
- Function is.term() returns TRUE if all terms of its argument have the same grade

Author(s)

Robin K. S. Hankin
Conj

Conjugate of a Clifford object

Description

The “conjugate” of a Clifford object is defined by Perwass in definition 2.9, p59.

Usage

## S3 method for class 'clifford'
Conj(z)
Arguments

z  Clifford object

Details

Perwass uses a dagger to indicate Conjugates, as in $A^\dagger$. If

$$A_{\langle k \rangle} = \bigwedge_{i=1}^{k} a_i$$

Then

$$A_{\langle k \rangle}^\dagger = (a_1 \wedge \ldots \wedge a_k)^\dagger = a_k^\dagger \wedge \ldots \wedge a_1^\dagger = \bigwedge_{i=1}^{k} a_i^\dagger$$

He gives the following theorem (3.58, p70):
Given blades $A_{\langle k \rangle}, B_{\langle l \rangle}$, then

$$(A_{\langle k \rangle} \wedge B_{\langle l \rangle})^\dagger = B_{\langle l \rangle}^\dagger \wedge A_{\langle k \rangle}^\dagger$$

and

$$(A_{\langle k \rangle} B_{\langle l \rangle})^\dagger = B_{\langle l \rangle}^\dagger A_{\langle k \rangle}^\dagger$$

See examples for package idiom.

Author(s)

Robin K. S. Hankin

References


See Also

g_grade, rev

Examples

signature(2)
A <- rblade(g=3)
B <- rblade(g=4)
Conj(A %^% B) - Conj(B) %^% Conj(A)  # should be small
\begin{verbatim}
Conj(A * B) - Conj(B) * Conj(A)    # should be small

x1 <- rblade(d=9,g=2)
x2 <- rblade(d=9,g=2)
x3 <- rblade(d=9,g=2)
x4 <- rblade(d=9,g=2)

LHS <- Conj(x1 %*% x2 %*% x3 %*% x4)
RHS <- Conj(x4) %*% Conj(x3) %*% Conj(x2) %*% Conj(x1)
Mod(LHS - RHS) # should be small

LHS <- Conj(x1 * x2 * x3 * x4)
RHS <- Conj(x4) * Conj(x3) * Conj(x2) * Conj(x1)
Mod(LHS - RHS) # should be small

signature(0)
\end{verbatim}

---

**drop**  
*Drop redundant information*

**Description**  
Coerce constant Clifford objects to numeric

**Usage**  
drop(C)

**Arguments**  
C          Clifford object

**Details**  
If its argument is a constant clifford object, coerce to numeric.

**Author(s)**  
Robin K. S. Hankin

**See Also**  
grade, getcoeffs


**Examples**

\[
\text{drop(as.clifford(5))}
\]

\[
\text{const(rcliff())}
\]

\[
\text{const(rcliff(),drop=FALSE)}
\]

---

**dual**  
*The dual of a clifford object*

**Description**

The dual of a clifford object \( C \), written \( C^* \)

**Usage**

\[
dual(C, n)
\]

**Arguments**

- \( C \quad \text{Clifford object} \)
- \( n \quad \text{Dimensionality of underlying vector space} \)

**Details**

The dual of clifford object \( C \) is \( CI^{-1} \) where \( I \) is the pseudoscalar.

The dual is sensitive to the signature. Note that applying the dual operation four times successively will return

**Author(s)**

Robin K. S. Hankin

**References**


**See Also**

- **pseudoscalar**

**Examples**

\[
a <- \text{rcliff()}
dual(dual(dual(dual(a,8),8),8),8) == a \quad \text{# should be TRUE}
\]
Even and odd clifford objects

Description

A clifford object is even if every term has even grade, and odd if every term has odd grade.

Functions `is.even()` and `is.odd()` test a clifford object for evenness or oddness.

Functions `evenpart()` and `oddpart()` extract the even or odd terms from a clifford object, and we write \( A_+ \) and \( A_- \) respectively; we have \( A = A_+ + A_- \).

Usage

```r
is.even(C)
is.odd(C)
evenpart(C)
oddpart(C)
```

Arguments

- `C`  
  Clifford object

Author(s)

Robin K. S. Hankin

See Also

`grade`

Examples

```r
A <- rcliff()
A == evenpart(A) + oddpart(A)  # should be true
```

Extract or Replace Parts of a clifford

Description

Extract or replace subsets of cliffords.
Usage

```r
## S3 method for class 'clifford'
C[index, ...]
## S3 replacement method for class 'clifford'
C[index, ...] <- value
coeffs(x)
coeffs(x) <- value
```

Arguments

- `C, x` A clifford object
- `index` elements to extract or replace
- `value` replacement value
- `...` Further arguments

Details

Extraction and replacement methods. The extraction method uses `getcoeffs()` and the replacement method uses low-level helper function `c_overwrite()`.

In the extraction function `a[index]`, if `index` is a list, further arguments are ignored. If not, the dots are used.

Replacement methods using list-valued `index`, as in `A[i] <- value` uses an ugly hack if `value` is zero.

Idiom such as `a[] <- b` follows the spray package. If `b` is a length-one scalar, then `coeffs(a) <- b` has the same effect as `a[] <- b`.

Functions `terms()` [see `term.Rd`] and `coeffs()` are the extraction methods. These are unordered vectors but the ordering is consistent between them (an extended discussion of this phenomenon is presented in the `mvp` package).

Function `coeffs<-()` (idiom `coeffs(a) <- b`) sets all coefficients of `a` to `b`. This has the same effect as `a[] <- b`.

See Also

- `Ops.clifford`, `clifford`, `term`

Examples

```r
A <- clifford(list(1,1:2,1:3),1:3)
B <- clifford(list(1:2,1:6),c(44,45))

A[1,c(1,3,4)]
A[] <- B
```
getcoeffs

Get coefficients of a Clifford object

Description

Access specific coefficients of a Clifford object using a list of terms.

Usage

getcoeffs(C, B)
const(C, drop = TRUE)
## S3 replacement method for class 'clifford'
const(x) <- value

Arguments

C, x Clifford object
B List of terms
value Replacement value
drop Boolean, with default TRUE meaning to return the constant coerced to numeric, and FALSE meaning to return a (constant) Clifford object

Details

Extractor method for specific terms. Function const() returns the constant element of a Clifford object. Note that const(C) returns the same as grade(C, 0), but is faster.

The slightly awkward R idiom in const<-() is to ensure numerical accuracy; see examples.

Author(s)

Robin K. S. Hankin

See Also

clifford

Examples

X <- clifford(list(1,1:2,1:3,3:5),6:9)
getcoeffs(X,1:2)
X <- X + 1e300
const(X) # should be 1e300
grade

The grade of a clifford object

Description

The grade of a term is the number of basis vectors in it.

Usage

grade(C, n, drop=TRUE)
grades(x)
gradesplus(x)
gradesminus(x)

Arguments

C, x Clifford object
n Integer vector specifying grades to extract
drop Boolean, with default TRUE meaning to coerce a constant Clifford object to numeric, and FALSE meaning not to

Details

A term is a single expression in a Clifford object. It has a coefficient and is described by the basis vectors it comprises. Thus 4e_{234} is a term but 1e3 + 2e5 is not. The grade of a term is the number of basis vectors in it. Thus the grade of e_1 is 1, and the grade of e_{125} = e_1 e_2 e_5 is 3. The grade operator ⟨·⟩_r is used to extract terms of a particular grade, with

\[ A = ⟨A⟩_0 + ⟨A⟩_1 + ⟨A⟩_2 + \cdots = \sum_r ⟨A⟩_r \]

for any Clifford object A. Thus ⟨A⟩_r is said to be homogenous of grade r. Hestenes sometimes writes subscripts that specify grades using an overbar as in ⟨A⟩_{\overline{r}}. It is conventional to denote the zero-grade object ⟨A⟩_0 as simply ⟨A⟩.

We have

\[ ⟨A + B⟩_r = ⟨A⟩_r \quad (λA)_r = λ ⟨A⟩_r \quad ⟨⟨A⟩⟩_s = ⟨A⟩_r δ_{rs}. \]

Function grades() returns an (unordered) vector specifying the grades of the constituent terms. Function gradesplus() returns the same but counting only basis vectors that square to +1, and gradesminus() counts only basis vectors that square to −1. These defined by Perwass, page 57. Function grade(C, n) returns a clifford object with just the elements of grade g, where g %in% n.

Function c_grade() is a helper function that is documented at Ops.clifford.Rd.
Note
In the C code, “blade” has a slightly different meaning, referring to the vectors without the associated coefficient.

Author(s)
Robin K. S. Hankin

References

Examples

```r
a <- clifford(sapply(seq_len(7),seq_len),seq_len(7))
grades(a)
grade(a,5)
```

---

**homog**

*Homogenous Clifford objects*

**Description**
A clifford object is homogenous if all its terms are the same grade. A scalar (including the zero clifford object) is considered to be homogenous. This ensures that `is.homog(grade(C,n))` always returns `TRUE`.

**Usage**

```r
is.homog(C)
```

**Arguments**

- `C` Object of class clifford

**Details**
Homogenous clifford objects have a multiplicative inverse.

**Author(s)**
Robin K. S. Hankin

**Examples**

```r
is.homog(rcliff())
is.homog(rcliff(include.fewer=FALSE))
```
Low-level helper functions for `clifford` objects

Description

Helper functions for `clifford` objects, written in C using the STL map class.

Usage

```c
 c_identity(L, p, m)
c_grade(L, c, m, n)
c_add(L1, c1, L2, c2, m)
c_multiply(L1, c1, L2, c2, m, sig)
c_power(L, c, m, p, sig)
c_equal(L1, c1, L2, c2, m)
c_overwrite(L1, c1, L2, c2, m)
```

Arguments

- `L, L1, L2`: Lists of terms
- `c1, c2, c`: Numeric vectors of coefficients
- `m`: Maximum entry of terms
- `n`: Grade to extract
- `p`: Integer power
- `sig`: Positive integer representing number of +1 on main diagonal of quadratic form

Details

The functions documented here are low-level helper functions that wrap the C code. They are called by functions like `clifford_plus_clifford()`, which are themselves called by the binary operators documented at `Ops.clifford.Rd`.

Function `clifford_inverse()` is problematic as nonnull blades always have an inverse; but function `is.blade()` is not yet implemented. Blades (including null blades) have a pseudoinverse, but this is not implemented yet either.

Value

The high-level functions documented here return an object of `clifford`. But don’t use the low-level functions.

Author(s)

Robin K. S. Hankin

See Also

`Ops.clifford`
magnitude

**Magnitude of a clifford object**

### Description
Following Perwass, the magnitude of a multivector is defined as

\[ ||A|| = \sqrt{A \ast A} \]

Where \( A \ast A \) denotes the Euclidean scalar product `eucprod()`. Recall that the Euclidean scalar product is never negative.

### Usage

```r
## S3 method for class 'clifford'
Mod(z)
```

### Arguments

- `z` Clifford objects

### Note
If you want the square, \( ||A||^2 \) and not \( ||A|| \), it is faster and more accurate to use `eucprod(A)`, because this avoids a needless square root.

There is a nice example of scalar product at `rcliff.Rd`.

### Author(s)
Robin K. S. Hankin

### See Also

- `Ops.clifford`, `Conj`, `rcliff`

### Examples

```r
Mod(rcliff())
```

# Perwass, p68, asserts that if A is a k-blade, then (in his notation)
# AA == A*A.

# In package idiom, A*A == A %star% A:
A <- rcliff()
Mod(A*A - A %star% A) # meh

A <- rblade()
Mod(A*A - A %star% A) # should be small

---

**neg**  

**Grade negation**

**Description**

The grade \( r \) negation operation applied to Clifford multivector \( A \) changes the sign of the grade \( r \) component of \( A \). It is formally defined as \( A - 2 \langle A \rangle_r \).

**Usage**

\[ \text{neg}(C, n) \]

**Arguments**

- \( C \)  
  Clifford object
- \( n \)  
  Integer vector indicating grades to negate

**Details**

The function is algebraically equivalent to \( \text{function}(C, n) \{ C - 2 \cdot \text{grade}(C, n) \} \) but uses faster and more efficient idiom.

**Author(s)**

Robin K. S. Hankin

**References**


**See Also**

Conj

**Examples**

\[ \text{A} <- \text{rcliff()} \]
\[ \text{neg}(A,1:2) == \text{A-grade}(A,1:2) \] # should be TRUE
numeric_to_clifford  Coercion from numeric to Clifford form

Description

Given a numeric value or vector, return a Clifford algebra element

Usage

numeric_to_clifford(x)
as.1vector(x)
is.1vector(x)
scalar(x=1)
as.scalar(x=1)
is.scalar(C)
basis(n,x=1)
e(n,x=1)
pseudoscalar(n,x=1)
as.pseudoscalar(n,x=1)
is.pseudoscalar(C)

Arguments

x  Numeric vector
n  Integer specifying dimensionality of underlying vector space
C  Object possibly of class Clifford

Details

Function as.scalar() takes a length-one numeric vector and returns a Clifford scalar of that value (to extract the scalar component of a multivector, use const()).

Function as.1vector() takes a numeric vector and returns the linear sum of length-one blades with coefficients given by x; function is.1vector() returns TRUE if every term is of grade 1.

Function pseudoscalar(n) returns a pseudoscalar of dimensionality n and function is.pseudoscalar() checks for a Clifford object being a pseudoscalar.

Function numeric_to_vector() dispatches to either as.scalar() for length-one vectors or as.1vector() if the length is greater than one.

Function basis() returns a clifford element comprising of single blade with grade 1; function e() is a synonym.

Author(s)

Robin K. S. Hankin
See Also

gcoeffs

Examples

as.scalar(6)
as.1vector(1:8)

Reduce(`+`, sapply(seq_len(7), function(n) { e(seq_len(n)) }, simplify = FALSE))
pseudoscalar(6)
pseudoscalar(7, 5) == 5*pseudoscalar(7)  # should be true

Description

Allows arithmetic operators to be used for multivariate polynomials such as addition, multiplication, integer powers, etc.

Usage

## S3 method for class 'clifford'
Ops(e1, e2)
clipfrod_negative(C)
geoprod(C1, C2)
clipfrod_times_scalar(C, x)
clipfrod_plus_clifford(C1, C2)
clipfrod_eq_clifford(C1, C2)
clipfrod_inverse(C)
clipfrod_dotprod(C1, C2)
fatdot(C1, C2)
lefttick(C1, C2)
righttick(C1, C2)
wedge(C1, C2)
scaiprod(C1, C2 = rev(C1), drop = TRUE)
euciprod(C1, C2 = C1, drop = TRUE)
maxyterm(C1, C2 = as.clifford(0))
C1 %*% C2
C1 %**% C2
C1 %%% C2
C1 %star% C2
C1 % % C2
Arguments

e1, e2, C1, C2  Objects of class `clifford`
x  Scalar, length one numeric vector
drop  Boolean, with default `TRUE` meaning to return the constant coerced to numeric, and `FALSE` meaning to return a (constant) Clifford object

Details

The function `Ops.clifford()` passes unary and binary arithmetic operators “+”, “-”, “*”, “/” and “^” to the appropriate specialist function.

Functions `c_foo()` are low-level helper functions that wrap the C code; function `maxyterm()` returns the maximum index in the terms of its arguments.

The package has several binary operators:

- **Geometric product**
  \[ A \ast B = \text{geoprod}(A, B) \]
  \[ AB = \sum_{r,s} \langle A \rangle_r \langle B \rangle_s \]

- **Inner product**
  \[ A \%\% B = \text{cliffdotprod}(A, B) \]
  \[ A \cdot B = \sum_{r \neq 0, s \neq 0} \langle \langle A \rangle_r \langle B \rangle_s \rangle |s-r| \]

- **Outer product**
  \[ A \%\% B = \text{wedge}(A, B) \]
  \[ A \wedge B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle s+r \]

- **Fat dot product**
  \[ A \%o\% B = \text{fatdot}(A, B) \]
  \[ A \bullet B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle |s-r| \]

- **Left contraction**
  \[ A \%\_\% B = \text{lefttick}(A, B) \]
  \[ A | B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle s-r \]

- **Right contraction**
  \[ A \%\_\% B = \text{righttick}(A, B) \]
  \[ A \}_ B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle r-s \]

- **Cross product**
  \[ A \%\% B = \text{cross}(A, B) \]
  \[ A \times B = \frac{1}{2} (AB - BA) \]

- **Scalar product**
  \[ A \%\% B = \text{star}(A, B) \]
  \[ A \ast B = \sum_{r,s} \langle \langle A \rangle_r \langle B \rangle_s \rangle \]

- **Euclidean product**
  \[ A \%\% B = \text{eucprod}(A, B) \]
  \[ A \ast B = A \ast B^\dagger \]

In R idiom, the geometric product `geoprod(., .)` has to be indicated with a “\*” (as in `A*B`) and so the binary operator must be `\%\%`: we need a different idiom for scalar product, which is why `%star%` is used.

Because geometric product is often denoted by juxtaposition, package idiom includes a `\%\% b` for geometric product.

Function `clifford_inverse()` is problematic as nonnull blades always have an inverse; but func-
The scalar product of two clifford objects is defined as the zero-grade component of their geometric product:

\[ A \ast B = \langle AB \rangle_0 \quad \text{NB: notation used by both Perwass and Hestenes} \]

In package idiom the scalar product is given by \( A \%\text{star}\% B \) or \( \text{scalprod}(A, B) \). Hestenes and Perwass both use an asterisk for scalar product as in “\( A \ast B \)”, but in package idiom, the asterisk is reserved for geometric product.

**Note: in the package, \( A \ast B \) is the geometric product.**

The Euclidean product (or Euclidean scalar product) of two clifford objects is defined as

\[ A \ast B = A \ast B^\dagger = \langle AB^\dagger \rangle_0 \quad \text{Perwass} \]

where \( B^\dagger \) denotes Conjugate [as in \( \text{Conj}(a) \)]. In package idiom the Euclidean scalar product is given by \( \text{eucprod}(A, B) \) or \( A \%\text{euc}\% B \), both of which return \( A \ast \text{Conj}(B) \).

Note that the scalar product \( A \ast A \) can be positive or negative [that is, \( A \%\text{star}\% A \) may be any sign], but the Euclidean product is guaranteed to be non-negative [that is, \( A \%\text{euc}\% A \) is always positive or zero].

Dorst defines the left and right contraction (Chisholm calls these the left and right inner product) as \( A \lfloor B \) and \( A \rfloor B \). See the vignette for more details.

**Value**

The high-level functions documented here return an object of \texttt{clifford}. But don’t use the low-level functions.

**Author(s)**

Robin K. S. Hankin

**See Also**

\texttt{scalprod}

**Examples**

```r
u <- rcliff(5)
v <- rcliff(5)
w <- rcliff(5)
u\ast v
u^3
```
\[ u \ast (v \ast w) = (u \ast v) \ast w \] # should be TRUE
\[ u \ast (v \ast w) = (u \ast v) \ast w \] # should be TRUE
\[ u \%\% v = (u \ast v - v \ast u) / 2 \] # should be TRUE

# Now if \( x, y, z \) are _vectors_ we would have:

\[ x <- \text{as.1vector}(5) \]
\[ y <- \text{as.1vector}(5) \]
\[ x \ast y = x \%\% y + x \%\% y \] # should be TRUE
\[ x \%\% y = x \%\% (y + 3 \ast x) \] # should be TRUE

## Inner product is "\%\%" is not associative:
\[ rcliff(5, g=2) \rightarrow x \]
\[ rcliff(5, g=2) \rightarrow y \]
\[ rcliff(5, g=2) \rightarrow z \]
\[ x \%\% (y \%\% z) \]
\[ (x \%\% y) \%\% z \]

## Geometric product *is* associative:
\[ x \ast (y \ast z) \]
\[ (x \ast y) \ast z \]

---

**print**

Print methods for Clifford algebra

### Description

Print methods for Clifford algebra

### Usage

#### S3 method for class 'clifford'

```r
print(x,...)
```

#### S3 method for class 'clifford'

```r
as.character(x,...)
catterm(a)
```

### Arguments

- **x**: Object of class clifford in the print method
- **...**: Further arguments, currently ignored
- **a**: Integer vector representing a term
Note

The print method does not change the internal representation of a clifford object, which is a two-

element list, the first of which is a list of integer vectors representing terms, and the second is a
numeric vector of coefficients.

The print method has special dispensation for length-zero clifford objects. It is sensitive to the value
of options("separate") which, if TRUE prints the basis vectors separately and otherwise prints in
a compact form. The indices of the basis vectors are separated with option("basissep") which
is usually NULL but if $n > 9$, then setting options("basissep" = ",") might look good.

Function as.character.clifford() is also sensitive to these options.

Function catterm() is a low-level helper function.

Author(s)

Robin K. S. Hankin

See Also

classical

Examples

rcliff()  # fine
rcliff(d=15)  # incomprehensible

options("separate" = TRUE)
rcliff(d=15)  # incomprehensible

options("separate" = FALSE)

Description

Functionality for converting quaternions to and from Clifford objects.

Usage

quaternion_to_clifford(Q)
clifford_to_quaternion(C)
Arguments

- C: Clifford object
- Q: Quaternion

Details

Given a quaternion \( a + bi + cj + dk \), one may identify \( i \) with \(-e_{12}\), \( j \) with \(-e_{13}\), and \( k \) with \(-e_{23}\) (the constant term is of course \( e_0 \)).

The functions documented here convert from quaternions to clifford objects and vice-versa.

Author(s)

Robin K. S. Hankin

Examples

```r
x1 <- clifford(list(numeric(0),c(1,2),c(1,3),c(2,3)),1:4)
clifford_to_quaternion(x1)
```

```r
# Following needs the onion package (it is discouraged to load both):
# library("onion")
# Q1 <- rquat(1)
# Q2 <- rquat(1)
# LHS <- clifford_to_quaternion(quaternion_to_clifford(Q1) * quaternion_to_clifford(Q2))
# RHS <- Q1*Q2
# LHS - RHS # zero to numerical precision
```

---

**rcliff**

*Random clifford objects*

**Description**

Random Clifford algebra elements, intended as quick “get you going” examples of clifford objects.

**Usage**

```r
rcliff(n=9, d=6, g=4, include.fewer=TRUE)
rblade(d=9, g=4)
```
Arguments

- **n**: Number of terms
- **d**: Dimensionality of underlying vector space
- **g**: Maximum grade of any term
- **include.fewer**: Boolean, with FALSE meaning to return a clifford object comprising only terms of grade \( g \), and default TRUE meaning to include terms with grades less than \( g \)

Details

Perwass gives the following lemma:

Given blades \( A_{(r)} \), \( B_{(s)} \), \( C_{(t)} \), then

\[
\langle A_{(r)} B_{(s)} C_{(t)} \rangle_0 = \langle C_{(t)} A_{(r)} B_{(s)} \rangle_0
\]

In the proof he notes in an intermediate step that

\[
\langle A_{(r)} B_{(s)} \rangle_t^* C_{(t)} = C_{(t)}^* \langle A_{(r)} B_{(s)} \rangle_t = \langle C_{(t)} A_{(r)} B_{(s)} \rangle_0.
\]

Package idiom is shown in the examples.

Author(s)

Robin K. S. Hankin

Examples

```r
rcliff()
rcliff(d=3,g=2)
rcliff(3,10,7)
rcliff(3,10,7,include=TRUE)

x1 <- rcliff()
x2 <- rcliff()
x3 <- rcliff()
x1*(x2*x3) == (x1*x2)*x3  # should be TRUE

rblade()

# We can invert blades easily:
a <- rblade()
ainv <- rev(a)/scalprod(a)
zap(a*ainv)  # should be 1
zap(ainv*a)  # should be 1
```
# Perwass 2009, lemma 3.9:

\[ A \leftarrow \text{rblade}(g=4) \quad \# r=4 \]
\[ B \leftarrow \text{rblade}(g=5) \quad \# s=5 \]
\[ C \leftarrow \text{rblade}(g=6) \quad \# t=6 \]

\[
\text{grade}(A \times B \times C, 0) - \text{grade}(C \times A \times B, 0) \quad \# \text{geometric product uses 'x'}
\]

## Intermediate step

\[ x_1 \leftarrow \text{grade}(A \times B, 7) \star C \]
\[ x_2 \leftarrow C \star \text{grade}(A \times B, 7) \]
\[ x_3 \leftarrow \text{grade}(C \times A \times B, 0) \]

\[ \max(x_1, x_2, x_3) - \min(x_1, x_2, x_3) \quad \# \text{should be small} \]

---

**rev**

Reverse of a Clifford object

**Description**

The “reverse” of a term is simply the basis vectors written in reverse order; this changes the sign of the term if the number of basis vectors is 2 or 3 (modulo 4). Taking the reverse is a linear operation. Both Hestenes and Chisholm use a dagger to denote the reverse of \( A \), as in \( A^\dagger \). But both Perwass and Dorst use a tilde, as in \( 
\tilde{A} \).

\[
(A^\dagger)^\dagger = A \quad (AB)^\dagger = B^\dagger A^\dagger \quad (A + B)^\dagger = A^\dagger + B^\dagger \quad \langle A^\dagger \rangle = \langle A \rangle
\]

where \( \langle A \rangle \) is the grade operator; and it is easy to prove that

\[
\langle A^\dagger \rangle_r = \langle A \rangle^\dagger_r = (-1)^{r(r-1)/2} \langle A \rangle_r
\]

We can also show that

\[
\langle AB \rangle_r = (-1)^{r(r-1)/2} \langle B^\dagger A^\dagger \rangle_r
\]

**Usage**

```r
## S3 method for class 'clifford'
rev(x)
```

**Arguments**

- **x** Clifford object
Author(s)

Robin K. S. Hankin

See Also

g
de

Examples

x <- rcliff()
rev(x)

A <- rblade(g=3)
B <- rblade(g=4)
rev(A %*% B) == rev(B) %*% rev(A)  # should be small
rev(A * B) == rev(B) * rev(A)      # should be small

signature

The signature of the Clifford algebra

Description

Getting and setting the signature of the Clifford algebra

Usage

signature(s)
is_ok_sig(s)
mymax(s)

Arguments

s

Integer, specifying number of positive elements on the diagonal of the quadratic form

Details

The function is modelled on `lorentz::sol()` which gets and sets the speed of light.
Clifford algebras require a bilinear form on $R^n \langle \cdot, \cdot \rangle$, usually written

$$\langle x, x \rangle = x_1^2 + x_2^2 + \cdots + x_p^2 - x_{p+1}^2 - \cdots - x_{p+q}^2$$

where $p + q = n$. With this quadratic form the vector space is denoted $R^{p,q}$, and we say that $p$ is
the signature of the bilinear form $\langle \cdot, \cdot \rangle$. This gives rise to the Clifford algebra $C_{p,q}$. 
If the quadratic form is positive-definite, package idiom is to use the default special value $p = 0$
(which means that zero entries on the main diagonal are negative).
Specifying a negative value for $p$ sets the quadratic form to be identically zero, reducing the geometric product to the exterior wedge product and thus a Grassman algebra. But use the `wedge` package for this, which is much more efficient and uses nicer idiom.
Function `is_ok_sig()` is a helper function that checks for a proper signature.
Function `mymax()` is a helper function that avoids warnings from `max()` when given an empty argument.

**Author(s)**
Robin K. S. Hankin

**Examples**

```r

e1 <- clifford(list(1),1)
e2 <- clifford(list(2),1)

signature()

e1*e1
e2*e2

signature(1)
e1*e1
e2*e2  #note sign

signature(Inf)
e2*e2
```

**Summary.clifford**  
*Summary methods for clifford objects*

**Description**  
Summary method for clifford objects, and a print method for summaries.

**Usage**

```r
## S3 method for class 'clifford'
summary(object, ...)
## S3 method for class 'summary.clifford'
print(x, ...)
first_n_last(x)
```
Arguments

object, x  Object of class clifford
...  Further arguments, currently ignored

Details

Summary of a clifford object. Note carefully that the “typical terms” are implementation specific. Function first_n_last() is a helper function.

Author(s)

Robin K. S. Hankin

See Also

print

Examples

summary(rcliff())

term  Deal with terms

Description

By basis vector, I mean one of the basis vectors of the underlying vector space $\mathbb{R}^n$, that is, an element of the set $\{e_1, \ldots, e_n\}$. A term (sometimes a basis blade or simple blade) is a wedge product of basis vectors (or a geometric product of linearly independent basis vectors), something like $e_{12}$ or $e_{12569}$.

From Perwass: a blade is the outer product of a number of 1-vectors (or, equivalently, the wedge product of linearly independent 1-vectors). Thus $e_{12} = e_1 \wedge e_2$ and $e_{12} + e_{13} = e_1 \wedge (e_2 + e_3)$ are blades, but $e_{12} + e_{34}$ is not.

Function is.blade() is not currently implemented: there is no easy way to detect whether a Clifford object is a product of 1-vectors.

Usage

terms(x)
is.blade(x)
is.basisblade(x)

Arguments

x  Object of class clifford
Details

- Functions `terms()` and `coeffs()` are the extraction methods. These are unordered vectors but the ordering is consistent between them (an extended discussion of this phenomenon is presented in the `mvp` package).
- Function `term()` returns clifford object that comprises a single term with unit coefficient.
- Function `is.basisterm()` returns TRUE if its argument has only a single term, or is a nonzero scalar; the zero clifford object is not considered to be a basis term.

Author(s)

Robin K. S. Hankin

References


See Also

`clifford`

Examples

```r
x <- rcliff()
terms(x)

is.basisblade(x)

a <- as.1vector(1:3)
b <- as.1vector(c(0,0,0,12,13))
a %*% b # a blade
```

---

**zap**

Zap small values in a clifford object

Description

Generic version of `zapsmall()`

Usage

```r
zap(x, drop=TRUE, digits = getOption("digits"))
```
Arguments

- x: Clifford object
- drop: Boolean with default TRUE meaning to coerce the output to numeric with drop()
- digits: number of digits to retain

Details

Given a clifford object, coefficients close to zero are ‘zapped’, i.e., replaced by ‘0’ in much the same way as base::zapsmall().

The function should be called zapsmall(), and dispatch to the appropriate base function, but I could not figure out how to do this with S3 (the docs were singularly unhelpful) and gave up.

Note, this function actually changes the numeric value, it is not just a print method.

Author(s)

Robin K. S. Hankin

Examples

```r
a <- clifford(sapply(1:10,seq_len),90^-(1:10))
azap(a)
options(digits=3)
azap(a)

a-zap(a)  # nonzero

B <- rblade(g=3)
mB <- B*rev(B)
zap(mB)
drop(mB)
```

zero

The zero Clifford object

Description

Dealing with the zero Clifford object presents particular challenges. Some of the methods need special dispensation for the zero object.

Usage

```r
is.zero(C)
```

Arguments

- C: Clifford object
Details
To create the zero object *ab initio*, use
\[
\text{clifford(list(), numeric(0))}
\]
although note that \text{scalar}(0) will work too.

Author(s)
Robin K. S. Hankin

See Also
\text{scalar}

Examples
\[
\text{is.zero(rcliff())}
\]
Index

*Topic **math**
  - summary.clifford, 30
*Topic **package**
  - clifford-package, 2
  - [.clifford (Extract.clifford), 12
  - [<-.clifford (Extract.clifford), 12
  - % % (Ops.clifford), 21
  - %.% (Ops.clifford), 21
  - %X% (Ops.clifford), 21
  - %%^% (Ops.clifford), 21
  - %euc% (Ops.clifford), 21
  - %o% (Ops.clifford), 21
  - %star% (Ops.clifford), 21
  - allcliff, 4
  - antivector, 5
  - as.1vector (numeric_to_clifford), 20
  - as.character (print), 24
  - as.clifford (clifford), 7
  - as.cliffvector (numeric_to_clifford), 20
  - as.pseudoscalar (numeric_to_clifford),
    - 20
  - as.scalar (numeric_to_clifford), 20
  - as.vector, 6
  - basis (numeric_to_clifford), 20
  - c_add (lowlevel), 17
  - c_equal (lowlevel), 17
  - c_fatdotprod (lowlevel), 17
  - c_getcoeffs (lowlevel), 17
  - c_grade (lowlevel), 17
  - c_identity (lowlevel), 17
  - c_innerprod (lowlevel), 17
  - c_lefttickprod (lowlevel), 17
  - c_multiply (lowlevel), 17
  - c_outerprod (lowlevel), 17
  - c_overwrite (lowlevel), 17
  - c_power (lowlevel), 17
  - c_righttickprod (lowlevel), 17
  - catterm (print), 24
  - cliffdotprod (Ops.clifford), 21
  - clifford, 4, 7, 13, 14, 25, 32
  - clifford-package, 2
  - clifford_cross_clifford (Ops.clifford),
    - 21
  - clifford_dot_clifford (Ops.clifford), 21
  - clifford_eq_clifford (Ops.clifford), 21
  - clifford_fatdot_clifford (Ops.clifford), 21
  - clifford_inverse (Ops.clifford), 21
  - clifford_lefttick_clifford (Ops.clifford), 21
  - clifford_negative (Ops.clifford), 21
  - clifford_plus_clifford (Ops.clifford),
    - 21
  - clifford_plus_numeric (Ops.clifford), 21
  - clifford_plus_scalar (Ops.clifford), 21
  - clifford_power_scalar (Ops.clifford), 21
  - clifford_righttick_clifford (Ops.clifford), 21
  - clifford_star_clifford (Ops.clifford),
    - 21
  - clifford_times_clifford (Ops.clifford),
    - 21
  - clifford_times_scalar (Ops.clifford), 21
  - clifford_to_quaternion (quaternion), 25
  - clifford_wedge_clifford (Ops.clifford),
    - 21
  - coe (Extract.clifford), 12
  - coe (Extract.clifford), 12
  - Conj, 8, 18, 19, 29
  - const (getcoeffs), 14
  - const (getcoeffs), 14
  - constant (getcoeffs), 14
  - constant (getcoeffs), 14
  - cross (Ops.clifford), 21
  - dagger (Conj), 8
  - dot (Ops.clifford), 21
INDEX

drop, 10
dual, 11
e (numeric_to_clifford), 20
euclid_product (Ops.clifford), 21
euclidean_product (Ops.clifford), 21
eucprod (Ops.clifford), 21
even, 12
evenpart (even), 12
extract (Extract.clifford), 12
Extract.clifford, 12
fatdot (Ops.clifford), 21
first_n_last (summary.clifford), 30
geometric_prod (Ops.clifford), 21
geometric_product (Ops.clifford), 21
groeprod (Ops.clifford), 21
gtcoeffs, 10, 14, 21
grade, 9, 10, 12, 15, 29
grades (grade), 15
gradesminus (grade), 15
gradesplus (grade), 15
homog, 16
homogeneous (homog), 16
is.1vector (numeric_to_clifford), 20
is.antivector (antivector), 5
is.basisblade (term), 31
is.blade (term), 31
is.clifford (clifford), 7
is.even (even), 12
is.homog (homog), 16
is.homogenous (homog), 16
is.odd (even), 12
is.pseudoscalar (numeric_to_clifford), 20
is.scalar (numeric_to_clifford), 20
is.term (term), 31
is.zero (zero), 33
is_ok_clifford (clifford), 7
is_ok_sig (signature), 29
lefttick (Ops.clifford), 21
lowlevel, 17
magnitude, 18
maxyterm (Ops.clifford), 21
Mod (magnitude), 18
mod (magnitude), 18
Mod.clifford (magnitude), 18
mymax (signature), 29
nbits (clifford), 7
neg, 19
nterms (clifford), 7
numeric_to_clifford, 6, 20
oddpart (even), 12
Ops (Ops.clifford), 21
Ops.clifford, 8, 13, 17, 18, 21
print, 24, 31
print.clifford (print), 24
print.summary.clifford
  (summary.clifford), 30
pseudoscalar, 11
pseudoscalar (numeric_to_clifford), 20
quaternion, 25
quaternion_to_clifford (quaternion), 25
rblade (rcliff), 26
rcliff, 18, 26
replace (Extract.clifford), 12
rev, 9, 28
righttick (Ops.clifford), 21
scalar, 34
scalar (numeric_to_clifford), 20
scalar_product (Ops.clifford), 21
scalprod, 23
scalprod (Ops.clifford), 21
signature, 29
star (Ops.clifford), 21
summary.clifford, 30
term, 13, 31
terms (term), 31
wedge (Ops.clifford), 21
zap, 32
zapsmall (zap), 32
zaptiny (zap), 32
zero, 33